Conceptual and Constraint Similarity in Component Matching

A thesis
submitted in partial fulfillment of the requirements
for the award of degree
of

Master of Engineering
in
Software Engineering

Under the Supervision of
Mr. Rajesh Kumar Bhatia
Assistant Professor
CSED, TIET, Patiala.

Submitted By
Rajiv Bammi
(Roll No: 8043117)

Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
(Deemed University)
Patiala-147004

May 2006

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Conceptual and Constraint Similarity in Component Matching",** in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering at Computer Science & Engineering Department of Thapar Institute of Engineering & Technology (Deemed University), Patiala, is an authentic record of my own work carried out under the supervision and guidance of Mr. Rajesh Kumar 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.

Rajiv Bammi

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

(Mr. Rajesh Kumar Bhatia)
Assistant Professor
Computer Science & Engineering Department
Thapar Institute of Engineering & Technology
Patiala- 147004

Countersigned By:

(Dr. (Mrs.) Seema Bawa)
(Professor & Head)
Computer Science & Engineering. Department
Thapar Institute of Engg and Tech.,
Patiala.

(Dr. T.P. Singh)
(Dean)
Academic Affairs
Thapar Institute of Engg and Tech
Patiala.

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 this thesis work. I owe him lots of gratitude for having me shown this way of research.

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

No thesis work could be written without being influenced by the thoughts of others, I would like to thank my colleagues Kapil Bhatia, Amarjit Manjotra and Rockey Goel who are always there at the hour of need and provided with all the help and support, which I needed. I would also like to express heartfelt thanks to my parents for their well wishing, motivation and support.

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

Rajiv Bammi
(8043117)

Abstract

Component reuse improves both the quality and productivity of software development. Before Software components can be reused they must be located from component repository. Reuse repository system provides an effective means to locate software components which can be reused to solve the problem in hand. As a result programmer need not to always begin from the scratch. This not only leads to good quality software but also reduces the development and testing time and effort.

This thesis has resulted in a fully functional reuse repository system with effective and easy to use search functionality so that needed reusable components can be easily extracted from repository. The constructed system has an easy to use interface, due to Web-browser based front-end. Concept similarity and Constraint compatibility has been used for searching the relevant components.

For Concept similarity, an Information Retrieval *Vector Space Model* with stemming algorithm, stop word removal and thesauruses has been implemented. Constraint compatibility is computed by *signature matching.*

To select the components from repository and compute the relevancy of components, system also considers the user experiences. For this purpose, system allows the user to assign the weight to components so that this knowledge can be used for effectively processing future queries. So more the system interact with user, more experience it will gain and as a result it makes the system more intelligent and increases its capability to return more useful and relevant reusable components.

Table of Contents

List of Figures

List of Tables

List of Tables

Chapter 1

Introduction

A wide gap exists between the constantly increasing demands for complex software systems and the capability of the software industry to deliver quality software systems in a timely and cost-effective manner. Software reuse, a development method of using existing reusable software components to create new programs, has been shown through empirical studies to improve both the quality and productivity of software development [3]. Software reuse increases the evolvability of software systems because complex systems evolve faster when they are built from stable subsystems [57].

1.1 Motivation

Software Reuse is not so simple as it sounds. It takes a lot of time and effort to locate components that are required to solve the current problem. If developer is not aware of the existence of a required and usable component he or she is unfortunately disinclined to make the effort of searching components [60]. This lead to failure of reuse process and as a result developer will have to write everything from the scratch. In this case the quality of solution greatly depends on the experience of developers and in most cases the quality of solution will not be as good as build with existing components. Still having a collection of components is not of much help if there is no interactive and efficient search functionality to help the developers in locating the reusable components.

Furthermore if such interactive system is unable to locate the required components within first few attempts the user is unlikely to return. This means the system has to handle a diverse range of user queries and also take into account the mental perception that developers mostly use to describe the components.

As such a reuse repository system may cover many different domains, some of returned components may not be relevant to the current problem. When the results of a query have been properly categorized and also gives the relevancy of components with respect to

query, the user should be able to quickly select the appropriate set of components very easily.

Moreover if system take the feedback from the user related to result it returned and use this knowledge in processing future queries then the system will become self learner and based on this automated learning, system can return more relevant components that satisfy the need of the user

This leads us to the problem description which will guide this work in the thesis.

1.2 Goal of the research / Problem description

This is the concrete description of what will be solved in this thesis. This is an empirical project where a reuse repository system will be constructed. This thesis report will answer the following questions:

- How can a reuse repository system be created, which supports efficient and relevant searching of software components?
- How can basic problem such as searching for any component not matching exactly but performing the similar functionality can be solved?
- How can results of a search be presented to the users so that they can select best components that satisfy their requirements?
- How Graphical User Interface (GUI) of the reuse repository system should be created so that user can easily specify their requirements?
- How user response can be gathered to make the system more intelligent and self-learning so that system can use this knowledge in processing future queries and can effectively deliver the required components to the user.

Our objective is to make a Reuse Repository System with effective and optimum search functionality along with easy to use user interface. The resulting Reuse Repository system should be easily extensible and adaptable to specific problem domain.

1.3 Brief System Description

In order to satisfy the requirements of problem description a functional prototype will be created. The user can access the system using any web browser which means reuse repository system can be easily used by several users concurrently regardless of their location. Repository will be stored at a central location and user can access them from any of the nodes that are connecting to that central location (Server).

System will first perform indexing of reusable software components. For this purpose Indexer module will create the component repository from existing collection of components. It will create and index and make it available for search.

In order to ensure better search results than simple exact text matching, a method from Information Retrieval called Vector Space Model will be used. Stemming algorithm is also applied to reduce the stemming effect of words. Thesauruses are also applied to make the search more efficient.

This reuse repository System will use both concept matching and signature matching to retrieve the components from the repository.

After retrieving the components from repository, system will present the ranked results to the user according to their relevancy to the query specified. System also allows the user to assign the weight to any particular component so that this knowledge can be used for effectively processing future queries. This Reuse Repository system will be discussed further in Chapter 4.

1.4 Organization of the Dissertation

Chapter 2 gives background information related to this thesis. This chapter also introduces the various technologies used in this thesis. This chapter first gives a brief overview of software reuse followed by description of information retrieval and its methods. After that it gives details description of vector space model and finally stop-words, stemming and thesauri are discussed.

Chapters 3 summarize the software reuse research and discuss major research contributions in this area.

Chapter 4 first describes the problem statement which is used as an objective and motivation for this thesis. After that it describe the proposed reuse repository system and its implementation details.

This discussion is completed with the conclusion in Chapter 5, which also contains the final thoughts on future developments of the system.

Chapter 2

Background: Concepts and Technology used

This section provides background information related to our thesis. This chapter also introduces the various technologies used in this thesis. The chapter is used as a reference when the implementation and experiments are described in later chapters. We start with explaining the principles of Software reuse, its benefits, limitations, Level of reuse and software reuse process. After that software paradigms and storage and retrieval methods are defined.

Section 2.2 describes the information retrieval and various strategies of information retrieval. Then Sub-Section 2.2.2 describes the vector space model. Finally stopwords, stemming and thesauri are discussed.

2.1 Software Reuse: Introduction

Software development cannot possibly become an engineering discipline so long as it has not perfected a technology for developing products from reusable assets in a routine manner. As discipline software reuse must define and promote the managerial, organizational and technical standards required to achieve this goal.

According to Milli, Software reuse is the process whereby an organization defines a set of systematic operating procedures to specify, produce, classify, retrieve and adapt software artifacts for the purpose of using them in its development activities.

“Software reuse is the use of existing components of source code to develop new software programs or applications. “ [47]

“Software reuse is re-application of knowledge about one system to another similar system in order to reduce the effort of development and maintenance of similar system.“ [20].

A good software reuse process facilitates the increase of productivity, quality, and reliability, and decrease of costs and implementation time. An initial investment is required to start a software reuse process, but that investment pays for itself in a few reuses.

Software reusability is an attribute of any good quality software and it can be defined as:

Reusability= usability+ usefulness

Where as:

Usability= Usability is the degree to which an asset is easy to use. (independent of functionality)

Usefulness= Usefulness is the frequency of suitability for use. (independent of packaging)

2.1.1 Software Reuse: Potentials and Pitfalls

Like any other engineering discipline, software engineering stands to gain a great deal from a sound discipline of reuse. But unlike most other engineering disciplines, software reuse does not arise naturally in software engineering and when it does it does not come with cost and risks.

Software reuse benefits can be classified into three categories:

- **Gains in productivity:** Gains in productivity are the main motivation for software reuse. By reusing existing assets, we can save the man power required to develop them again.

- **Gains in quality:** When a component is developed for reuse, one can rationalize large investment on quality on the premises that these investments will be amortized over its multiple uses.

- **Gains in development schedule**: By using reusable assets, we save not only the man power (person-month) but also on development schedule (month).

For all its promises, Software reuse has not been a matter of routine practice for a number of reasons:

- **Limited Reuse Potential:** Software assets are very information rich. This limits the likelihood of a match between a specific query and available asset. Modern programming disciplines which emphasis on information hiding and modern programming languages which support information hiding, help us control this problem but do not over come it entirely.
- **Non-negligible Overheads:** It is easy to be misled by the reuse propaganda and to overlook the nontrivial costs associated with software reuse. But in reality even under the best conditions the margin of benefits of software reuse is fairly limited.
- **Non-negligible Risks:** It is also possible for a poorly planned reuse initiative to actually cost more than it saves.
- **Non-negligible Obstacles:** The introduction of reuse requires rather profound changes in the operational procedures of an organization. These changes are usually met with resistance. Also reuse requires healthy investments and does not bring returns immediately. Hence it requires long-term managerial support.

In the light of this background, one must remember that software reuse is no panacea. It requires careful planning, realistic expectations and a long-term perspective.

2.1.2 Software Reusable Assets

A software artifact can be defined as a piece of formalized knowledge that can contribute to the software development process. There are two types of software artifacts:

- Software products that are created as "things" or deliverables during the development process.
- Development knowledge that is applied to the process.

The most commonly reused software product is source code, which is the final and most important product of software development. In addition to code, any intermediate life cycle products can be reused which means that software developers can pursue the reuse of requirement documents, system specifications, modular designs, test plans, test cases, and documentation in various stages of software development.

2.1.3 Levels of Reuse

Reuse of software can be considered at different levels:

- **Application system reuse**: The whole of an application system may be reused. The key problem here is ensuring that the software is portable and it can easily execute on several platforms.
- **Subsystem reuse:** Major subsystem of an application may be reused. For example a pattern matching system developed as part of a text processing system may be reused in a database management system.
- **Module or Object reuse:** Components of a system representing a collection of functions may be reused. For example an Ada package or a c++ object implementing a binary tree may be reused in different platforms.
- **Function reuse:** Software component, which implements a single function such as mathematical function, may be reused.

2.1.4 Software Reuse Process

It is possible to identify the three stakeholders in the process of software reuse. Software reuse process can be defined by reviewing in turn the activities of these stakeholders. These stakeholders and activities perform by them are:

- *Cooperate Management:* They initiate the reuse initiative and monitor the attending costs and benefits.
- *Domain Engineering Management:* They are responsible for producing, classifying and maintaining reusable assets.
- *Application engineering Management:* They are responsible for producing applications possible using reusable assets.

So reuse involves the following major activities:

- ***Selection:*** Reusable description fragments are held in a reuse library, catalogued by their structural and behavioral characteristics; selection involves identifying potentially applicable fragments through either query-based search or browsing.

- ***Adaptation**:* The selected fragments may need to be modified to meet the current project's requirements; adaptation involves customization, tuning or extension to meet the needs of the current project.

- ***Assembly**:* Once the fragments have been retrieved or created, they have to be composed into a full system description; assembly involves editing and linking the fragments as well as developing additional fragments needed to produce a unified system description.

- ***Cataloguing:*** The new fragments and the assembled system description are themselves potentially reusable on future projects and therefore should be catalogued into the reuse library; cataloguing involves classification and storage of the new fragments as well as library maintenance operations needed to make the information retrievable through the library access mechanisms.

- ***Assessment**:* All of the activities must be guided by the ultimate goal of obtaining a suitable description of the required system; assessment involves validation and verification needed to guide and converge the other activities.

2.1.5 Reuse Libraries: Vertical versus Horizontal

All the reusable assets artifacts are stored in a repository called **"Software Reusable Repository".** In order to be useful, a reuse library must have a simple characterization of assets that is included there in. In order to make effective use of reuse library, user must have clear understanding of its contents, so as to determine whether his needs are likely to be adequately met by library.

Characteristic of reuse library depends on the volume of code of a typical application which can be divided into three categories:

- ***Generic components:*** Which provide general-purpose programming support on top of the programming language. Such components include abstract data types, graphics utilities, mathematical routines and menu drivers. These components are

typically useful across application domains and are known to account for up to 20% of the size of a typical application.

- ***Domain specific components****:* Which fulfill functions that are specific to the application domain of the software product. If the domain is say data processing such components may include sorting packages, file management packages and hashing functions. If the domain is system programming such components may include device handlers, resource managers and specialized priority queue handlers. These components are typically specific to the application domain in hand and are known to account for up to 65% of a typical application.
- ***Application specific code****:* Which servers the purpose of the specific application at hand and can hardly be useful across applications. Application specific code may perform such function as customizing domains specific components or providing specific functionality. Typically application –specific code is not expected to be useful in applications other than those in which it is written and is known to account for up to 15% of the size of a typical application.

On the basis of this observation there are two possible characteristic of reusable library:

(1) Horizontal definition: We may want to build a library that contains generic components which augment the expressive power of the programming language without committing to a specific application domain and hence potentially several development projects. The drawback of such a library is that its leverage is limited to about 20% of the size of the application.

(2) Vertical definition: We may want to build a library that contains domain-specific components. The advantage of such a library is that it can be used only with in a single application domain and hence potentially few projects. Vertical libraries are particularly useful for organizations that develop and maintain large-scale, long-lived software products. The Contrast between vertical and horizontal libraries and their role in building described Figure2.1. The X-axis represents the component genericity and Y-axis represents the component functionality.

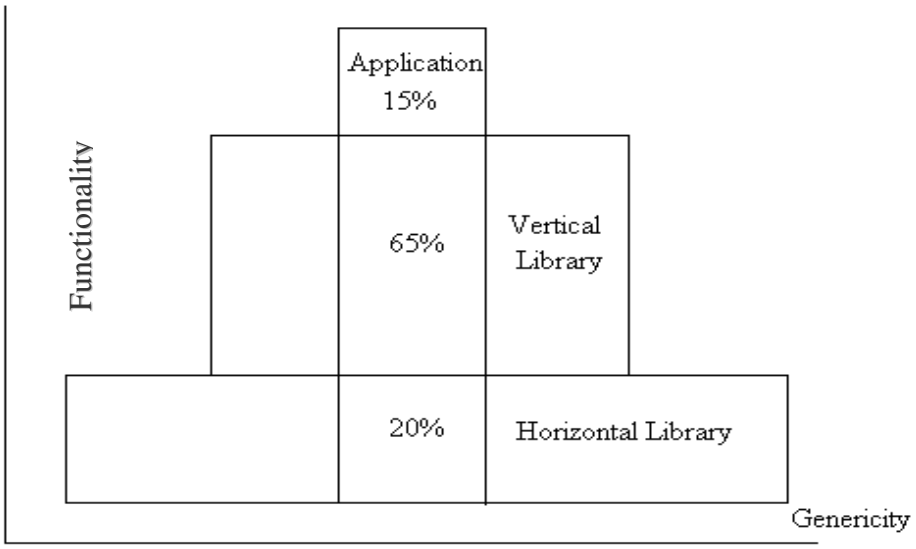

Figure 2.1: Horizontal vs. Vertical Library

A horizontal library offers great domain variation but limited functionality where as a vertical library offers more functionality but limited domain variation. An application cuts across both libraries, borrowing much more from domain-specific vertical library than from the generic horizontal library. An application that falls outside the application domain would use little reusable code.

2.1.6 Software Reuse Paradigms

The Process of software reuse can be applied in variety of paradigms. These paradigms are software retrieval, software adaptation and software composition.

2.1.6.1 Paradigms for Software Retrieval

It is possible to distinguish between two paradigms for using a software library: Browsing and Retrieval. These can be contrasted as follows:

- ***Browsing:*** consist of navigating the software library to acquaint oneself with its holding. Browsing is performed before product design at a time when designers have no precise definition of what they need – but do have some idea of what kinds of assets may be useful.

- ***Retrieval:*** consist of navigating the library to find assets that satisfy pre-specified requirements. Retrieval is performed after product design when the designer has clearly specified requirements to fulfill.

Retrieval methods in turn can be divided into two classes:

(a) Method of exact retrieval: where by we seek to identify library assets that precisely fulfill our requirements. Exact retrieval fits in the life cycle of *black box reuse*.

(b) Method of approximate retrieval*:* where by we seek to identify library assets that almost fulfill all the requirements or fulfill almost all the requirements.

2.1.6.2 Paradigms for Software Adaptation

It is common to recognize two distinct patterns of software reuse which differ mostly by their adaptation step. These patterns are:

- **Black box reuse***:* where by retrieved assets are integrated into host systems verbatim without modification. Black-box reuse does not require the user know about the design or implementation details of assets – only its function and its invocation protocols. We consider as black-box reuse instances where the user must set some parameters as part of the invocation protocols.
- **White box reuse:** whereby retrieved assets are analyzed and modified before being integrated into a host system, because they do not necessarily satisfy the requirements of the host system. White box reuse does require that the user analyze and understand the implementation details of the assets

Component adaptation activity performed under black box reuse as *instantiation* and activities performed under white box reuse as *modification*.

Generally speaking, black box reuse occurs less often but produces greater benefits; where as white-box reuse occurs more often but has smaller benefits margins.

2.1.6.3 Paradigms for Software Composition

It is possible to distinguish between two paradigms of reuse based software development, depending on the nature of reusable software assets:

- ***Compositional Development****:* Compositional development is applicable whenever the reusable assets are finished software products in compliable and executable form and consists in composing these assets to produce larger software systems.
- ***Generative Development:*** Generative development is applicable whenever the reusable assets are represented *intensively* in terms of instanciable *patterns* rather than as finished products. Generative development consists then of instantiating the pattern that embodies the reusable asset by providing specification-level parameters. A generator uses those specification-level parameters to generate a usage specific instance of the pattern, possible leaving stubs to be filled out by hand in an application-specific way.

2.1.7 Characterizing Storage and Retrieval Methods

Methods for the storage and retrieval of software assets in software libraries abound in the literature. In order to understand these methods and have a sound basis for classifying them each method is characterize by a number of orthogonal attributes [35]. These attributes are:

1. Nature of the asset: The most important feature of a software library is, of course the nature of assets that are stored therein. The most typical asset is code but other kinds of assets are also possible: specifications, designs, test data, documentation, etc. Some library methods are restrictive, in the sense that they work for a single kind of asset, whereas others may work for a wide range of assets.

2. Scope of the library*:* Another crucial feature of a software library is the scope of the library: whether the library is expected to be used within a single project, within an organization, or on a larger scale. In order to use a software library effectively, a reuser must share some common knowledge with the maintainer of the library (pertaining, e.g., to the interpretation of terminology, the representation of assets, the form and meaning of error messages, etc.), and with other users.

3. Query representation: A software library can be characterized by the form that queries submitted to the library must take. Among possible options there can be: a formal functional specification, a signature specification, a behavioral sample, a natural language query, or a set of keywords.

4. Asset representation: The representation of assets is an important feature of a library, not only because it dictates what form user queries take, but also because it determines how retrieval is performed. In a perfectly transparent library, the representation of an asset is irrelevant to the user, but no library is perfectly transparent. The possible values of this attribute can be: formal specifications, signature specifications, set of keywords, the source text, the executable code, and requirements documentation.

5. Storage structure: The most common logical storage structure in software libraries is "no structure at all". These means software assets are stored side by side with no ordering between them. While in traditional database systems entries are ordered by their identifying keys, it is difficult to define a general key that can be used to order software assets in a meaningful way.

6. Navigation scheme: This attribute is correlated to *storage structure*, because the storage structure determines to a large extent the navigation scheme of the method. In flat of all the entries. storage structures, only possible pattern of navigation is brute force exhaustive search

7. Retrieval goal: In principle, the goal of a retrieval operation is to find one or several programs that are correct with respect to a given query. If this retrieval operation fails to turn up candidate programs, one may want to perform another retrieval operation, with the lesser ambition of finding programs that approximate the query, with the expectation that we must modify the retrieved programs.

8. Relevance criterion: The relevance criterion defines under what condition a library asset is considered to be relevant for the submitted query with respect to the predefined retrieval goal.

Attributes	Characterization
Nature of asset	Source code, executable code, requirements specification, design description, test data, documentation, proof.
Scope of library	Within a project, across a program, across a product line, across multiple product lines, organization-wide world-wide.
Query representation	Functional specification, signature specification, keyword list, design pattern, behavioral sample.
Asset representation	Functional specification, signature specification, source code, executable code, requirements documentation, keywords.
Storage structure	Flat structure, hypertext links, refinement ordering, ordering by genericity.
Navigation scheme	Exhaustive linear scan, navigating hypertext links, navigating refinement relations.
Retrieval goal	Correctness, functional proximity, structural proximity.
Relevance criterion	Correctness, signature matching, minimizing functional distance, minimizing structural distance.
Matching criterion	Correctness formula, Signature identity, Signature refinement, Equality/Subsumption of keywords, Natural language analysis, Pattern recognition.

Figure 2.2: Attributes of a software library [35]

9. Matching criterion: The matching criterion is the condition that we choose to check between the submitted query and a candidate library asset to decide whether the asset is relevant. Ideally the matching criterion should be equivalent to the relevance criterion, but it is not always so. If the asset's surrogate is too abstract or the relevance criterion is too intractable, these two criteria may differ significantly.

Figure 2.2 describes a table of all the attributes discussed above along with tentative indication of value that each attribute can take.

2.2 Information Retrieval

Information retrieval is devoted to finding relevant documents not finding simple matches to patterns. Normally there are two document categories correspond to any issued query. Namely in the collection there are documents which are retrieved and there are those documents which are relevant. In a perfect system these two sets would be equivalent.

To measure effectiveness, two ratios are used: Precision and recall.

- Precision is the ratio of the number of relevant documents retrieved to the total number retrieved. Precision provides an indication of the quality of the answer set.
- Recall considers the total number of retrieved documents. It is the ratio of the number if relevant documents retrieved to the total numbers of documents in the collection that are believed to be relevant.

A typical precision/ recall curve is shown in Figure 2.3.

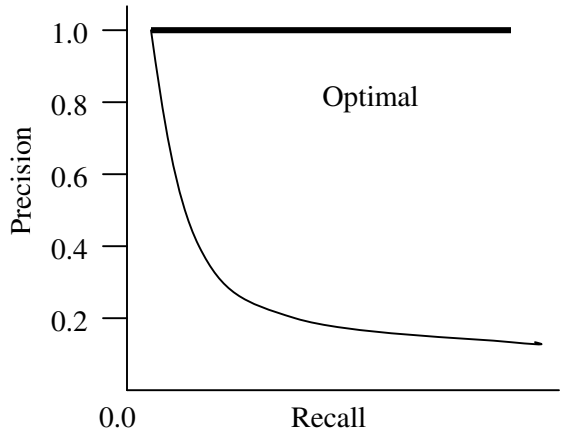

Figure 2.3: Typical and optimal Precision/Recall Graph

Typically as higher recall is desired, more documents must be retrieved to obtain the desired level of recall. In a perfect system, only relevant documents are retrieved. This means at any level of recall, precision would be 1.0.

2.2.1 Retrieval strategies

Retrieval strategies assign a measure of similarity between a query and a document. These strategies based on the common notion that the more of the modern terms are found in both the document and the query, the more relevant the document is deemed to be the query.

A retrieval strategy is an algorithm that takes a query Q and a set of documents D_1,D_2…..,D_n and identifies the similarity coefficient SC(Q, Di) for each of the documents 1<=I<=n.

The Retrieval strategies identified are

- **Vector Space Model** – Both the query and each document are represented as vector in term space. A measure of similarity between two vectors is computed.
- **Probabilistic retrieval** – A probability retrieval model based on the likelihood that a term will appear in a relevant document is computed for each term in the collection. For terms that match between a query and a document, the similarity measure is computed as the combination of the probabilistic of each matching terms.
- **Language models**– A language Model is built for each document and the likelihood that the document will generate query is computed.
- **Latent Semantic Indexing**– It is an algebraic model of document retrieval based on a singular value decomposition of the vector space of index terms. The occurrence of terms in documents is represented with a term-document matrix. The matrix is reduced via singular Value decomposition (SVD) to filter out the noise found in document so that two documents which have the same semantics are located close to one another in a multi-dimension space.
- **Inference Networks** –A Bayesian network is used to infer the relevance of a document to a query. This is based on the evidence in a document that allows an inference to be made about the relevance of the document. The strength of this inference is used as similarity coefficient.
- **Neural Networks** – It is an algebraic model of document retrieval based on representing query, index terms, and documents as a neural network. A sequence of "neurons" or nodes in a network, that fire when activated by a query triggering links to documents. The strength of each link in the network is transmitted to the document and collected to form a similarity coefficient between the query and the document. Networks are trained by adjusting the weights on links in response to predetermined relevant and irrelevant documents.

- **Boolean Indexing** – A score is assigned such that an initial Boolean query results in a ranking. This is done by associating a weight with each query term so that this weight is used to compute the similarity's coefficient.
- **Genetic Algorithms** –An optimal query to find relevant documents can be generated by evolution. An initial query is used with either random or estimated term weights. New queries are generated by modifying these weights. A new query survives by being close to known relevant documents and queries with less fitness are removed from subsequent generations.
- **Fuzzy Set Retrieval** – A document is mapped to a fuzzy set. Fuzzy set is a set that contains not only elements but a number associated with each element that indicates the strength of membership). Boolean queries are mapped into fuzzy set intersection, union and complement operations that result in a strength of membership associated with each document that is relevant to the query. This strength is used as a similarity coefficient.

2.2.2 Vector Space Model

The vector space model computes a measure of similarity by defining a vector that represents each document and a vector that represents the query [55]

In this model every document in the collection is represented by a multidimensional vector. Each component of such a vector reflects a particular key word or term connected with the given document. The value of each component (Term Weighting) depends on the degree of relationship between its associated term and the respective document. Many schemes for measuring this relationship, very often referred to as **term weighting**, have been proposed.

So **Term weighting** (Wij) basically can be described as a process of calculating the degree of relationship (or association) between a term (Ti) and a document (Dj).

Various techniques for term weighting are -:

1) Binary weighting

In the simplest case the association is binary: W*ij*=1 when key word *i* occurs in document *j*, W*ij*=0 otherwise. The binary weighting informs about the *fact* that a term is somehow related to a document but carries no information on the *strength* of the relationship.

2) Term Frequency Weighting

In this scheme W*ij*=*tfij* where *tfij* denotes how many times term *i* occurs in document *j*. Clearly, the term frequency is more informative than the simple binary weighting. But its drawback is that it focuses on local word occurrences only, not considering the global distribution of terms between documents.

3) *tf-idf* weighting

The ***tf-idf*** (term frequency inverse document frequency) scheme aims at balancing the local and the global term occurrences in the documents. In this scheme W*ij*=*tfij* * log (*N*/*dfi*) where *tfij* is the term frequency, *dfi* denotes the number of documents in which term *i* appears, and *N* represents the total number of documents in the collection. The log (*N*/*dfi*), which is very often referred to as the *idf* (inverse document frequency) factor, accounts for the global weighting of term *i*. Indeed, when a term appears in all documents in the collection, *dfi*=*N* and thus the balanced term weight is 0, indicating that the term is useless as a document discriminator.

2.2.2.1 Similarity Measures

Several different means of comparing a query vector with a document vector have been implemented. The most common of these is the cosine measure where the cosine of the angle between the query and the document vector is given.

$$SC(Q, D_i) = \frac{\sum_{j=1}^{t} w_{qj} x \ d_{ij}}{\sqrt{\sum_{j=1}^{t} (d_{ij})^2 \sum_{j=1}^{t} (w_{qj})^2}}$$

The cosine measure "normalizes" the result by considering the length of the document.

The Dice Coefficient is defined as:

$$SC(Q, D_i) = \frac{2 \sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2}$$

The Jaccard Coefficient is defined as:

$$SC(Q, D_i) = \frac{\sum_{i=1}^{t} x_i y_i}{\sum_{i=1}^{t} x_i^2 + \sum_{i=1}^{t} y_i^2 - \sum_{i=1}^{t} x_i y_i}$$

Unfortunately, choice of a particular similarity measure for certain application lacks theoretical justification and is rather arbitrary [54]. The most commonly used measure is the cosine coefficient [30], which is equal to the cosine of *t*-dimensional angle between two compared vectors.

2.2.3 Stop-words

Whenever text is automatically indexed, some words occur very frequently. Common examples include "and", "me", "she", "that", "the", etc. These words are called Stop-Words. These words shouldn't be considered during processing of the text, as they are not only meaningless, but moreover occur very frequent in normal sentences.

These words are not good at discerning the meaning of the document, and they are commonly removed in IR systems.

Stop-word removal is usually performed before stemming, if that is used, as undesirable removal of words would occur otherwise. However, the inclusion of stop-words in the set of terms can prove beneficial in some cases as it gives the system a bit more data to work on especially with very short descriptions.

2.2.4 Stemming

Stemming is a method used to remove word inflections, in order to reduce the number of different terms for the same base word or word stem. The basic example is the removal of the plural "s" from the end of all words, which of course is too simple an algorithm to give any benefits in practice.

The Porter Stemmer is an example of a more advanced and widely used algorithm [46]. It performs a number of passes over a word and uses tables of common inflections. The best matching inflection is then removed using a carefully constructed set of rules. Stemming is, of course, completely language dependent, so adding support for an additional language in an IR system employing stemming would also mean adding another stemmer.

2.2.5 Thesauri

One of the most intuitive ideas for enhancing effectiveness of an information retrieval system is to include the use of thesaurus. Almost from the dawn of the first information

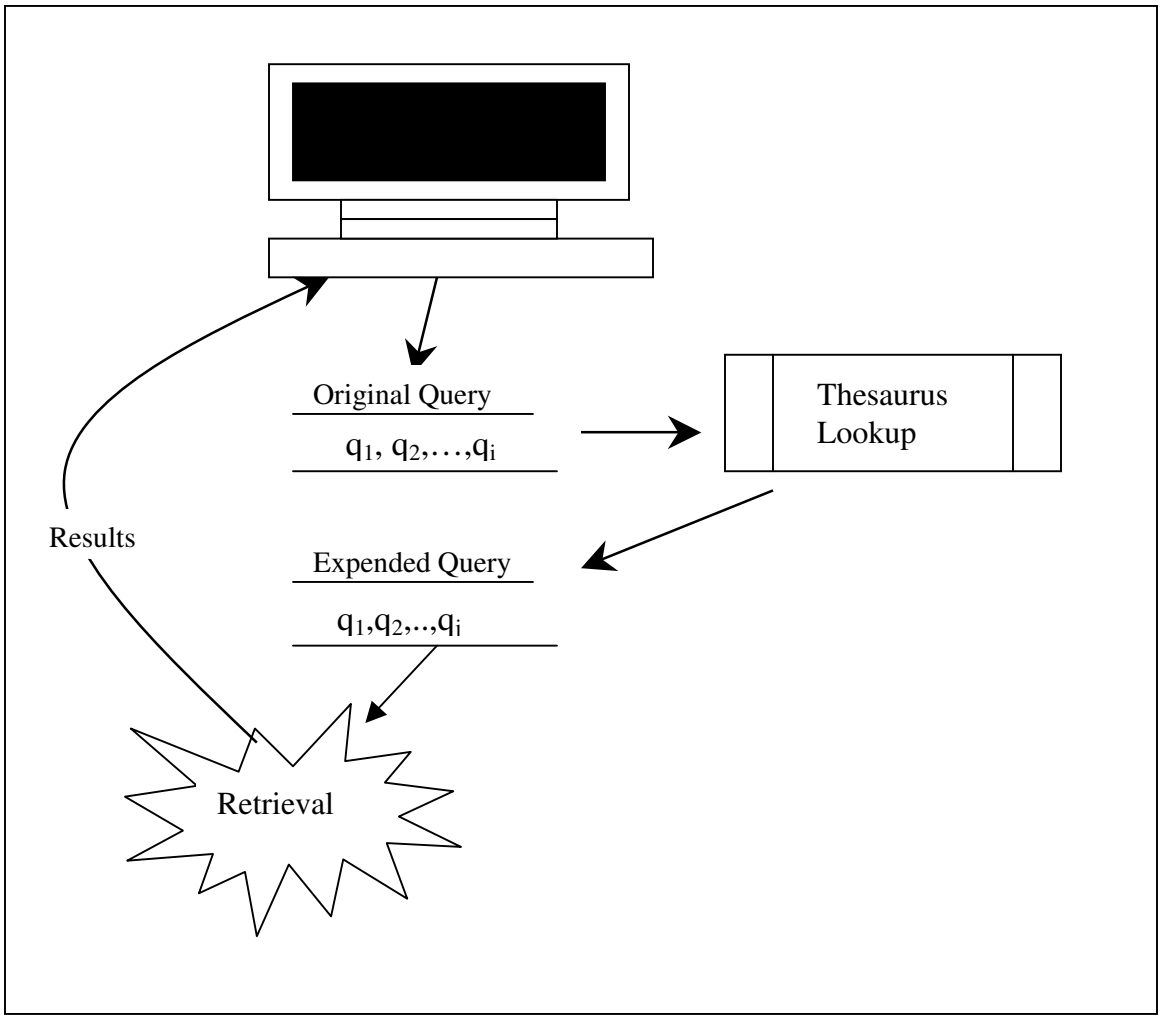

Figure 2.4: Using a Thesaurus to Expand a Query

retrieval system in early 1960's, researchers focused on incorporating a thesaurus to improve precision and recall. The process of using a thesaurus to expand the query is illustrated in Figure 2.4.

A document relevant to query might not match any of the terms in the query. A thesaurus can be used to assign a common term for all synonym of a term, or to expand a query to include all synonymous terms. Detail description of automatic thesaurus generation is found in [56].

Chapter 3

Review of State of Art

This research has tried to find a suitable way to Construct a Reuse Repository System that allow effective access of reusable software components based on the query specified by the user. This work has been greatly influenced by various research efforts on Software Reuse, Component repository systems, Search and retrieval methods. The main objective of this chapter is to summarize software reuse research.

Section 3.1 explores all existing reuse repository systems i.e. concept based, constrained based and code based repository systems. After that Web based and Artificial intelligence based repository system will be discussed. At last semantic based repository system are discussed followed by structured based repository system. Then in section 3.2, Representational methods used in various existing repository systems are discussed. Finally Information Retrieval Methods used in various existing systems will be discussed.

3.1 Component Repository System

A component repository system consists of a repository for storing reusable assets, a search interface that allows users to search for assets in the repository, a representation method for the asset's representation, and facilities for change management. Much research on reuse libraries has been done. Various reuse repository systems are -:

3.1.1 Concept-Based Component Repository Systems

A lot of research on software reuse mainly focus on the concept based reuse repository. Various reuse repositories i.e. GURU, Code-Broker, Code-Finder, LaSSIE are concept based component repositories.

Retieval mechnism used in Code Broker [61] *is* similar to free-text indexing. GURU [30] indexes components based on their textual documentation. Moreover Etzkorn and Davis also tried to use header comments to index the object-oriented programs [13]. Moreover

Girardi [18] and Difelice [12] also used comment and identifiers name to index the components. Michail and Notkin have demonstrated the possibility of using identifier names only to find similar reusable components for comparison [33].

Free-text indexing is much convient and easy for the purpose of developing the component repository and also for the developers of the system to formulate the reuse queries. Researchers also have found that free text indexing performs no worse in terms of retrieval effectiveness than other effort consuming and complex repository systems [15,37].

But the main shortcoming of free text indexing based systems is that these systems do not support the conceptual gap in query formulation. One attempt to bridge the conceptual gap is to use structured representations and knowledge bases. Both Code Finder [22] and LaSSIE [11] use frames to represent reusable components.

Frames in CodeFinder are connected by an associative network whose links have weights to reflect the semantic relationships among components. Further Searching of relevant components is supported by spreading activation. Frames in LaSSIE are structured into hierarchical, taxonomic categories by human experts.

Main problem with both of these systems is that its very difficult to create the frame representations of components. To ease this difficulty ROSA ([18] applies natural language processing techniques to automate the reuse process. But still understanding the semantic meaning of each natural language sentence is very difficult.

The multiple faceted classification schema [49] is another way to structured represent the reusable components.

In this scheme reusable components are represented with multiple facets, each of which is described with a term. A conceptual distance graph has to be constructed to reflect the semantic relationships among terms.

AIRS is a system that combines multiple facets and the frame-based approach [39]. Structured representation-based systems requires a lot of effort in creating representations of components and knowledge bases.

3.1.2 Constraint-Based Component Repository Systems

Constraints of programs can also be used to index and retrieve reusable components. Rittri first proposed to use signatures in reusable component retrieval [52]. His work is further extended by Zaremski and Wing who give a general framework for signature matching in functional programming languages [63]. They present *signature matching* as a method for achieving this goal by using signature information easily derived from the component. They consider two kinds of software components, functions and modules, and hence two kinds of matching, function matching and module matching. The signature of a function is simply its type; the signature of a module is a multiset of user-defined types and a multiset of function signatures. For both functions and modules, they consider not just *exact* match, but also various flavors of *relaxed* match. They also describe an experimental facility written in Standard ML for performing signature matching over a library of ML functions.

Most of the research on signature matching has largely focused on functional programming language but signature matching is also applicable to other strong-typed programming languages. An Ada version of signature matching has been implemented in [58].

Code Broker [61] also applies this technique to the strong-typed object-oriented programming language. Signature matching in Code Broker is not used as the only main method of retrieving components rather it is used as a filter to exclude those components that are different from the current requirements in terms of constraint compatibility.

The formal specification-based approach is another form of using constraints to index and retrieval components. Zaremski and Wing have adopted pre- and post-predicates to find components that exactly match or approximately match a reuse query [62]. A.Mili et al. have tried to classify reusable components based on their refinement order that exists among their formal specifications [36].

3.1.3 Code-Based Reuse Repository Systems

Behavior sampling exploits the code aspect of programs to retrieve reusable components [19,44]. In behavior sampling-based systems, a programmer randomly chooses a small set of sample inputs and computes the corresponding outputs after having specified the signature of the module. Reusable components whose signature is compatible are found and executed on the sample inputs. Components whose outputs match the outputs

computed by the programmer are returned. Behavior sampling is difficult to apply to components with complicated data structures. Moreover, it is unable to find close but not identical components.

3.1.4 Web Based Reuse Repository System

WebBase is a large shared repository of web pages, which is being developed as part of the WebBase project at Stanford University [23] The prototype already has a collection of around 25 million web pages. An earlier version of the prototype was used as the backend storage system of the Google search engine.

The architecture of repository includes five main modules the crawler, the storage manager, the metadata and indexing module, the multicast module, and the query engine.

The crawler module is responsible for retrieving pages from the web and handing them to the storage management module. The crawler searches for the web to retrieve new pages and also fresh copies of pages already existing in the repository.

The storage module performs various critical functions that include assignment of pages to storage devices, handling updates after every fresh crawl and scheduling and servicing various types of requests for pages.

The metadata and indexing module is responsible for extracting metadata from the collected pages and for indexing both the pages and the metadata. The query engine and the multicast module together provide access to the content stored in the repository.

The repository supports three type of access modes i.e. Random based, Query Based and Streaming access for retrieving pages.

In Random access mode a specific page is retrieved from the repository by specifying the URL associated with that page. In Query Based Access mode requests for a set of pages are specified by queries that characterize the pages to be retrieved. In the Streaming access mode the pages in the repository are retrieved and delivered in the form of a data stream directed to the requesting client application. This access mode is important for applications that need to deal with a large set of pages.

3.1.5 Artificial Intelligence Based Reuse Repository

AIRS is an AI-based library system for software reuse, which was developed by E.J. Ostertag, J.A. Hendler, R. Prieto-Diaz, C. Braun [12]. AIRS allows a developer to browse a software library in search of components that best meet some stated requirement. A component is described by a set of (feature, term) pairs. A feature represents a classification criterion, and is defined by a set of related terms. AIRS also allow representation of packages, that is, logical units that group a set of related components. As with components, packages are described in terms of features. Unlike components, a package description includes a set of member components.

Candidate reuse components and packages are selected from the library based on the degree of similarity between their descriptions and a given target description. Similarity is quantified by a non-negative magnitude (called distance) that represents the expected effort required to obtain the target given a candidate. Distances are computed by functions called comparators. Three such functions are presented: subsumption, closeness, and package comparators. The AIRS classification approach is based on a formalization of the concepts and is similar to faceted classification. The functionality of a prototype implementation of the AIRS system is illustrated by application to two different software libraries: a set of Ada packages for data structure manipulation, and a set of C components for use in Command, Control, and Information systems existing CASE and development tools, such as structure design tools, versioning systems and configuration management systems.

3.1.6 Denotational Semantics Based Repository

This section describes the current research on software libraries whose operation depends on the denotational semantic definition of software assets.

In [42,43], Perry and Popovich introduce a prototype of a software library, where software assets are represented by predicates that define their main functional features and interface characteristics. The software library, named *Inquire*, is based on a specification-based software development environment, named *Inscape* [42]. In [52], Rittri proposes a method for software component storage and retrieval that applies to modules written in a functional language and is based on signature matching. The matching criterion is defined using polymorphic type systems and provides independence of the order of components in a type.

In [51], Rittri improves the recall of his method by weakening the matching condition using type isomorphism. Runciman and Toyn [53] propose a similar solution, which uses polymorphic type systems to define criteria of signature matching in the context of functional programming, and provides independence of the number of arguments; this latter feature is intended to preserve recall in cases when the signatures differ by syntactic details.

In [8], Cheng and Jeng discuss an organization of a software library that is based on formal specifications of components and queries. Cheng and Jeng extend their work in [9] by investigating matching criteria that attempt to minimize measures of distance between the query at hand and candidate library components, and they extend it in [26] by defining matching criteria between components, and between methods; these matching criteria make provisions for sub-typing, variable renaming, and parameter permutation. Chen and Cheng extend this work further to deal with software development at the architectural level.

In [5], Boudriga *et al.* discuss the design of a software library based on relational specifications of components and queries, and on an ordering of library components by a refinement ordering relation. The matching condition is defined as the correctness of the component with respect to the query, and the ordering between library components is used to guide the search process.

In [63,65], Zaremski and Wing discuss signature matching as a mechanism for retrieving software components from a software library. Queries and library components are represented by their signatures, and a hierarchy of matching criteria is defined and discussed. The basic matching condition provides for equality between the two signatures, modulo variable renaming and parameter ordering. Zaremski and Wing obtain a *generalized match* (if the type of the component is more general than the type of the query) or a *specialized match* (if the type of the query is more general than the type of the component); also, by applying uncurrying and currying transformations to the query and the component, they can match functions that do not have the same number of parameters. Zaremski and Wing extend their work on signature matching by investigating specification matching in [64]. They represent queries and components by (precondition, post condition) pairs, and define a general matching criterion.

In [41], Penix and Alexander advocate a formal specification based, domain theory oriented, approach to software component retrieval. Like Mili *et al.* [5,27,28] Penix and Alexander make a clear distinction between *exact retrieval* and *approximate retrieval* and make separate provisions for them. Also, like Zaremski and Wing [63,64,65], Penix and Alexander use a hierarchy of matching criteria and specify components in Larch. Penix *et al.* further extend this work in [40] by considering software assets that are represented at the architectural level.

Fischer *et al.* [14] propose a stepwise filtering procedure that proceeds in three steps:

1. Signature matching.

2. Model checking.

3. Theorem proving.

The idea of this approach is, of course, to reduce the search space as the retrieval algorithm gets more and more complex The semantics of an asset are described in terms of pre and post-conditions expressed in Hoare logic [24].

3.2. Component Representation methods

A large number of solutions for search and retrieval have been proposed. This solution depends on how the components are represented in the repository. Methods to represent the component in repository can be classified into four different types [34]. These are:

- **Simple keyword and string search:** This *keyword-based* approach [36] is indeed the archetype of descriptive method. In this approach, a software asset is represented by a set of keywords.
- **Faceted classification and retrieval:** Faceted classification approaches [50,66] for components retrieval consist of a collection of facets or classifications, which represent the type of information, that is relevant for identifying reusable components. Each facet has a name and an associated term-space called vocabulary, which is a collection of terms used to describe aspects of the facet.
- **Signature matching:** This approach [65] primarily relies upon type matching and type transformation. Classes are represented by a multi-set of feature signature.

Signature matching approach assumes that the set of allowable signatures in the system is known. Feature can be simple types, constructed types, user defined types, type variables and functions.

- **Behavioral matching:** Behavioral-based retrieval approaches [63,64,19] are based on the notion of exploiting the executability of software components to classify them. Testing the component with different arguments calling his functions yields dynamic responses, which are collected. This collection is called the component behavior. An ordering on behaviors is then used to classify components and to search through the library of components. The programs used to produce the components behavior try to call a subset or all the functions of the component and recover the results. If the program calls functions that do not exist in the component, the components will ignore the call.

In the behavioral-based retrieval approach, the engineer query is a program that calls some functions with specific arguments and this program will be plugged to all components of the library to test the components behavior. The components that respond to the searched behavior will be selected and presented to the engineer.

3.3 Information Retrieval methods

In [16,17], Frakes and Nejmeh apply an information retrieval method to the storage and retrieval of software assets in a software library. They use a strictly free-text approach to the indexing of software assets, which are C programs. Each C module is characterized by a set of single-term indices, which are extracted from the natural language headers; for the sake of uniformity, these indices are extracted automatically by a natural language parser.

In [30,31], Maarek, Berry and Kaiser discuss an information retrieval system for the storage and retrieval of software components. The method of Maarek *et al.* relies on a natural language description of assets and a natural language representation of queries. Software assets are automatically indexed to ensure uniformity across the library; the indexing process extracts from each asset a set of indices that define its *profile*, and that play the role of *surrogate* in subsequent retrieval operations.

Indexed documents are organized in the library into a browsing hierarchy, using clustering techniques. Retrieval from the library takes place in three steps, namely: query specification, linear retrieval and browsing. Maarek *et al.* have developed a tool to support their method, called GURU, and have experimented with it on sample queries.

In [21], Helm and Maarek apply information retrieval considerations for the retrieval of classes from class libraries; here, the additional information provided by inheritance structures is used.

In [11], Devanbu *et al.* develop a specialized system called *Large Software System Information Environment* (LaSSIE) that incorporates a large knowledge base, a semantic retrieval algorithm based on formal inference, and a powerful user interface; LaSSIE is intended to help programmers find useful information about a large software system - specifically, an AT&T scalable PBX product. The knowledge base of LaSSIE is built using a classification based knowledge representation language; the knowledge base serves as a repository of information about the PBX product, as well as an index for retrieving reusable components. It is structured as taxonomy of nodes that have four possible types: *action*, *object*, *doer*, and *state*.

A LaSSIE query is a description of an action, formulated by filling various frame-like slots; LaSSIE resolves a query by retrieving instances of the knowledge base where the structure of the query occurs with the proper fillers. This method could be viewed as an information retrieval method or a descriptive method.

In [10], Clifton and Li use design information as abstraction and propose neural network technology to accomplish the match. In [38], Mittermeir and W¨urfl suggest to perform automatic code analysis and propose to match the resulting flow-graphs with control and data flow sketches done by the programmer.

Hypertext methods are specialized forms of traditional information retrieval methods but difference is that they have the ability to deal with multimedia documents, and their ability to work across computing sites. The work of Lucarella and Zanzi [29] fits this

characterization. Other hypertext approaches include the works of Isakowski and Kaufmann [25] and Poulin and Werkman [48]. In [48], Poulin and Werkman describe a Reusable Software Library (RSL) interface and a search tool using Mosaic.

Chapter 4

Proposed System and its Implementation

Before programmer can take advantage of reuse they have to locate components. It takes a lot of time and effort to locate components that are required to solve the current problem. Whether a component is going to be used to solve a given problem depends on whether such component actually exists and if it is possible to locate it with in reasonable time [60]. If developer is not aware of the existence of a required and usable component he or she is unfortunately disinclined to make effort of searching components [61]. These leads to failure of reuse process and as a result developer will have to write everything from scratch.

4.1 Problem Statement

Having a collection of components in repository is not of much help if there is no interactive and efficient search functionality to help the developers in locating the reusable components. Furthermore if such interactive system is unable to locate the required components within first few attempts the user is unlikely to return. This means the system has to handle a diverse range of user queries and also take into account the mental perception that developers mostly use to describe the components.

As such a reuse repository system may cover many different domains, some of the returned components may not be relevant to the current problem. When the results of a query have been properly categorized and also gives the relevancy of components with respect to query, the user is able to quickly select the appropriate set of components very easily. If the system also takes feedback from the user related to result it returned and use this knowledge in processing future queries then the system will become self-learner and based on this automated learning, system can return more relevant components that satisfy the need of the user.

This leads us to the problem statement that we will try to solve in this thesis.

This thesis report will answer the following questions:

- How can a reuse repository system be created, which supports efficient and relevant searching of software components?
- How can basic problem such as searching for any component not matching exactly but performing the similar functionality can be solved?
- How can results of a search be presented to the users so that they can select best components that satisfy their requirements?
- How Graphical User Interface (GUI) of the reuse repository system should be created so that user can easily specify their requirements?
- How user response can be gathered to make the system more intelligent and self-learning so that system can use this knowledge in processing future queries and can effectively deliver the required components to the user.

Objective is to make a Reuse Repository System with effective and optimum search functionality along with easy to use user interface. The resulting Reuse Repository System should be easily extensible and adaptable to specific problem domain.

Why it is worthwhile to answer this question:

Despite its many benefits, component reuse has not yet received wide success in practice due to many difficulties associated with it. Due to the large volume and constantly evolving nature of component repositories, programmers often fail to anticipate the existence of reusable components as a result they did not make any effort to search or locate it in repository. Even if programmers are aware of the existence of reusable components, many times they are not able to start the locating process because mostly they do not know how to locate the reusable components from repository. Moreover Component can be located or retrieved effectively if they are stored in the repository in a well- defined uniform manner.

So in order to take complete advantage of reuse, there is a need for Component Reuse Repository System. There exist a lot of component reuse repository systems. Most of them are not effective i.e. retrieve irrelevant components or take too much processing time or resources and those who are effective are very complex and difficult to use by

user. i.e. provide very complex user interface, or user cannot specify the query in natural language .So this lead to motivation for us and as a result a Reuse Repository System has been proposed and developed.

4.2 Proposed System Overview

Proposed system will first perform indexing of reusable software components. For this purpose Indexer module is used which create the indexes and component repository from existing collection of components. It will create index and make it available for search.

In order to ensure better search results than simple exact text matching, a method from Information Retrieval called Vector Space Model will be used. Stemming algorithm is also applied to reduce the stemming effect of words. Thesauruses are also applied to make the search more efficient. This Reuse Repository System will use both concept matching and signature matching to retrieve the components from the repository.

After retrieving the components from repository, system presents the ranked results to the user according to their relevancy. To effectively process the future queries, system also permits the user to assign the weight to any particular component.

4.3 Architecture

Our proposed Reuse Repository System consists of three main modules: Indexer Module, Fetcher module and Display module. The Architecture of system is shown in Figure 4.1. The Indexer module creates the index of components so that they can be easily retrieved as required. Indexer module will be described in detail in section 4.5. User uses the web browser to specify the requirements to retrieve the needed components.

User can specify the requirements by writing query in natural language and to increase the accuracy of retrieved components user can also specify the signature of the components

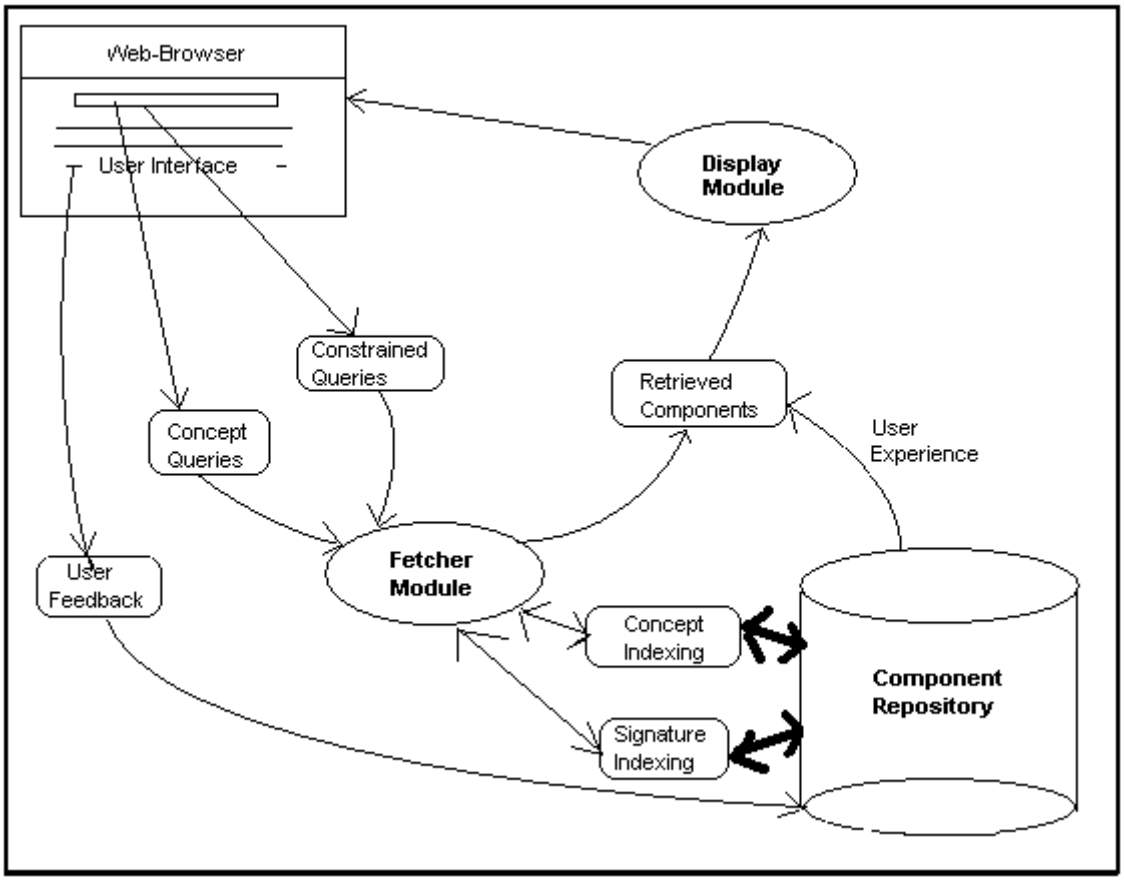

Figure 4.1: Architecture of Proposed Reuse Repository System

Once user completely specifies the query, system starts query processing. In proposed Reuse Repository System, query processing consists of mainly four steps. First stop words are removed from the query. Second thesaurus is applied for query expansion so that user query includes all synonymous terms. Then whole expanded query is passed to the Porter stemming algorithm [46] so that word inflections can be removed. Finally resultant query is passed to the Fetcher Module which retrieves the matching components from repository. Fetcher Module uses both concept matching and signature matching to retrieve the components from repository.

Reusable components retrieved by fetcher module are passed to the Display module which displays the description of matching components on the user interface (web-browser).

4.4 Creating Component Repository

Proposed Reusable Component Repository is created by using the indexer module and parser that we developed in java. Component repository is created in three phases as shown in Figure 4.2

First relevant information is extracted from components for indexing purpose. Concepts and signatures are extracted from documentation using javadoc and docflex.

In second step parser will parse them and store all the component information in relational database and in third and last step indexer module will create the indexes based on concepts, signatures and other attributes like author name of class etc. These indexes are also stored in relational database for their persistent storage purposes.

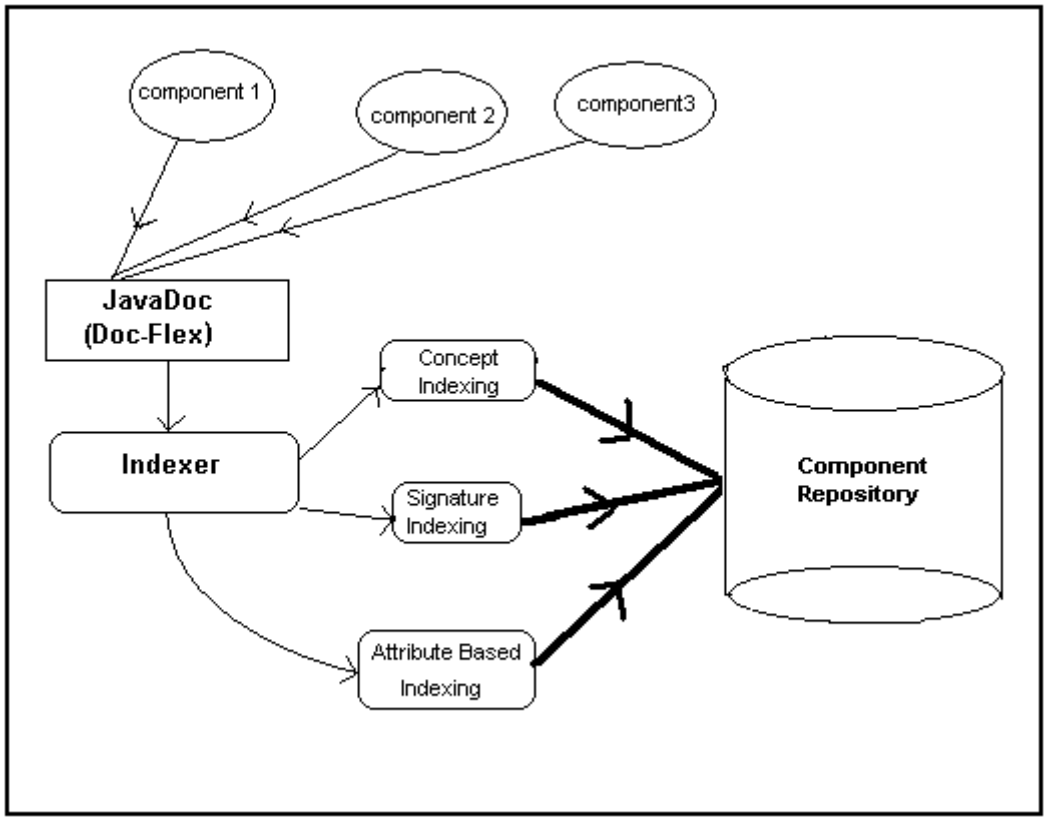

Figure 4.2: Creating Component Repository

4.5 Indexing Phase

An effective retrieval mechanism is essential for retrieving the relevant reusable components. Component retrieval effectiveness mostly depends on how components are stored in repository and indexed. Moreover representation of query also depends on how indexing is performed.

This reuse repository supports two basic processes: the location and indexing of components and the search and retrieval of a component. The location and indexing of components is primarily an automated background task, while a user of system typically performs search and retrieval.

The indexing phase of components is subdivided into four steps as shown in Figure 4.3. The first step is the initial indexing where a collection of components are parsed using Javadoc tool with a custom Doclet ”Docflex” which extract their concepts (which mostly revealed in javadoc comments) and then give the output in .doc files.

After that Parser (created in java language) parse the whole set of files and create the relational database, which consist of components description. Then indexer module created in java is used to normalize the Database. After normalization indexer create the inverted index, which can be used later for searching and retrieval of components.

Six type of information is extracted for the purpose of indexing a class i.e. class name, its parent classes name, name of interfaces it implements, description of the class, author name and version of the class. For each class this complete description is stored as attributes.

For indexing methods, five type of information is extracted i.e. method name, its class name, its return type, arguments and method description .For each method this complete description is stored as attributes in relational database.

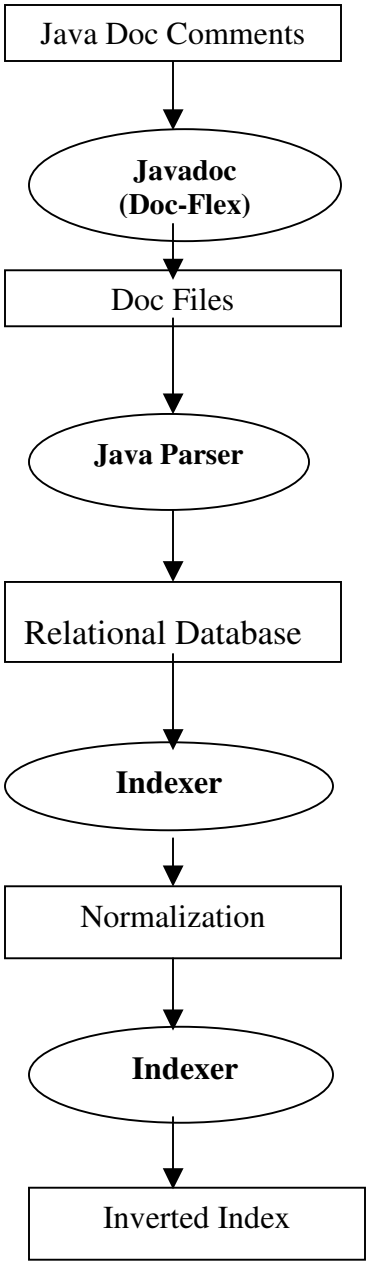

Figure 4.3: Indexing Phase

Use of attributes make it easy to provide user extra facilities for retrieval of components i.e. retrieval by author name or by parent class etc. Parser also assigns unique identifier to each class and method so that they can be referenced during searching and retrieval of components.

In brief we can say that our implementation of Reuse Repository System decodes components based on both concepts and constraints. System extracts the concept of a

component from its associated documentation embedded in source programs and the constraint from the signature of a component. Reuse queries are represented in the same format i.e. conceptual and constraint form. User will write the query in natural language i.e. English and he can also specify the signatures of components to retrieve more relevant components. Moreover he can also specify various other important attributes i.e. author name etc to narrow the search.

4.6 Searching and Delivering Components

Once Indexing phase is complete, user can use this reuse repository system for searching the relevant components. User can specify the query in the interface provided by the system. System will process this query and as a result retrieve the relevant components from repository and display them to the user. User can select or choose needed component and use them to solve the problem in hand.

4.6.1 User Interface

The interface of the system is similar to ordinary search engine which allows the user to specify the query in Natural language i.e. English. After writing the query user can specify the signature of components. For example user needs to specify return type of method, the number of parameters, type of each parameter etc. More over user can also use various advance options to restrict the search in particular direction i.e. search by author. Figure 4.4 shows the interface of the system.

This interface is easy to use and doesn't put so much effort on the user to specify their needs.

Once user submits their requirements (i.e. query, signatures etc) system will process the query, interpret it, find the relevant components and display the result to the user. It will show all relevant components description along with their relevancy to the query specified. User can easily access this interface using the web-browser.

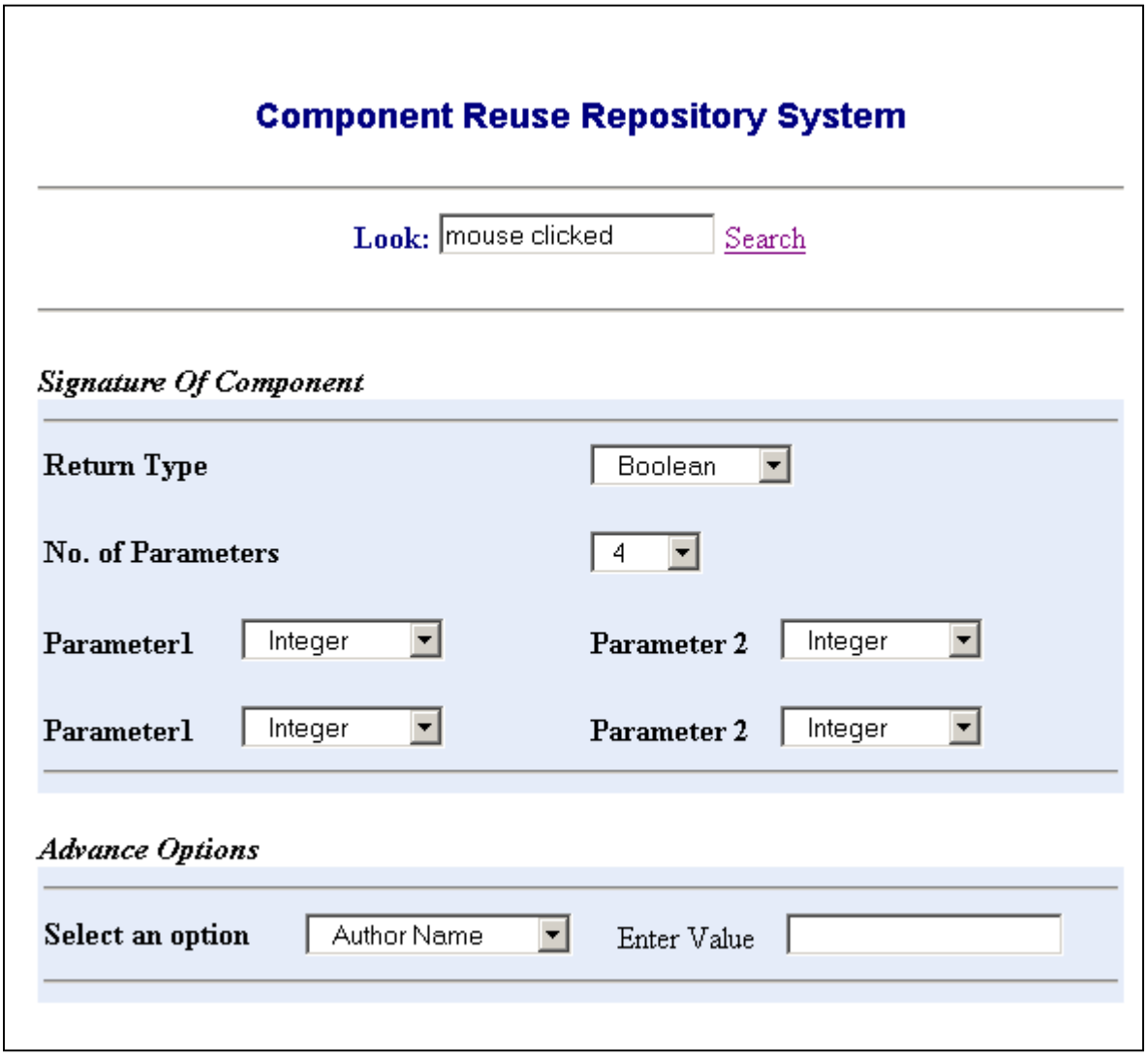

Figure 4.4: Interface of Reuse Repository System

Figure 4.5 shows the result when we enter the query " mouse clicked" and no advance options are specified. Whenever user click on the name of any particular component system will display all the necessary detail of component in separate window.

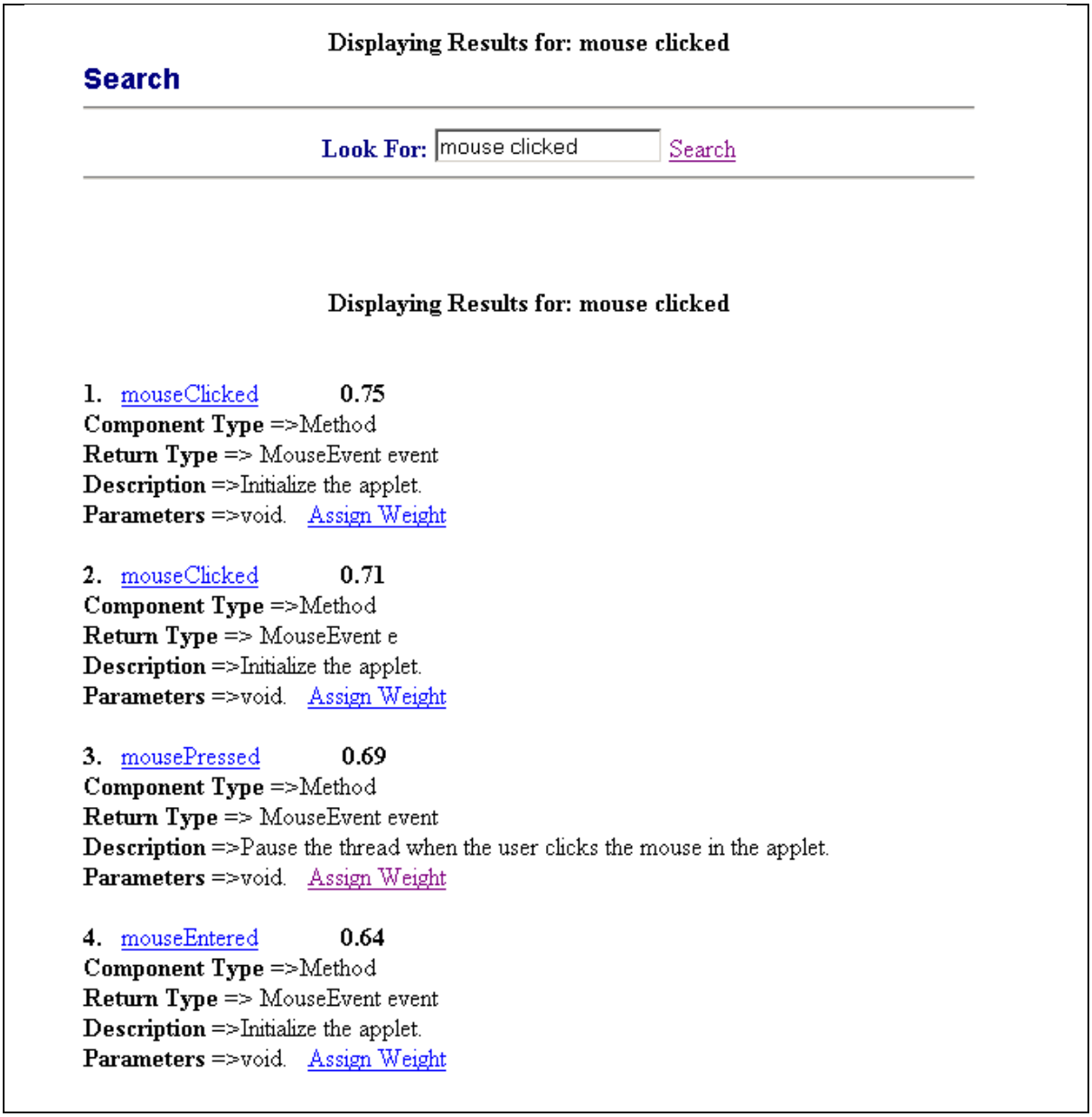

Figure 4.5: Result For Query "Mouse Clicked"

Once the result is displayed to the user, he or she can also select option "assign weight" to give weight to the components according to their need. Whenever user clicks this option, system opens a dialog box and asks the user to enter the weight of component between 0 to 10 as shown in Figure 4.6.

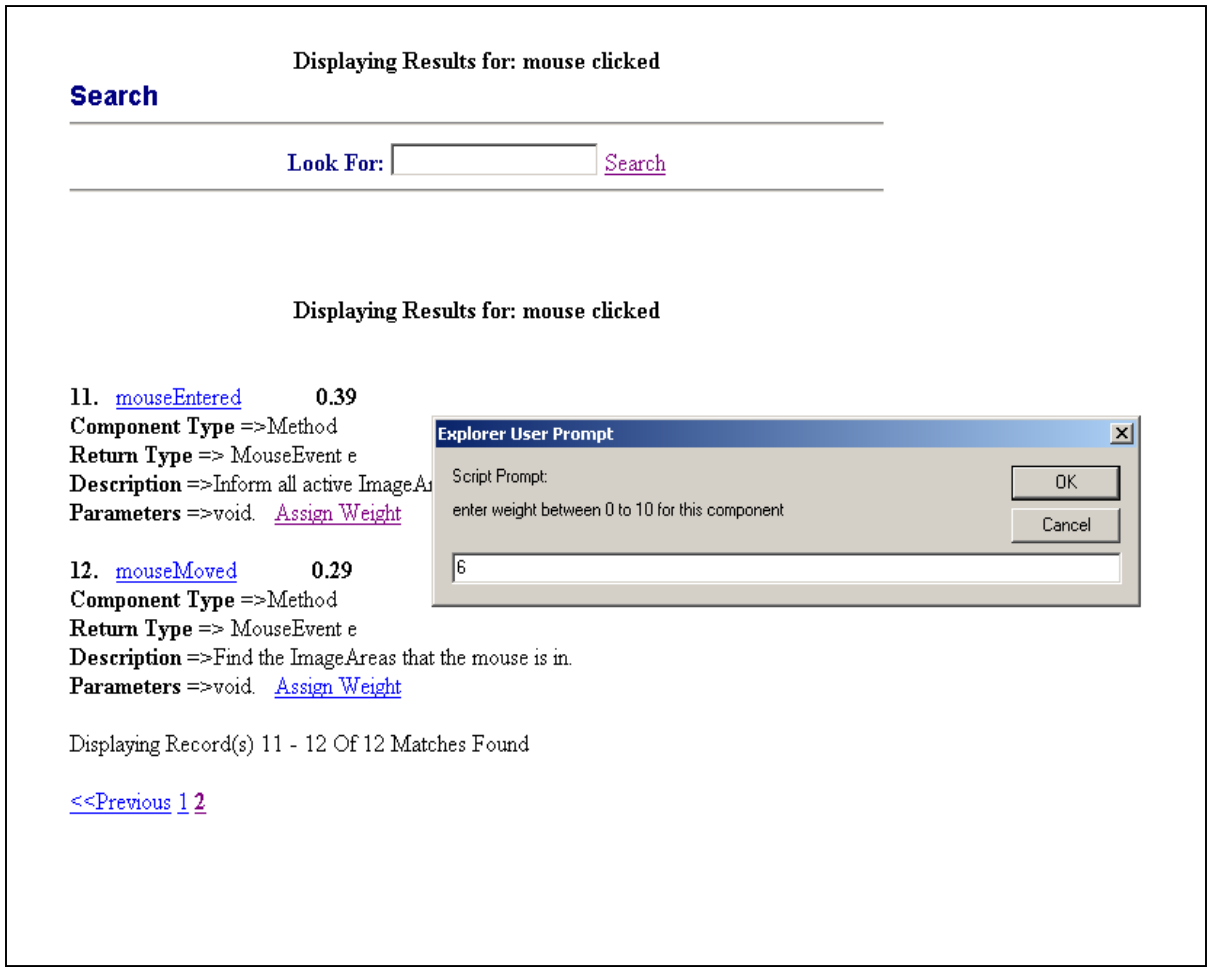

Figure 4.6: Weight Assignment of component.

Whenever user submits this weight, system will store that value in the repository itself. This information will make system more intelligent and more capable of returning the relevant component in processing future queries. So more the system interact with user, more experience it will gain and as a result it make system more intelligent and increase its capability to return more useful and relevant reusable components.

4.6.2 Query Interpretation

Once the user submits the query, first system will take the query and split the whole query into set of tokens. After that system first remove the stop words from these tokens because they are the meaningless words and are not good at discerning the meaning of the component. The process of query interpretation is shown in following Figure 4.7.

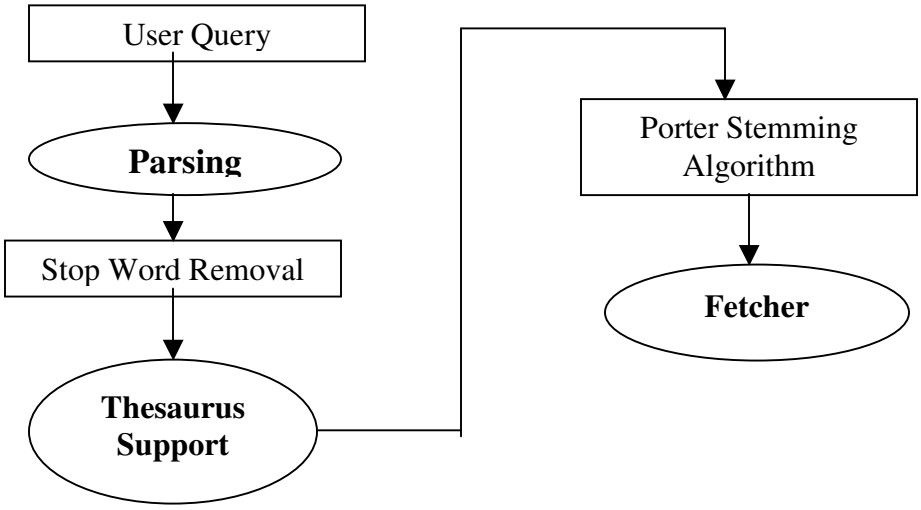

Figure 4.7: Query Interpretation

Once stop words are removed thesaurus is applied to these tokens for the purpose of query expansion so that user query include all synonymous terms and can retrieve more relevant components. Then the expanded collection of tokens is passed to the Porter stemming algorithm [Porter, 1980] so that word inflections can be removed. Finally resultant collection of tokens is passed to the Fetcher Module, which retrieve the matching components from repository.

4.6.3 Retrieval of Components

This system use both Concept similarity and Constraint similarity for retrieving the components from the system. First it performs the concept similarity by comparing the query with the components stored in repository. System uses the Vector space model to perform the concept similarity. After this it perform signature matching to filter out and retrieve more relevant components. Finally if user has specified any constraints then system will search the components using those constraints only.

At last system sorts the retrieved components according to relevancy value and display them to user in descending order so that most relevant component are seen by user first.

4.7 Relevancy of Components

Relevance of component is determined by the combination of concept similarity, constraint compatibility and weight given by user to the components during the previous searches. Vector space model based indexing and retrieval technique is used to compute *concept similarity*. Constraint compatibility is computed by *signature matching*. So relevance value is calculated as:

Relevance value = ((concept similarity * wg1 + constraint similarity * wg2) / 2+ user experience * wg3)/2

Here wg1, wg2 and wg3 are the importance given to concept similarity, constraint similarity and user experience respectively.

Developers can assign different weight to these values according to the importance they want to give to the concept similarity, constraint compatibility and user experience value.

4.7.1 Concept Similarity

In the vector space model, components and queries are represented as vectors of terms

contained in the whole collection of components. The value of each element in the vector reflects the importance of a particular term in representing the concept of that component.

Under this scheme both components and queries are represented as vectors of terms and similarity between those vectors is determined by calculating cosine angle between them.

When the component and the query are identical or related, their vectors should be identical in the vector space and the cosine is one; and when they share no common terms so they are orthogonal to each other, the cosine is zero. Upon receiving a query from a user the system will thus compute the cosine for each component against the query vector and return their concept similarity value.

4.7.2 Signature Matching

Once components are retrieved based on the concept similarity then signature matching is used to filter out the unwanted components and to increase the precision of more relevant components.

Signature matching is the process of determining the compatibility of two components in terms of their signatures [62,63,64]. It is a technique that associates signatures with components. These signatures can then be used as keys to retrieve relevant components from an existing library of components.

Two signatures

Sig1: Parameter 1 → Return type1

Sig2: Parameter 2→ Return type 2

will return exact match only if parameter1 is in structural conformance with parameter2 and return type1 is in structured conformance with return type2. Two types are in structural conformance if they are formed by applying same type constructer.

In our implementation Signature matching value will be determined by considering three factors: Return Type of method, Number of Parameters and Type of Parameters. In our implementation we give equal priority to all these three factors.

So signature-matching value can be determined as:

Signature matching value = (w1 +w2 +w3)/3

w1=Return type matching value
w2= Parameters type matching value
w3= Value obtained by matching number (count) of Parameters

These weight values are computed using following Rules:

- If number of parameters of a component is equal to number of parameter specified by user in query then this matching value will be 1.0 otherwise it will be 0.
- If there is exact matching between the return type of component and return type specified by user in query then this value will be 1.0. If exact match was not found then system will try to perform transformation relaxation matching [63]. Under this matching valid type conversions like (converting Integer to Byte etc) are performed to find out any relaxation matching. If relaxation matching is found then system will return 0.5 otherwise it return 0.

Matching between parameters type will be considered into 2 cases.

a) If number of parameters of component in repository is equal to number of parameters specified by user then system will try to find matching between parameters in specified order.

   Matching value will depend on number of parameters i.e. if there are 3 parameters then total Parameter matching value will be average of matching value of these three parameters corresponding to parameters specified by user.

   First system will try to find out the exact match between each parameter of component and their corresponding parameter specified by user. If exact match found matching value will be 1.0 for this parameter.

   If exact match is not found system will try to perform transformation relaxation matching. If it is found matching value will be 0.5 for this parameter else it will be considered as 0.

   After performing matching of all parameters, their average is calculated and returned as the final matching value for this factor.

b) If number of parameters of component in repository is not equal to number of parameters specified by user then system will try to find matching between parameters in their corresponding ordering i.e. if component have 3 parameters and user specify only 1 parameter then this parameter will be matched against the first parameter of component.

   If exact match is found, 1.0 as matching value is returned otherwise transformation relaxation matching is searched. If it is found, 0.5 as matching value is returned otherwise 0 is returned as matching value.

   Finally average matching value of all parameters is calculated and returned as final matching value.

4.7.3 User Feedback

Our implementation of reuse repository system not only performs concept similarity and constraint similarity but also consider the user experiences while computing relevancy of components. This is implemented by giving the user a choice of assigning weight to a each returned result as relevant or irrelevant – and then submitting this information to the system.

Whenever the system displays the result of a query to the user, it also allows the user to give the weight to the returned relevant components according to their requirements. This experience is stored in the repository and is used next time whenever user enters the same query.

First time the user experience value is considered as 0.5 for each component that are returned as result because we think that after performing concept similarity and constraint similarity, system will return only relevant components. User can change this value for each component when reuse repository system display results to the user. User can use change weight option to perform this.

4.8 Experimental Results

Few case queries are conducted to demonstrate the query matching used in proposed system. 27609 components from JDK1.4.1 have been considered as sample repository. In sample case queries 68 components of repository are considered for simplicity. This chosen set is a mix of most relevant as well as out of question components compared to user query. Appendix B shows the specification of these components in repository.

Various sample queries, System response and relevancy calculation are given below:

Case 1:

1. User gives only Conceptual Query and doesn't specify any signature.

(A) Query: Mouse Clicked

Results: Based on this query Fetcher extracted following components from repository.

S.No.	Method Name	Class Name	Function Parameters	Method Description	Return Type
1	mouseEntered	Animator	void	Initialize the applet.	MouseEvent event
2	mouseExited	Animator	synchronized void	Initialize the applet.	MouseEvent event
3	mousePressed	Animator	void	Pause the thread when the user clicks the mouse in the applet.	MouseEvent event
4	mouseClicked	ImageMap	void	Initialize the applet.	MouseEvent e
5	mouseDragged	ImageMap	void	Inform all active ImageAreas of a mouse drag.	MouseEvent e
6	mouseEntered	ImageMap	void	Inform all active ImageAreas of a mouse drag.	MouseEvent e
7	mouseExited	ImageMap	void	Make sure that no ImageAreas are highlighted.	MouseEvent e
8	mouseMoved	ImageMap	void	Find the ImageAreas that the mouse is in.	MouseEvent e
9	mousePressed	ImageMap	void	Inform all active ImageAreas of a mouse press.	MouseEvent e
10	mouseReleased	ImageMap	void	Inform all active ImageAreas of a mouse release.	MouseEvent e

Table 4.1 Result of Query Mouse Clicked

Relevancy of each component is calculated in this case as:

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	mouseClicked	1	0	0.5	0.5
2	mouseEntered	0.46	0	0.5	0.365
3	mouseExited	0.46	0	0.5	0.365
4	mousePressed	0.46	0	0.5	0.365
5	mouseDragged	0.46	0	0.5	0.365
6	mouseEntered	0.46	0	0.5	0.365
7	mouseExited	0.46	0	0.5	0.365
8	mouseMoved	0.46	0	0.5	0.365
9	mousePressed	0.46	0	0.5	0.365
10	mouseReleased	0.46	0	0.5	0.365

Table 4.2 Calculation of relevance of components for query "Mouse Clicked"

w1=w2=w3=1

We observed from Table 4.2 that component "mouseClicked" is more relevant as compare to other components because it has the higher precision (0.5) as compare to other components.

(B) Query: Image ADD

Results: Based on this query Fetcher extracted following components from repository.

S.No.	Method Name	Class Name	Function Parameters	Method Description	Return Type
1	MakeImage	ImageMap	ImageProducer,Image Filter,Integer	Create the Dynamic Image	Image
2	CreateImage	ImageMap	ImageProducer,Image Filter,Integer	Create the Image	FilterImageSource
3	AddImage	ImageMap	Image	Add one image into another	Image
4	GetImage	ImageMap	String, String	Return the Image specifed by document base and Code base	Image
5	imageUpdate	ImageMap	Image, int, int, int, int, int	Handle updates from images being loaded.	Boolean
6	drawImage	ImageMap	Image, int, int, Handle	Draw the image on the Client Area	Boolean

Table 4.3 Result of Query Image Add

Relevancy of each component is calculated in this case as:

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	addImage	1	0	0.5	0.5
2	makeImage	0.64	0	0.5	0.41
3	createImage	0.64	0	0.5	0.41
4	getImage	0.64	0	0.5	0.41
5	imageUpdate	0.64	0	0.5	0.41
6	drawImage	0.64	0	0.5	0.41

Table 4.4 Calculation of relevance of components for query "Image Add"

Case 2:

1. User gives Conceptual Query and also specify exact signature.

(A) Query: Mouse Clicked

Signatures:

Return Type: Mouse event

Number of Parameter: 1

Return Type of Parameter: void

In this case too System returns the same Table 4.1 but component relevancy is improved because of exact signatures.

Relevancy of each component is calculated in this case as:

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	mouseClicked	1	1	0.5	0.75
2	mouseEntered	0.46	1	0.5	0.615
3	mouseExited	0.46	1	0.5	0.615
4	mousePressed	0.46	1	0.5	0.615
5	mouseDragged	0.46	1	0.5	0.615
6	mouseEntered	0.46	1	0.5	0.615
7	mouseExited	0.46	1	0.5	0.615
8	mouseMoved	0.46	1	0.5	0.615
9	mousePressed	0.46	1	0.5	0.615
10	mouseReleased	0.46	1	0.5	0.615

Table 4.5 Calculation of relevance of components for query "Mouse Clicked" with exact signatures

On comparing the results shown in table 4. 5 and table 4.2, we observed that in case of exact signature match (Table 4.5) results are more relevant as compare result shown in Table 4.2 because system used both Signature matching and Conceptual matching to retrieve components.

(B) Query: Image Add

Signature:

Return Type: Image

Number of Parameter: 1

Return Type of Parameter: Image

Results: Based on this query Fetcher extracted following components from repository.

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	addImage	1	1	0.5	0.75
2	makeImage	0.64	(1+0+0)/3=0.33	0.5	0.49
3	createImage	0.64	(0+0+0)/3=0	0.5	0.41
4	getImage	0.64	(1+0+0)/3=0.33	0.5	0.49
5	imageUpdate	0.64	(0+1/5+1)/3=0.46	0.5	0.52
6	drawImage	0.64	(0+1/4+1)/3=0.41	0.5	0.51

Table 4.6 Calculation of relevance of components for query "Image Add" with exact signatures

***Constraint Similarity value = (w1 +w2 +w3)/3**

w1=Return type matching value

w2= Parameters type matching value

w3= Value obtained by matching number of Parameters

Case 3:

User gives Conceptual Query and also specify signature but not exact.

Query: Mouse Clicked

Signature:

Return Type: Void

Number of Parameter: 1

Return Type of Parameter: Integer

In this case too System return same Table 4.1 but relevancy is improved because of signatures specification given by user.

Relevance Calculation is shown in following tables

S.No.	Method Name	Concept Similarity (A)	*Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	mouseClicked	1	(0+1+0)/3=0.33	0.5	0.58
2	mouseEntered	0.46	(0+1+0)/3=0.33	0.5	0.44
3	mouseExited	0.46	(0+1+0)/3=0.33	0.5	0.44
4	mousePressed	0.46	(0+1+0)/3=0.33	0.5	0.44
5	mouseDragged	0.46	(0+1+0)/3=0.33	0.5	0.44
6	mouseEntered	0.46	(0+1+0)/3=0.33	0.5	0.44
7	mouseExited	0.46	(0+1+0)/3=0.33	0.5	0.44
8	mouseMoved	0.46	(0+1+0)/3=0.33	0.5	0.44
9	mousePressed	0.46	(0+1+0)/3=0.33	0.5	0.44
10	mouseReleased	0.46	(0+1+0)/3=0.33	0.5	0.44

Table 4.7 Calculation of relevance of components
for query "Mouse Clicked" with signatures but not exact

***Constraint Similarity value = (w1 +w2 +w3)/3**

w1=Return type matching value

w2= Parameters type matching value

w3= Value obtained by matching number of Parameters

On comparing the above result with result shown in table 4. 2 and table 4.5, we observed that in case of exact signature match (Table 4.5) results are more relevant as compare result shown in Table 4.2 and 4.7. Component relevancy is decreased in Table 4.7 because user doesn't specify the exact signature. But still results in Table 4.7 give more relevant components as compare to Table 4.2 due to signature specification.

Query: Image Add

Signature:

Return Type: Boolean

Number of Parameter: 2

Return Type of Parameter: Image, Integer

Results: Based on this query Fetcher extracted following components from repository.

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	addImage	1	(0+0+0.5)/3=0.17	0.5	0.54
2	makeImage	0.64	(1+0+0)/3=0.33	0.5	0.49
3	createImage	0.64	(0+0+0)/3=0	0.5	0.41
4	getImage	0.64	(1+0+1)/3=0.67	0.5	0.57
5	imageUpdate	0.64	(0+1/6+0)/3=0.05	0.5	0.42
6	drawImage	0.64	(0+2/4+0)/3=0.16	0.5	0.45

Table 4.8 Calculation of relevance of components
for query "Image Add" with a signatures but not exact

w1=1,w2=1,w3=1

***Constraint Similarity value = (w1 +w2 +w3)/3**

w1=Return type matching value

w2= Parameters type matching value

w3= Value obtained by matching number of Parameters

Case 4:

User Search by using advance option features:

Query: AquaTheme

Author Name: Steve Wilson

In this case System return Following Components

S.No	Class Name	Description	Concept Similarity (A)	*Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/ 2+C)/2
1	AquaTheme	This class describes a theme using "blue-green" colors.	1	0	1	0.75

2	AquaMetalTheme	This class describes a theme using "blue-green" colors.	0.75	0	1	0.68

Table 4.9 Result of Query "AquaTheme" with Advance option

***Constraint Similiarity value = (w1 +w2 +w3)/3**

w1=Return type matching value

w2= Parameters type matching value

w3= Value obtained by matching number of Parameters

Case 5

User gives Conceptual Query and also specify signature but not exact. After retrieving result user changes the component relevancy.

Query: Image Add

Signature:

Return Type: Boolean

Number of Parameter: 2

Return Type of Parameter: Image, Integer

Results: Based on this query, Fetcher extracted the components shown in Table 4.7.

Suppose user assigns the following components weights according to his experience.

S.No.	Method Name	User Assigned Weight
1	addImage	1
2	makeImage	1
3	createImage	0.75
4	getImage	0.45
5	imageUpdate	0.6
6	drawImage	0.3

Table 4.10 Weights given by user

Now next time when user issues the same query, system considered the user experience also and calculated the relevance according to that. Result returned by the system is shown in following table:

S.No.	Method Name	Concept Similarity (A)	Constraint Similarity (B)	User Weight (C)	Total Weight (A*w1+B*w2)/2+C)/2
1	addImage	1	(0+0+0.5)/3=0.17	1	0.79
2	makeImage	0.64	(1+0+0)/3=0.33	1	0.74
3	createImage	0.64	(0+0+0)/3=0	0.75	0.53
4	getImage	0.64	(1+0+1)/3=0.67	0.45	0.55
5	imageUpdate	0.64	(0+1/6+0)/3=0.05	0.6	0.47
6	drawImage	0.64	(0+2/4+0)/3=0.16	0.3	0.35

Table 4.11 Computation of relevance considering user experience

As compared to Table 4.8, considerable improvement has been observed because user experiences are considered with component specification.

Chapter 5

Conclusions and Future Scope

Software reuse is a development method of using existing reusable software components to create new programs to improve both the quality and productivity of software development. By reuse, programmers can avoid repetitive work and focus on the unique features of the problem in hand.

To take maximum advantages from reuse process, there is a need for effective reuse repository system. So a Reuse Repository System with effective and easy to use search mechanism has been implemented.

5.1 Conclusions

A reuse repository system with effective retrieval mechanism has been developed using Conceptual matching and Constraint matching.

To perform Conceptual matching Vector Space Model (VSM) with stop word removal, stemming and thesauruses has been implemented. For implementing Constraint matching, Signature matching is used. Transformation matching is used along with exact matching to make it more efficient.

The Experimental results shown in Section 4.8 (Table 4.5) shows that there is a considerable improvement in retrieval relevancy by using Conceptual matching in hybridation of Constraint matching.

Further, Retrieval relevancy can be improved by considering user experiences in relevancy matching as shown in Section 4.8 (Table 4.11). More the system will be used, it will start gaining user experiences and as a result it will return more accurate and relevant components.

5.2 Future Work

1. Extending reuse repository to other phases of Software development

Although our implementation of repository is designed to promote reuse in the phase of coding, the underlying principles are equally applicable to higher levels of software development activities i.e. Design, testing etc. For example any class diagram can be specified by specifying the signature of the classes it contains and by specify the type of relation between those classes. Reuse system can extract such type of information from component design and use this information for indexing purposes.

2. Supporting More Complicated Indexing and Retrieving Mechanisms

In our implementation we tried to retrieve the components based on conceptual information revealed through comments and constraint information revealed through signatures. However, there are other kinds of information that can be utilized. For example the relationship between classes i.e. inheritance relationships,

Moreover if repository system will use class level comments, comments inside methods and package or class identifiers (i.e. variables) to compute the concept similarity then it will also increase the precision and help in returning the more relevant components.

3. Clustering of results

To properly present the result of a query to the user a clustering engine can be created that can be used to group the ranked list of search results. Clustering is the process of organizing a set of elements into a number of groups or clusters. These clusters should reflect a similarity between the contained elements. Various approaches like flat clustering, Hierarchical clusters [32], Agglomerative Hierarchical Clustering [59] etc can be used to create cluster of resultant relevant components.

4. Performance enhancements

Using multiple threads to solve queries in parallel could probably speed up the returning of results on systems with multiple CPU. The comparison with all document vectors is

currently linear, comparing with one column at a time. This is easily converted into a parallel calculation where each computer can work on a fixed subrange of the matrix.

References

[1] Atkinson S. and Duke R. A methodology for behavioral retrieval from class libraries. Technical Report 94-28, Software Verification Research Centre, Dept. of Computer Science, Univ. of Queensland, Australia, 1994. http://citeseer.nj.nec.com/atkinson94methodology.html.

[2] Atkinson S. and Duke R. Behavioral retrieval from class libraries. Australian Computer Science Communications, 17(1), pages 13-20, January 1995.

[3] Basili, V., Briand, L. & Melo,W. (1996), How Reuse Influences Productivity in Object-Oriented Systems, *Communications of the ACM* 39(10), 104–116.

[4] Boehm, B. (1999), Managing Software Productivity and Reuse, *IEEE Computer* 16(9), 111–113.

[5] Boudriga, N., A. Mili and R.T. Mittermeir (1992), "Semantic-Based Software Retrieval to Support Rapid Prototyping," *Structured Programming 13*, 109–127.

[6] Boudriga, N., A. Mili and R. Zalila (1992a), "An Automated Tool for Specification Validation: Design and Preliminary Implementation," In *Proceedings of the 25th Hawaii International Conference on System Sciences*, IEEE Computer Society Press, Los Alamitos, CA, pp. 74–82.

[7] Chen, Y. and B. Cheng (1997), "Facilitating an Automated Approach to Architecture-Based Software Reuse," In *Proceedings of the 12th IEEE International Automated Software Engineering Conference*, IEEE Computer Society Press, Los Alamitos, CA, pp. 238–246.

[8] Cheng, B.H.C. and J.J. Jeng (1992a), "Formal Methods Applied to Reuse," In *Proceedings of the 5th Annual Workshop on Software Reuse*, WISR-5, Palo Alto, CA, ftp://gandalf.umcs.maine.edu/pub/WISR/wisr5/proceedings/.

[9] Cheng, B.H.C. and J.J. Jeng (1992b), "Reusing Analogous Components," In *Proceedings of the IEEE International Conference on Tools with AI*, IEEE Computer Society Press, Los Alamitos, CA.

[10] Clifton, C. and W.-S. Li (1995), "Classifying Software Components Using Design Characteristics," In *Proceedings of the 10th Knowledge-Based Software Engineering Conference*, KBSE '95, IEEE Computer Society Press, Los Alamitos, CA, pp. 139–146.

[11] Devanbu, P., Brachman, R. J., Selfridge, P. G. & Ballard, B. W. (1991), LaSSIE: A Knowledge-Based Software Information System, Communications of the ACM 34(5), 34–49.

[12] DiFelice, P. & Fonzi, G. (1998), How to Write Comments Suitable for Automatic Software Indexing, *Journal of Systems and Software* 42, 17–28.

[13] Etzkorn, L. H. & Davis, C. G. (1997), Automatically Identifying Reusable OO Legacy Code, *IEEE Computer* 30(10), 66–71.

[14] Fischer, B., M. Kievernagel and G. Snelting (1995), "Deduction-Based Software Component Retrieval," In *Proceedings of the IJCAI Workshop on Reuse of Proofs, Plans and Programs*, German Research Center for Artificial Intelligence, Saarbr¨ucken, Germany, pp. 6–10.

[15] Frakes,W. B. & Pole, T. P. (1994), An Empirical Study of Representation Methods for Reusable Software Components, *IEEE Transactions on Software Engineering* 20(8), 617–630.

[16] Frakes, W.B. and B.A. Nejmeh (1987a), "An Information System for Software Reuse," In *Proceedings of the 10th Minnowbrook Workshop on Software Reuse*, Syracuse University, Minnowbrook, NY.

[17] Frakes, W.B. and B.A. Nejmeh (1987b), "Software Reuse Through Information Retrieval," In *Proceedings of the 20th Annual Hawaii International Conference on System Sciences*, IEEE Computer Society Press, Los Alamitos, CA, pp. 530–535.

[18] Girardi, M. R. & Ibrahim, B. (1995), Using English to Retrieve Software, *Journal of Systems and Software* 30, 249–270.

[19] Hall, R.J. (1993), "Generalized Behaviour-Based Retrieval," In *Proceedings of the 15th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA. pages 371-380, 1993.

[20] Harandi. M.T. The role of analogy in software reuse. *ACM symposium on applied computing states of the art and practice*, 1993.

[21] Helm, R. and Y.S. Maarek (1991), "Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries," In *Proceedings of the OOPSLA '91, SigPlan Notices 26*, 11, 47–61.

[22] Henninger, S. (1997), An Evolutionary Approach to Constructing Effective Software Reuse Repositories, *ACM Transactions on Software Engineering and methodology* 6(2), 111–140

[23] Hirai, Jun; Raghavan, Sriram; Garcia-Molina, Hector; Paepcke, Hector.(1999), " WebBase: A repository of web pages", 9th Intl. WWW Conference

[24] Hoare, C.A.R. (1969), "An Axiomatic Basis for Computer Programming," *Communications of the ACM 12*, 10, 576–583.

[25] Isakowitz, T. and R.J. Kauffman (1996), "Supporting Search for Reusable Software Objects," *IEEE Transactions on Software Engineering 22*, 6, 407–423.

[26] Jeng, J.J. and B.C.H. Cheng (1995), "Specification Matching for Software Reuse: A Foundation," In *Proceedings of the ACM SIGSOFT Symposium on Software Reuse*, SSR '95, ACM Press, New York, NY, pp. 97–105

[27] Jilani, L.L., R. Mili, M. Frappier, J. Desharnais and A. Mili (1997a), "Retrieving Software Components That Minimize Adaptation Effort," In *Proceedings of the 12th IEEE International Automated Software Engineering Conference*, ASE '97, IEEE Computer Society Press, Los Alamitos, CA, pp. 255–262.

[28] Jilani, L.L., R. Mili and A. Mili (1997b), "Approximate Retrieval: An Academic Exercise or a Practical Concern?" In *Proceedings of the 8th Annual Workshop on Software Reuse (WISR-8)*, ftp://gandalf.umcs.maine.edu/pub/WISR/wisr8/proceedings/.

[29] Lucarella, D. and A. Zanzi (1996), "A Visual Retrieval Environment for Hypermedia Information Systems," *ACM Transactions on Information Systems 14*, 1, 3–29.

[30] Maarek, Y.S., D.M. Berry and G.E. Kaiser (1991), "An Information Retrieval Approach for Automatically Constructing Software Libraries," *IEEE Transactions on Software Engineering 17*, 8, 800–813

[31] Maarek, Y.S. and D.M. Berry (1989), "The Use of Lexical Affinities in Requirements Extraction," In *Proceedings of the 5th International Workshop on Software Specification and Design*, IWSSD-5, IEEE Computer Society Press, Los Alamitos, CA, pp. 196–202.

[32] Manning and Sch¨utze, 1999] Manning, C. D. and Sch¨utze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press, Cambridge, Massachusetts. (cited on pp 15, 19, 33, 34)

[33] Michail, A. & Notkin, D. (1999), Assessing Software Libraries by Browsing Similar Classes, Functions and Relationships, in *Proceedings of 21st International Conference on Software Engineering (ICSE'99)*, ACM Press, Los Angeles, CA, pp. 463–472.

[34] Mili, H., Valtchev, P., Di-Sciullo, A., and Gabrini, P. (2001). "Automating the Indexing and Retrieval of Reusable Software Components," *Proceedings of the 6th International Workshop NLDB'01*, June 28 -29, Madrid, Spain, pp. 75 - 86.

[35] Mili, A., Mili, R., & Mittermeir, R. T. (1998). A Survey of Software Reuse Libraries. *Annals of Software Engineering*, 5, 349-414

[36] Mili, A., Mili, R. & Mittermeir, R. (1997a), Storing and Retrieving Software Components: A Refinement-Based System, *IEEE Transaction on Software Engineering* 23(7), 445–460.

[37] Mili, H., E. AhKi, R. Godin and H. Mcheick (1997b), "Another Nail to the Coffin of Faceted Controlled- Vocabulary Component Classification and Retrieval," In *Proceedings of the Symposium on Software Reusability,* SSR '97*, ACM Software Engineering Notes 22*, 3, 89–98.

[38] Mittermeir, R.T. and L. Wuerfl (1995), "Abstract Visualization of Software: A Basis for a Complex Hash-Key," In *Advances in Intelligent Computing*, B. Bouchon-Meunier, R.R. Yager and L.A. Zadeh, Eds., LNCS 945, Springer, Berlin, Germany, pp. 545–554.

[39] Ostertag, E., Hendler, J., Prieto-Diaz, R. & Braun, C. (1992), Computing Similarity in a Reuse Library System: An AI-Based Approach., *ACM Transactions on Software Engineering and Methodology* 1(3), 205–228.

[40] Penix, J., P. Alexander and K. Havelund (1997), "Declarative Specification of Software Architectures," In *Proceedings of the 12th IEEE International Automated Software Engineering Conference*, IEEE Computer Society Press, Los Alamitos, CA, pp. 201–208.

[41] Penix, J. and P. Alexander (1995), "Design Representation for Automating Software Component Reuse," In *Proceedings of the 1st International Workshop on Knowledge Based Systems for the (Re) Use of Software Libraries*, INRIA, Sophia Antipolis, France

[42] Perry, D.E. (1989), "The Inscape Environment," In *Proceedings of the 11th International Conference on Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 2–12.

[43] Perry, D.E. and S.S. Popovich (1993), "Inquire: Predicate-Based Use and Reuse," In *Proceedings of the 8th Knowledge Based Software Engineering Conference*, KBSE '93, IEEE Computer Society Press, Los Alamitos, CA, pp. 144–151.

[44] Podgurski, A. and L. Pierce (1993), "Retrieving Reusable Software by Sampling Behavior," *ACM Transactions on Software Engineering and Methodology 2*, 3, 286–303.

[45] Podgurski A. and Pierce L. Behaviour sampling: A technique for automated retrieval of reusable components. In Proceedings of the 14th International Conference on Software Engineering, pages 349-360,1992.

[46] Porter, M. F. (1980). An algorithm for suffix stripping. Program, 14(3):130–137. (Cited on p 18)

[47] Poulin, J. S.: *Measuring Software Reuse - Principles, Practices, and Economic Models.* Addison-Wesley, 1997

[48] Poulin, J. and K.J.Werkman (1995), "Melding Structured Abstracts and the WorldWide Web for Retrieval of Reusable Components," In *Proceedings of the ACM SIGSOFT Symposium on Software Reuse*, SSR '95, ACM Press, New York, NY, pp. 160–168.

[49] Prieto-Diaz, R. (1991), Implementing Faceted Classification for Software Reuse, *Communications of the ACM* 34(5), 88–97.

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

[51] Rittri, M. (1992), "Retrieving Library Identifiers via Equational Matching of Types," Technical Report 65, Programming Methodology Group, Department of Computer Science, Chalmers University of Technology and University of Geteborg, Geteborg, Sweden.

[52] Rittri, M. (1989), "Using Types as Search Keys in Function Libraries," *The Journal of Functional Programming 1*, 1.

[53] Runciman, C. and I. Toyn (1989), "Retrieving Reusable Software Components by Polymorphic Type," In *Proceedings of the 4th International Conference on Functional Programming Languages and Computer Architectures*, ACM Press, New York, NY, pp. 166–173.

[54] Salton, G.(1989).Automatic Text Processing. Addison-wesley

[55] Salton, G., Yang, C.S., and Wong, A(1975). A Vector-Space model for automatic indexing. Communications of the ACM, 15(1):8-36

[56] Salton, G.(1971) The SMART Retrieval System- Experiments in Automatic Document Processing, chapter Information Analaysis and Dictionary construction, Pages 115-142. Prentice Hall

[57] Simon, H. A. (1996), *The Sciences of the Artificial*, third edition, The MIT Press, Cambridge, MA.

[58] Stringer-Calvert, D. W. J. (1994), Signature Matching for Ada Software Reuse, Master's thesis, University of York, UK.

[59] Wr´oblewski, M. (2003). A hierarchical www pages clustering algorithm based on the vector space model. Master's thesis, Pozna´n University of Technology, Poland.

[60] Ye, Y. (2001). Supporting Component-Based Software Development with Active Component Repository Systems. PhD thesis, University of Colorado.

[61] Ye, Y., Fischer, G. & Reeves, B. (2000), Integrating Active Information Delivery and Reuse Repository Systems, in *Proceedings of ACM SIGSOFT 8th International Symposium on the Foundations of Software Engineering*, ACM Press, San Diego, CA, pp. 60–68.

[62] Zaremski, A. M. &Wing, J. M. (1997), Specification Matching of Software Components, *ACM Transaction on Software Engineering and Methodology* 6(4), 333–369.

[63] Zaremski, A. M. & Wing, J. M. (1995a), Signature Matching: A Tool for Using Software Libraries, *ACM Transactions on Software Engineering and Methodology* 4(2), 146–170.

[64] Zaremski, A.M. and J.M. Wing (1995b), "Specification Matching of Software Components," In *Proceedings of the SIGSOFT '95: 3rd ACM SIGSOFT Symposium on the Foundations of Software Engineering, Software Engineering Notes 20*, 4, 6–17.

[65] Zaremski, A.M., and Wing, J.M. (1993). "Signature Matching: A Key to Reuse," *Software Engineering Notes*, Vol. 18, No. 5, pp. 182-190.

[66] Z. Zhang, L. Svensson, U. Snis, C. Srensen, H. Fgerlind, T. Lindroth, M.Magnusson, C. Stlund. Enhancing Component Reuse Using Search Techniques, Proceedings of IRIS 23. Laboratorium for Interaction Technology, University of Trollhttan Uddevalla, 2000.

Appendix A

Running the System

A.1. Running the system

This appendix describes how the system can be used to index components (Javadoc documentation) and how to configure the system for access using a web-browser.

The following assumes any form of operating system i.e. win 2000, Win xp, Linux or Unix as the platform. This project was developed under win 2000.

A.1.1 Prerequisites

Ensure the following are installed:

- ✓ Web server (IIS is used)
- ✓ Java 1.4 (1.4.2 is used),
- ✓ Visual Basic (visual basic 6.0 is used)
- ✓ Any DBMS (Ms-Acess is used)
- ✓ Java class path is assumed to be set to jdk1.4\bin Dir
- ✓ prj_reuse.dll is properly installed. If not then use windows/system32/regsvr32.exe to install prj_resue.dll
- ✓ Create System DSN (Data Source Name) and name it "reuse"

A.2 Indexing

To perform indexing of components either Run batch file "indexer.bat" Or perform manual indexing.

To perform Manual Indexing Following steps can be used:

1. Set the class path to home directory where indexer module is stored. Also set path variable to Jdk home directory.
2. Compile four java files: reader.java, writer.java, method.java, keyword.java

```
C:\j2sdk\bin > java reader out/output.txt
C:\j2sdk\bin > java writer out/class.txt
C:\j2sdk\bin > java method out/class.txt
C:\j2sdk\bin > java keyword
```

A.3 Accessing the Search Engine User Interface

Assuming that web server runs on localhost: 8080, the search service should

now be available at:

http://localhost: 8080/reuse_repository/

Appendix B

Test Data

This section describes the test data that is used to perform experiment on the implemented reuse system.

The following table shows the Class description extracted from components.

Class Id	Class Name	Extends	Implements	Description	Author	Version
1	Animator	Applet	Runnable, MouseListener	An applet that plays a sequence of images, as a loop or a one-shot. Can have a soundtrack and/or sound effects tied to individual frames. See the Animator home page for details and updates.	Herb Jellinek	1.8, 06/13/02
2	AquaMetalTheme	DefaultMetalTheme	Runnable, MouseListener	This class describe theme using "blue-green" colors.	Steve Wilson	1.7 06/13/02
3	AquaTheme	DefaultMetalTheme	Runnable, MouseListener	This class describes a theme using "blue-green" colors.	Steve Wilson	1.7 06/13/02
4	BulletIcon	Object	Icon	Draws a bullet icon like the one shown on the picture below:	Filigris Works	1.7 06/13/02
5	ButtonDemo	DemoModule	ChangeListener	JButton, JRadioButton, JToggleButton, JCheckBox Demos	Jeff Dinkins	1.9 06/13/02
6	DrawTest	Applet	ChangeListener	---------- ----------	Jeff Dinkins	1.9 06/13/02

Class Id	Class Name	Extends	Implements	Description	Author	Version
				---------- ----------		
7	ImageMap	Applet	Runnable, MouseListener, MouseMotionListener	An extensible ImageMap applet class. The active areas on the image are controlled by ImageArea classes that can be dynamically loaded over the net.	Jim Graham	1.17, 06/13/02
8	CharcoalTheme	DefaultMetalTheme	Runnable, MouseListener, MouseMotionListener	This class describes a theme using gray colors. 1.6 06/13/02	Steve Wilson	1.17, 06/13/02

The following table shows the Method description extracted from components.

Method_ Id	Class Name	Function Name	Function Return Type	Function Parameters	Method Description	Class Id
1	Animator	destroy	String	Void	Applet Destroy	1
2	Animator	getAppletInfo	String	Void	Applet info.	1
3	Animator	getParam	String[][]	String key	Local version of getParameter for debugging purposes.	1
4	Animator	getParameterInfo	void	void	Parameter info.	1
5	Animator	handleParamsi	void	void	Get parameters and parse them	1
6	Animator	init	void	void	Initialize the applet.	1
7	Animator	mouseClicked	void	MouseEvent event	Initialize the applet.	1
8	Animator	mouseEntered	void	MouseEvent event	Initialize the applet.	1
9	Animator	mouseExited	synchronized void	MouseEvent event	Initialize the applet.	1
10	Animator	mousePressed	void	MouseEvent event	Pause the thread when the user clicks the mouse in the applet.	1
11	Animator	mouseReleased	void	MouseEvent	Pause the thread	1

Method_Id	Class Name	Function Name	Function Return Type	Function Parameters	Method Description	Class Id
				event	when the user clicks the mouse in the applet.	
12	Animator	paint	void	Graphics g	Paint the current frame	1
13	Animator	run	void	void	Run the animation.	1
14	Animator	start	synchronized void	void	Start the applet by forking an animation thread.	1
15	Animator	stop	void	void	Stop the insanity, um, applet.	1
16	AquaMetalTheme	update	String	Graphics g	No need to clear anything; just paint.	2
17	AquaMetalTheme	getName	protected ColorUIResource	void	No need to clear anything; just paint.	2
18	AquaMetalTheme	getPrimary1	protected ColorUIResource	void	No need to clear anything; just paint.	2
19	AquaMetalTheme	getPrimary2	protected ColorUIResource	void	No need to clear anything; just paint.	2
20	AquaTheme	getPrimary3	String	void	No need to clear anything; just paint.	3
21	AquaTheme	getName	protected ColorUIResource	void	No need to clear anything; just paint.	3
22	AquaTheme	getPrimary1	protected ColorUIResource	void	No need to clear anything; just paint.	3
23	AquaTheme	getPrimary2	protected ColorUIResource	void	No need to clear anything; just paint.	3
24	BulletIcon	getPrimary3	int	void	No need to clear anything; just paint.	4
25	BulletIcon	getIconHeight	int	void	The icon height.	4
26	BulletIcon	getIconWidth	void	void	The icon width.	4
28	ButtonDemo	addButtons	void	void	This allows to specify the color for the bullet.	5
29	ButtonDemo	addCheckBoxes	void	void	This allows to specify the color for the bullet.	5
30	ButtonDemo	addRadioButtons	void	void	This allows to specify the color	5

Method_Id	Class Name	Function Name	Function Return Type	Function Parameters	Method Description	Class Id
					for the bullet.	
31	ButtonDemo	addToggleButtons	JPanel	void	This allows to specify the color for the bullet.	5
32	ButtonDemo	createControls	void	void	This allows to specify the color for the bullet.	5
33	ButtonDemo	createListeners	Vector	void	This allows to specify the color for the bullet.	5
34	ButtonDemo	getCurrentControls	static void	void	This allows to specify the color for the bullet.	5
35	ButtonDemo	main	void	String[] args	main method allows us to run as a standalone demo.	5
27	ButtonDemo	setColor	void	Color color	This allows to specify the color for the bullet.	5
55	CharcoalTheme	update	protected ColorUIResource	Graphics g	Update the active highlights on the image.	8
56	CharcoalTheme	getBlack	String	void	Update the active highlights on the image.	8
57	CharcoalTheme	getName	protected ColorUIResource	void	Update the active highlights on the image.	8
58	CharcoalTheme	getPrimary1	protected ColorUIResource	void	Update the active highlights on the image.	8
59	CharcoalTheme	getPrimary2	protected ColorUIResource	void	Update the active highlights on the image.	8
60	CharcoalTheme	getPrimary3	protected ColorUIResource	void	Update the active highlights on the image.	8
61	CharcoalTheme	getSecondary1	protected ColorUIResource	void	Update the active highlights on the image.	8
62	CharcoalTheme	getSecondary2	protected ColorUIResource	void	Update the active highlights on the image.	8
63	CharcoalTheme	getSecondary3	protected ColorUIResource	void	Update the active highlights on the image.	8
64	ExampleFileFilter	getWhite	boolean	void	Update the active highlights on the	0

Method_Id	Class Name	Function Name	Function Return Type	Function Parameters	Method Description	Class Id
					image.	
65	ExampleFileFilter	addExtension	String	String extension	Adds a filetype "dot" extension to filter against.	0
66	ExampleFileFilter	getDescription	String	void	Returns the human readable description of this filter.	0
67	ExampleFileFilter	isExtensionListInDescription	void	void	Returns whether the extension list (.jpg, .gif, etc) should show up in the human readable description.	0
68	ExampleFileFilter	setDescription	void	String description	Sets the human readable description of this filter.	0
36	ImageMap	stateChanged	void	ChangeEvent e	main method allows us to run as a standalone demo.	7
37	ImageMap	destroy	String	void	main method allows us to run as a standalone demo.	7
38	ImageMap	getAppletInfo	String[][]	void	main method allows us to run as a standalone demo.	7
39	ImageMap	getParameterInfo	boolean	void	main method allows us to run as a standalone demo.	7
40	ImageMap	imageUpdate	void	Image img, int infoflags, int x, int y, int width, int height	Handle updates from images being loaded.	7
41	ImageMap	init	void	void	Initialize the applet.	7
42	ImageMap	mouseClicked	void	MouseEvent e	Initialize the applet.	7
43	ImageMap	mouseDragged	void	MouseEvent e	Inform all active ImageAreas of a mouse drag.	7
44	ImageMap	mouseEntered	void	MouseEvent e	Inform all active ImageAreas of a mouse drag.	7

Method_Id	Class Name	Function Name	Function Return Type	Function Parameters	Method Description	Class Id
45	ImageMap	mouseExited	void	MouseEvent e	Make sure that no ImageAreas are highlighted.	7
46	ImageMap	mouseMoved	void	MouseEvent e	Find the ImageAreas that the mouse is in.	7
47	ImageMap	mousePressed	void	MouseEvent e	Inform all active ImageAreas of a mouse press.	7
48	ImageMap	mouseReleased	void	MouseEvent e	Inform all active ImageAreas of a mouse release.	7
49	ImageMap	newStatus	void	void	Scan all areas looking for the topmost status string.	7
50	ImageMap	paint	void	Graphics g	Paint the image and all active highlights.	7
51	ImageMap	run	void	void	Paint the image and all active highlights.	7
52	ImageMap	start	synchronized void	void	Paint the image and all active highlights.	7
53	ImageMap	startAnimation	synchronized void	void	Paint the image and all active highlights.	7
54	ImageMap	stop	void	void	Paint the image and all active highlights.	7
55	ImageMap	makeImage	Image	ImageProducer,Image Filter,Integer	Create the Dynamic Image	7
56	ImageMap	createImage	FilterImageSource	ImageProducer,Image Filter,Integer	Create the Image	7
57	ImageMap	addImage	Image	Image	Add one image into another	7
58	ImageMap	getImage	Image	String, String	Return the Image specifed by document base and Code base	7
59	ImageMap	imageUpdate	Boolean	Image , int , int , int , int ,int	Handle updates from images being loaded.	7
60	ImageMap	drawImage	Boolean	Image,int,int, Handle	Draw the image on the Client Area	7

Appendix C

Installation of Doclet/DocFlex

C.1 DocFlex/Doclet (Version 1.4) Installation

This section describes the installation steps of DocFlex/Doclet which is used with JavaDoc tool of JDK.

C.1.1. Windows Installation

To install DocFlex/Doclet on Windows, please follow these steps:

1. Unpack the downloaded archive.

2. Edit **generator. bat** file in DocFlex/Doclet root directory to specify the **JDK** variable according to the location of SDK 1.4.x installed on your system.

Now, everything is ready, click on the generator. bat to run javadoc with Doclet/Docflex

C.1.2. Linux Installation

To install DocFlex/Doclet on Linux, please follow these steps:

1. Unpack the downloaded archive.

2. Copy **linux/generator.sh** shell script file to DocFlex/Doclet root directory and edit it to specify the **JDK** variable according to the location of SDK 1.4.x installed on your system.

3. Edit "Permission" properties of **generator.sh** file to allow it being executed by Linux.

Copy **linux/docflex.config** file to **lib** directory.

Specifically, this file contains settings prepared for Linux to launch automatically a viewer for generated documentation. In particular, those settings specify that each RTF document should be open with OpenOffice.org Writer and each HTML -- with Firefox web browser. If you need to use different applications for opening those files, please edit docflex.config according to the instruction contained in it.

Now, everything is ready, click on the generator. bat to run javadoc with Doclet/Docflex

Paper(s) Communicated/ Accepted/ Published

1. Rajiv Bammi, Rajesh Bhatia, "**Comparative Analysis of Existing Software Repositories** ", International Conference on Rapid Integration of Software Engineering techniques **(RISE 2006)**, University of Geneva**,** Switzerland, 13-15 September 2006**. (Communicated).**