

EFFICIENT RETRIEVAL OF
SOFTWARE COMPONENTS
FROM A REPOSITORY

By

SITARAM DONTU

Bachelor of Technology

Regional Engineering College

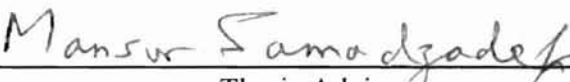
Warangal, Andhra Pradesh, India

1994

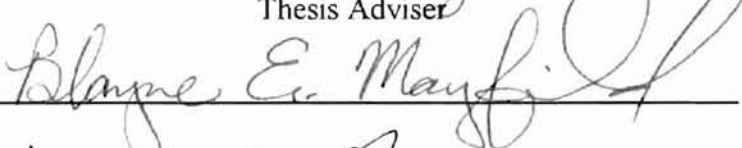
Submitted to the faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December 1996

EFFICIENT RETRIEVAL OF
SOFTWARE COMPONENTS
FROM A REPOSITORY

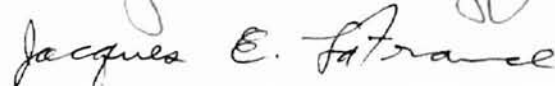
Thesis Approved:




Thesis Adviser



Blayne E. Mayfield



Jacques E. LaFrance



Dean of the Graduate College

PREFACE

Searching for a component in a software repository is a recurring problem in software reuse. The retrieval scheme used for such repositories is syntactic and is generally based on a predefined set of keywords. In many cases the desired component may not be retrieved even though it is present in the repository. This can be attributed to the misspelling of the search pattern, or a different representation of the software component in question by the user or classifier. If the search tool allows for only exact pattern matching, the process of specifying, locating, and retrieving a component can be complex and time consuming, and hence frustrating.

This thesis introduced an inexact search scheme into an already existing repository scheme. The inexact pattern matching was also compared with exact pattern matching. Different levels of inexact searching can be selected. Tests were conducted using the same search patterns for inexact and exact searching. Statistical analysis was applied on the obtained data for both types of searches. Graphs were drawn and compared for exact and inexact search methods. GUI (Graphical User Interface) is provided that reduces the tedium involved in contending with textual interfaces. The work is implemented on a UNIX multi-processor machine (Sequent Symmetry S/81) using C and Motif 1.1. Any terminal supporting the X protocol can be used to display the tool's GUI.

Significant savings in time were achieved in inexact pattern searching over exact pattern searching for retrieval of software components. A total of 27 searches were

conducted for each type (exact and inexact) of search methods. An average of 5 searches were needed to retrieve a desired file in approximate pattern searching, unlike exact pattern matching where an average of 17 searches were needed. The time spent to retrieve a desired file in the inexact search method was approximately 70% less than the time spent in the exact search method, thus saving the user's time and lessening their frustration.

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to my major advisor, Dr. Mansur H. Samadzadeh for his intelligent guidance, constructive criticism, and inspiration. I also wish to thank my other committee members Drs. Blayne E. Mayfield and Jacques E. LaFrance.

I wish to thank my friends for their suggestions and timely humor. I also extend my thanks to others who directly or indirectly helped in the progress of this work.

Finally, I would like to express my sincere gratitude to my parents, brother, and sister for their moral support and encouragement.

TABLE OF CONTENTS

Chapter	Page
I INTRODUCTION	1
II SOFTWARE REUSE AND REPOSITORIES	3
2.1 Software Reuse	3
2.2 Software Classification Systems	3
2.3 Software Repositories	5
2.3.1 Retrieval Methods	5
2.3.1.1 Low-Level Retrieval Methods	5
2.3.1.2 High-Level Retrieval Methods	6
III VCI: A VERSIONING AND REPOSITORY SYSTEM	9
3.1 Design and Implementation	9
3.2 Main Features	12
IV EFFICIENT RETRIEVAL	16
4.1 Agrep: An Approximate Pattern-Matching Tool	17
4.1.1 Features	18
4.1.2 Algorithms	19
V IMPLEMENTATION AND EVALUATION	22
5.1 Implementation	22
5.1.1 Details of the Implementation	22
5.1.2 Usage of the Integrated Search Tool	27
5.2 Evaluation	29
5.2.1 Background	29
5.2.1.1 Collection of Data	30
5.2.1.2 Classification of Data	30
5.2.1.3 Measures of Central Tendency	32
5.2.2 Test Suite	34
5.2.3 Test Results	40
5.2.4 Observations	42
VI SUMMARY, CONCLUSIONS, AND FUTURE WORK	43

Chapter	Page
REFERENCES	45
APPENDICES	48
APPENDIX A - GLOSSARY	49
APPENDIX B - TRADEMARK INFORMATION	50
APPENDIX C - PROGRAM LISTING	51

LIST OF TABLES

Table	Page
I Marks obtained by 64 students in a Thermodynamics class [Source: Grewal 90]	31
II Frequency Distribution Table for the marks obtained by 64 students in a Thermodynamics class [Source: Grewal 90]	33
III Number of hits for all the files in the repository for approximate and exact pattern matches	35
IV Frequency Distribution Table for exact pattern matching	40
V Frequency Distribution Table for approximate pattern matching	41

LIST OF FIGURES

Figure	Page
1. A delta chain in RCS [Source: Sobell 95]	10
2. Initial Screen with all the option buttons [Source: Nadella 95]	13
3. Zoom-out mode of the display of RCS file structure [Source: Nadella 95]	14
4. Pattern Search Dialog Box	23
5. Dialog Box prompting the user to select one of the options for the Libout process [Source: Nadella 95]	24
6. Comparison between approximate pattern matching and exact pattern matching	28
7. Frequency Distribution Graph showing the number of files in each hit interval for exact and approximate pattern matching	42

CHAPTER I

INTRODUCTION

Although software productivity has been on the increase over the years, the software industry has been having difficulty meeting the high demand for software productivity and quality [Mili et al. 95] [Boehm 87] [Cox 90]. According to Mili et al., “nothing short of an order of magnitude increase in productivity will extricate the software industry from its perennial crisis” [Mili et al. 95].

Software reuse appears to possess the potential to increase software productivity and quality. For software reuse to work, it is necessary to maintain a software repository containing reusable software components. Users can check in or check out reusable components from the repository. The reusable components must be classified, stored, and retrieved in a cost effective and efficient way [Fernandez-Chamizo et al. 95].

There has been a lot of attention on the efficient retrieval of software components from repositories [Zand and Samadzadeh 95] [Girardi and Ibrahim 95]. Researchers have come up with various solutions for the retrieval of software components. Most of the proposed repository schemes have been shown to be difficult to use or not practically useful for software reuse. GUIs (Graphical User Interfaces) are used to make it easy for users to retrieve software components from a repository [Schlubbier 95].

Typically, the proposed search tools for retrieving software components allow for only exact pattern matching (for patterns capturing and representing the characteristics and features of the components). As a result, a desired component may not be retrieved even though it is present in the repository. This can be attributed to the contention that human reasoning is to a large extent approximate and inexact rather than precise in nature [Zadeh 65].

The above discussion can be used as a motivation to construct an intelligent search tool which can incorporate and exploit inexactness in pattern matching. Such a search tool, which was designed and implemented as part of this thesis work, reduces a user's effort in retrieving a desired component and hence lessens the frustration incurred otherwise.

This thesis concentrated on the idea of an intelligent search tool. It integrated an approximate pattern matching method into an existing exact matching tool that was built on top of a version control system [Nadella 95]. Nadella's work involved developing a software reuse assistant with a GUI to a software repository.

The rest of this thesis report is organized as follows. Chapter II introduces software reuse and a number of retrieval methods. Chapter III discusses an existing repository system (VCI) based on a versioning system (RCS). Chapter IV describes approximate pattern matching. Chapter V gives the implementation and evaluation details, and Chapter VI is the concluding chapter containing the summary, conclusions, and future work.

CHAPTER II

SOFTWARE REUSE AND REPOSITORIES

2.1 Software Reuse

Software reuse is the process of creating software systems from existing software rather than from scratch [Krueger 92]. It is aptly justified in the present competitive world of increasing pressure on the software industry for product quality and developer productivity.

It is argued that reuse in general reduces the amount of work to be done by reusing previously written and thoroughly tested functions stored in a software repository. However, software reuse involves other parameters such as storage, retrieval, and classification of software objects [Fernandez-Chamizo et al. 95]. This thesis work concentrated on the issue of retrieval of software components in an efficient manner [Mansur et al. 96] by utilizing approximate pattern matching.

2.2 Software Classification Systems

Software objects are classified based on different criteria. To support effective retrieval, the software components have to be represented by some formal or informal specification. Once they are classified, the retrieval becomes easy. Some of the important classification systems are given in the following paragraphs.

Prieto-Diaz [Prieto-Diaz and Freeman 87] used a faceted classification scheme, in which they used six facets to describe each component (Function, Object, Medium, System Type, Function Area, and Setting), some related to the environment and some related to the functionality. The different values a facet can have are called terms. These terms are organized in a conceptual graph that represents manually encoded knowledge about the domain.

Swanson and Samadzadeh [Swanson and Samadzadeh 92] implemented the ideas of Prieto-Diaz and Freeman [Prieto-Diaz and Freeman 87], and developed a prototype that can be used to catalog and retrieve software components. They also implemented the “terms thesaurus” suggested by Prieto-Diaz and Freeman.

Embley and Woodfield [Embley and Woodfield 87] classified the software components as a knowledge structure consisting of ADTs (Abstract Data Types). Different relationships among the ADTs are supported in this knowledge structure, and it helps in browsing and finding software documents using keywords and natural language descriptions.

Wood and Sommerville [Wood and Sommerville 88] proposed a frame-based software component catalogue. It has a frame which describes the main function performed by a software component and the slots inside the frame specify the objects manipulated by the software component.

The LaSSIE system [Devanbu et al. 91] is composed of a knowledge base which helps users to understand the whole software system and also helps in retrieving a desired component from the repository. It supports a GUI (Graphical User Interface).

The GURU system [Maarek et al. 91] classifies software based on attributes

automatically extracted from their natural language documentation by using an indexing scheme based on the notions of lexical affinity and quantity of information.

2.3 Software Repositories

A software repository is a virtual storage or depository for software components. Repositories are indispensable nowadays where the reuse of software artifacts is on the rise. A software repository grows as new software components are added. But at the same time the disk space should be utilized in an efficient way. Software configuration management tools can be used to save disk space.

2.3.1 Retrieval Methods

Software component retrieval is the process of navigating the user through a software repository for the retrieval of the desired component(s). Retrieval tools attempt to retrieve components based on a user's queries. The efficiency of a retrieval tool is measured by the amount of time spent and the accuracy attained in retrieving a desired software component. The following subsections briefly discuss low-level and high-level retrieval methods.

2.3.1.1 Low-Level Retrieval Methods Software reuse dates back a long time to the use of statistical and mathematical subroutines. Various methods have been used for software component retrieval from repositories. Most of the primitive tools retrieve components based directly on code documentation [Fernandez-Chamizo et al. 95]. As a result, effective retrieval is directly proportional to the quantity and quality of the documentation available in the code.

This is a good approach but fails when the components are inadequately documented.

2.3.1.2 High-Level Retrieval Methods In recent years, new methods of retrieval have been developed that are based not only on external documentation available, but also on semantic information [Fernandez-Chamizo et al. 95]. Most of the new methods use the automatic indexing approach. This approach is based on the semantic and lexical classification of the software components. Artificial Intelligence can also contribute in this area. The important requirement of this approach is that it needs pre-encoded semantic information about software artifacts.

Some researchers proposed the use of templates [Burton et al. 87] [Frakes and Nejme 87] for retrieving components. Prieto-Diaz [Prieto-Diaz and Freeman 87] proposed a faceted classification scheme for organizing software artifacts. This scheme uses six facets to describe each software document. The different values that a facet can have are called terms. This is represented in a conceptual graph. This semantic information is utilized in retrieving a software component. The six facets are described below.

1. **Function:** The function performed by a software component.
2. **Object:** The objects which are manipulated in the software component.
3. **Medium:** The entities that served as the “locale of action”, such as files and tables.
4. **System Type:** Functionally identifiable modules, e.g., a number sorter
5. **Function Area:** The keywords describing the application area of the software component, e.g., a manufacturing department, a quality control department, etc.
6. **Setting:** The environment where the software component will be used, e.g., a chip

manufacturing plant, a steel plant, etc.

Wood and Sommerville [Wood and Sommerville 88] proposed a frame based classification of software modules. The frames are constructed manually. The frames contain slots which describe the objects manipulated by a module. Here also semantic information is provided in the frames, and this information is utilized in the retrieval process.

Automatic text indexing systems automatically extract the natural language specifications provided by the user, and these attributes are used in the retrieval of software components. Software components in the repository are classified based on the terms extracted from the natural language documentation of the software modules. Here the retrieval systems mostly depend on the lexical elements present in the natural language documentation, and they ignore the syntactic and semantic information found in the documentation. The GURU system [Maarek et al. 91] follows this type of retrieval approach.

Knowledge-based systems, unlike the automatic indexing method, try to understand the software modules even more closely by analyzing the natural language specifications syntactically and semantically. This helps a retrieval system in retrieving a software document more accurately than it would otherwise. But here a large amount of pre-encoded information must be provided for each software module. LaSSIE system [Devanbu et al. 91] is an example of this kind of approach.

Although different approaches for component retrieval have been proposed, most of them allow for no errors in the query posed by a user. So the retrieval of components heavily depends on the exactness of the user's query. But, as previously noted in this thesis report, human reasoning is generally approximate. Some of the components may not be retrieved

even if they are present in the repository, because of the exactness of the query. A tool which allows for inexactness in queries is therefore necessitated. Such a tool can search the knowledge base (the software repository) for the requested item and retrieve all the related software components.

CHAPTER III

VCI: A VERSIONING AND REPOSITORY SYSTEM

This chapter discusses Nadella's [Nadella 95] work and the software package that he developed (as part of his thesis) called VCI (Version Control Interface). The present thesis work was built on top of VCI, replacing some of its parts and significantly improving it. This process can be considered a good example of software reuse because the existence of a software repository is a prerequisite for this thesis work, and VCI fills this prescription, thus saving valuable time and letting this thesis concentrate on the retrieval of software components. The following sections discuss the design, implementation issues, and important features of VCI.

3.1 Design and Implementation

VCI has a GUI (Graphical User Interface) to RCS (Revision Control System) and combines the concept of software reuse with that of version (configuration) management. The notion of RCS is discussed in the next paragraph which will explain how VCI used it as a library package to the software repository.

RCS is a configuration management tool available on most of the UNIX and DOS-based systems. A large project involving many people can have problems of coordination, keeping track of different versions of files, simultaneous updates of source code, etc. In these

situations, RCS can be used to keep track of all the source code and documentation files in a large project [Sobell 95]. Although it can be used on any text file, RCS is most often used to “manage source code and software documentation” [Sobell 95].

RCS keeps track of each update done on a particular file by writing down the author’s name, the changes made in the code, the reason, and the date the code was updated. Each change is called a *delta* and it is identified by a version number consisting of two or four components. The four components are called *release*, *level*, *branch*, and *sequence* numbers. An original RCS file is assigned a version number of 1.1. The subsequent delta’s are assigned version numbers 1.2, 1.3, and so on. When major changes are done, the release number can be changed. When a branch is created, the four components are used to form a version. For example version 2.1.1.1 (see Figure 1) is the result of application of the first delta to the first branch on Release 2, Level 1. Further nodes in that branch would be version(s) 2.1.1.2, 2.1.1.3, and so on.

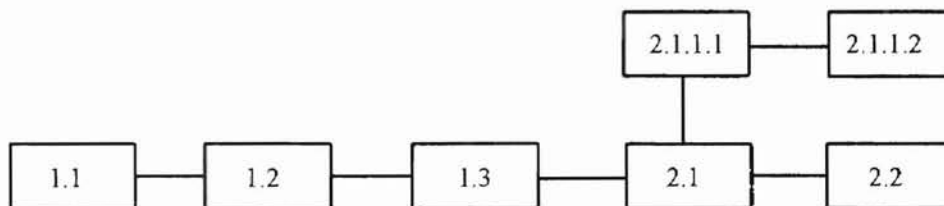


Figure 1. A delta chain in RCS [Source: Sobell 95]

RCS saves only the deltas (the changed lines in the source code from the previous version) and not the entire changed files, thus attempting to save space. However for a small

project, the savings might not be significant, since RCS files themselves (i.e., the deltas) take some space to store the information for each update.

VCI uses RCS as a helping hand in the construction of a repository. It uses RCS to conserve disk space in the repository by making the checked-in components as versions of previously checked-in components. The time to access (check out) the files is generally increased as a tradeoff to the savings in disk space.

VCI maintains a structure or dependence hierarchy of the files as stored in the repository using RCS. This structure is the backbone of VCI. The structure helps in storing and retrieving a file. Because of the obvious importance of it, only the system administrator can alter the structure of the files in the repository. Hence a file checked in by any user is deposited tentatively in the repository only as an experimental file and not as a delta in the RCS chain. The information about the file, the user who checked in the file, and other relevant details are written in an information file. The VCI system administrator later checks the information file for all the deposits, verifies their usefulness and re-checks the useful files as versions of existing files in the repository. The structure of the files in the repository is altered to reflect the changes in the new RCS files in the repository.

A typical file to be checked in is to be accompanied by the following information:

- Author's name
- Author's E-mail address
- Function. The main function performed by the software component
- Method: The objects (data types) manipulated by the software module

- **Implementation Details:** A brief description of the implementation details of the software module

During the retrieval process, the above information is searched. VCI uses exact pattern matching to retrieve a desired file.

VCI uses the concept of distance [Prieto-Diaz 89] to capture some aspects of the nature of relationships among software modules stored in the repository. It also uses a thesaurus to help the users find the documents using keywords which are similar in meaning, the reason being that a file retrieved using a keyword might not be retrieved using a synonymous keyword. VCI was written using C and Motif 1.1. The tool (VCI) was built on the Oklahoma State University Computer Science Department's multi-processor machine Sequent Symmetry S/81 running DYNIX/ptx.

3.2 Main Features

This section discusses the important features of VCI (Version Control Interface) [Nadella 95].

VCI is a tool that is built on top of an established version management tool (i.e., RCS) to provide a software repository. The GUI provided by VCI is intended to be user friendly. The initial screen with all the option buttons is shown in Figure 2. VCI is password protected and only authorized persons can use the repository. Normal users can check in or check out files from the library. Only the library administrator or VCI system manager has the final say over whether a file can be deposited as an RCS file, which helps in eliminating arbitrary

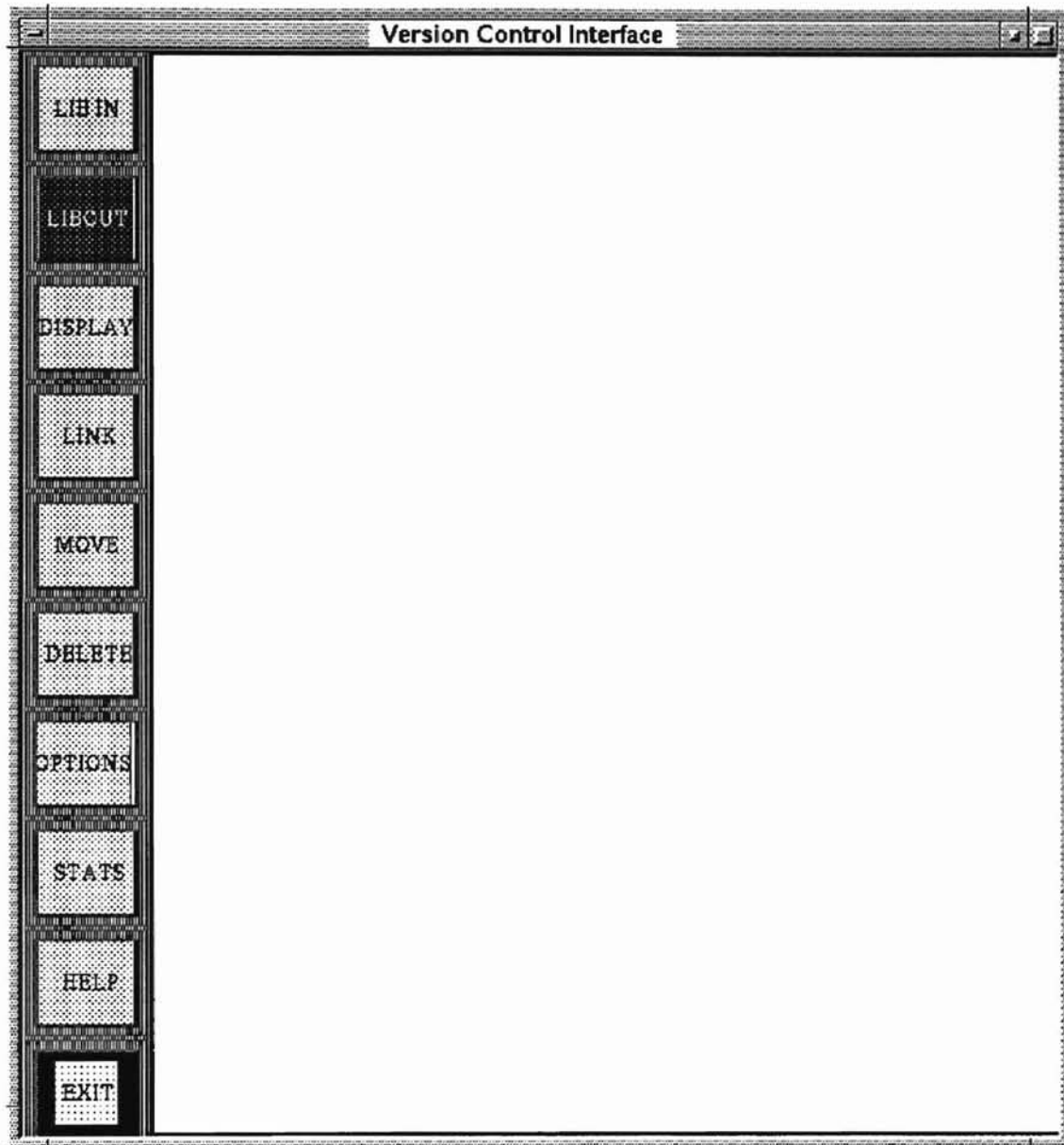


Figure 2. Initial Screen with all the option buttons [Source: Nadella 95]

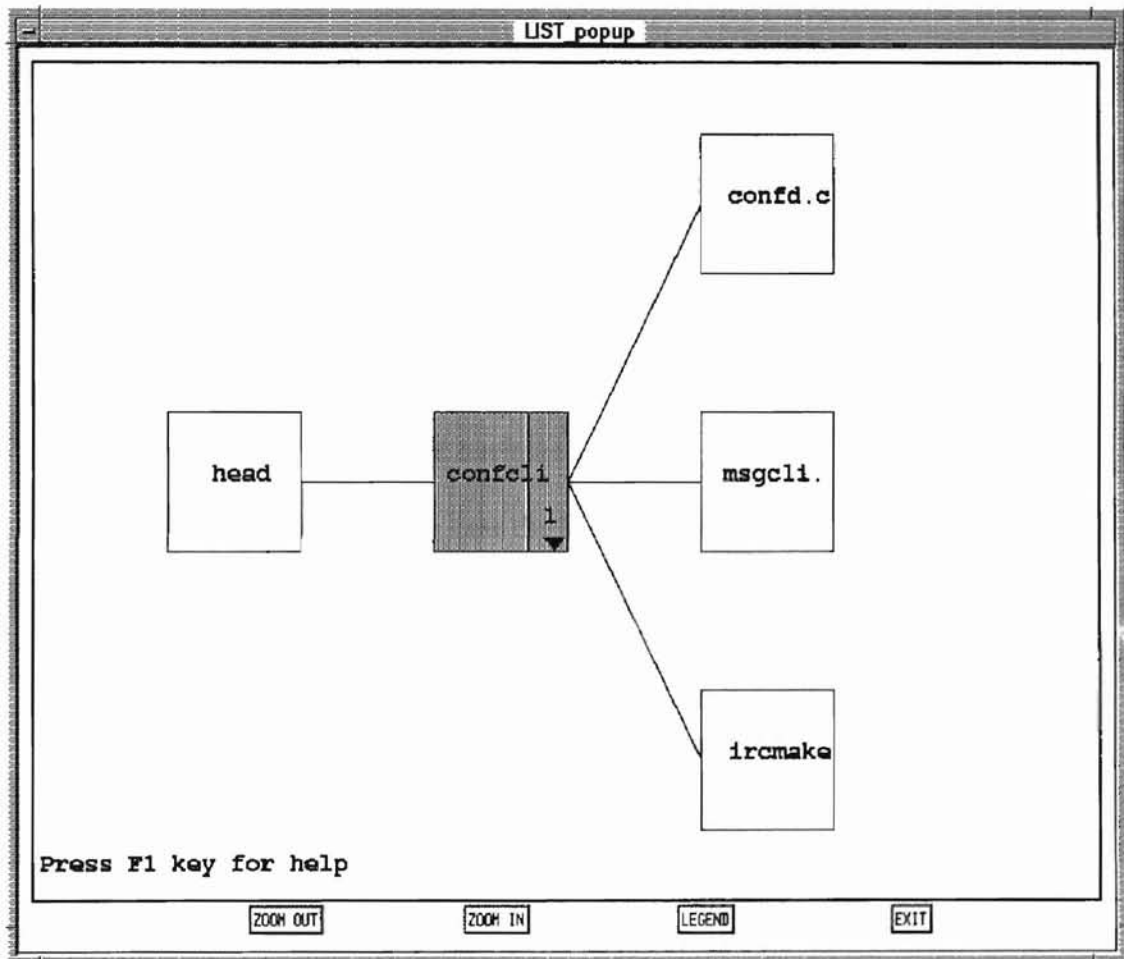


Figure 3. Zoom-out mode of the display of RCS file structure [Source: Nadella 95]

additions to the library and optimizing the RCS structure for efficient space utilization.

VCI helps save disk space for the repository by exploiting RCS which is a widely used software configuration management system. VCI provides graphs on the space savings achieved and the file checkout frequency. It uses BLT [McLennan 93] (the language built on Tcl/Tk) to draw the graphs.

VCI provides the option to change the structure of the repository in order to provide faster access to most frequently checked out files. The move and delete buttons allow for moving a file to anywhere in the repository and deleting a file. This is a novel feature of VCI that helps change the library hierarchy based on the usage (i.e., reuse history) of the files in the repository.

The GUI for VCI helps a user navigate through a visual display of the entire library structure of the repository. A user can view the library structure starting from a specific node and optimally zoom in or zoom out for a low level or high level view of the hierarchy. If no specific node is given by a user, the entire library structure is displayed (see Figure 3).

CHAPTER IV

EFFICIENT RETRIEVAL

Efficient retrieval means that a required software document is obtained from the repository quickly. To achieve this goal, an approximate pattern matching method was incorporated in the retrieval process of VCI (see chapter III for a detailed discussion of VCI). Approximate string matching is the process of finding a string B which is either “nearly exact” or the same as a given string A. It is used in diverse areas such as information retrieval, pattern recognition, error correction, and molecular genetics [Ukkonen 85].

Various techniques have been used for the problem of approximate string matching using Neural Networks and Dynamic Programming. Neural Networks apply fuzzy logic theory while Dynamic Programming employs dynamic programming algorithms for the problem of approximate string matching.

Many exact string matching algorithms are in use today. Such algorithms are used in UNIX grep, Perl, GNU Emacs, etc. However, only a few algorithms have been proposed for approximate string matching. Among the present approximate string matching methods, agrep [Wu and Manber 92], which is a tool for fast approximate pattern matching and is based on the well-known Knuth-Morris-Pratt algorithm [Knuth et al. 77], and the Boyer-Moore algorithm [Boyer and Moore 77], is noteworthy.

A brief introduction to the agrep tool as well as some of its important features and algorithms are included in the following subsections.

4.1 Agrep: An Approximate Pattern-Matching Tool

Most of the ideas in this subsection are adapted from the paper written by Wu and Manber [Wu and Manber 92], which forms the basis for this thesis work.

We search for patterns quite often in different circumstances, for instance in a file containing text or source code. But most often we are disappointed for being unable to locate the pattern we search for, the most plausible reason being the misspelling of the pattern. So, an approximate pattern searching method can be used to bring up all the strings or words that match a pattern “nearly exactly”.

For example, consider the case where we are searching for a string $S = s_1 s_2 \dots s_n$ inside a text file T . All the substrings “nearly equal” to S can be found under some criteria of approximation. Under such criteria, we can state that a string $S1$ is at a distance D from a string $S2$, if $S2$ can be obtained from $S1$ by any sequence of ‘ D ’ insertions, deletions, or substitutions of single characters in any place in $S1$.

A tool, which searches a database for all the terms nearly exactly matching a query, was developed by Wu and Manber [Wu and Manber 92]. This tool is called agrep (which stands for approximate grep) and is very similar to the UNIX grep family. It makes some important additions to the grep family. Agrep supports wild cards, sets of patterns, and regular expressions in addition to a number of other types of queries. Agrep is relatively fast (except when the number of errors is very large) compared to the well known algorithm for

approximate matching to arbitrary regular expressions by Myers and Miller [Myers and Miller 89].

In this thesis, `agrep` was used to replace the exact pattern matching used in the reuse library tool developed by Nadella [Nadella 95] for the retrieval of desired components from a software repository. The significant features of `agrep` are discussed below.

4.1.1 Features

The following three features of `agrep` are the most important additions of the `agrep` family to the `grep` family of pattern matching/search tools.

a. Searching for approximate patterns

agrep -3 Hello Mail

The above query searches for all words that can be obtained from the string *Hello* by at most three substitutions, insertions, or deletions from the file called *Mail*. Different costs can be given to insertions, deletions, and substitutions. Consider the following query.

agrep -1 -S2 -D2 tom students

This query will find all the names that can be obtained by inserting at most one character in the string *tom* in the file called *students*. No substitutions or deletions of single characters are allowed as the cost assigned to them is two, while the number of errors allowed is only one.

b. Record oriented rather than just line oriented

`Agrep` by default outputs only the line(s) containing a given pattern. But it can be user defined to output all the lines containing the required 'pattern' delimited by a 'specific string' at the beginning and at the end. This is called a record. The example below outputs the

whole record containing a 'pattern'.

```
agrep -d 'a tab' pattern Thesis
```

The above query searches for the string *pattern* in all the records starting and ending with *a tab* ('a tab' represents the *TAB* key stroke) in the file called *Thesis*.

c. Sets of patterns with AND (or OR) logic queries

A logical query can be formed by using the logical operators AND and OR implicitly.

```
agrep -d 'a tab' 'pattern1,pattern2' Thesis
```

The above query searches all the records starting and ending with *a tab* and outputs those records containing either *pattern1*, *pattern2*, or both. Similarly, we can create a query with AND (',' stands for OR and ';' stands for AND).

```
agrep -d 'a tab' 'pattern1;pattern2' Thesis
```

We can form a complex query, as the one below, by combining all the above features.

```
agrep -d 'a tab' -2 'pattern1;pattern2;<199[1-5]>' database
```

This query outputs all the records, containing *pattern1*, *pattern2* and a year between 1990 and 1996 with at most 2 errors in any of the sub-patterns, from a file called *database*. '< >' doesn't allow for the occurrence of any errors in the string between the corner brackets.

4.1.2 Algorithms

This subsection briefly describes the algorithms used in *agrep* by Wu and Manber [Wu and Manber 92]. *Agrep* uses a slightly modified Boyer-Moore algorithm for simple exact patterns and a partition scheme for simple patterns with errors. It uses new algorithms for patterns with unlimited wild cards, patterns with uneven costs with different edit operations,

multi-patterns, and arbitrary regular expressions.

The main approach for finding simple patterns with errors is given. This algorithm is taken as is from Wu and Manber's work [Wu and Manber 92]. The algorithm is based on the 'shift-or' algorithm of Baeza-Yates and Gonnet [Baeza-Yates and Gonnet 89]. Let $P = p_1 p_2 \dots p_n$ be the search string to be searched in a large text file $T = t_1 t_2 \dots t_n$ and R be a bit array of size m (the size of the pattern). We denote by R_j the value of array R after the j th character of the text has been processed. The bit array R_j contains information about all matches of prefixes of P with a suffix of the text that ends at j . More precisely, $R_j[i] = 1$ if the first i characters of the pattern match exactly the last i characters up to j in the text. These are all partial matches that may lead to full matches later on. When we read t_{j+1} we need to determine whether t_{j+1} can extend any of the partial matches so far. The transition from R_j to R_{j+1} can be summarized as follows.

Initially, $R_0[i] = 0$ for all i , $1 \leq i \leq m$, and $R_0[0] = 1$.

$$R_{j+1}[i] = \begin{cases} 1 & \text{if } R_j[i-1] = 1 \text{ and } p_i = t_{j+1} \\ 0 & \text{otherwise} \end{cases}$$

If $R_{j+1}[m] = 1$, then we output a match that ends at position $j+1$.

This transition, which we have to compute once for every text character, seems quite complicated. However, suppose $m \leq 32$ (which is usually the case in practice) and that R is represented as a bit vector using one 32-bit word. For each character s_i in the alphabet, we construct a bit array S_i of size m such that $S_i[r] = 1$ if $p_r = s_i$ (It is sufficient to construct the S array only for the characters that appear in the pattern). It is easy to verify now that the

transition from R_j to R_{j+1} amounts to no more than a *right shift* of R_j and an AND operation with S_j , where $s_j = t_{j+1}$. So, each transition can be executed with only two simple arithmetic operations, a shift and an AND. The algorithm for multi-patterns uses a hashing technique combined with a different Boyer-Moore Algorithm.

CHAPTER V

IMPLEMENTATION AND EVALUATION

5.1 Implementation

5.1.1 Details of the Implementation

The code responsible for pattern searching, written by Nadella [Nadella 95], was completely removed and new code was added to bring approximate pattern matching into focus.

A dialog box (see Figure 4) with all the features necessary to facilitate easy retrieval of software components from the library, using approximate pattern matching, was developed. The dialog box contains the following components.

1. Text Box: For typing a query (search pattern).
2. Option Menu: To select the number of errors allowed in the search pattern during the search process.
3. Scrolled List: To display the retrieved files.
4. Scrolled Text: To display information about the file selected in the Scrolled List.
5. Frame Box: To display which file is selected.
6. Action Area: To contain Check out, Search, and Cancel buttons.

The screen which leads to the dialog box mentioned above is shown in Figure 5.

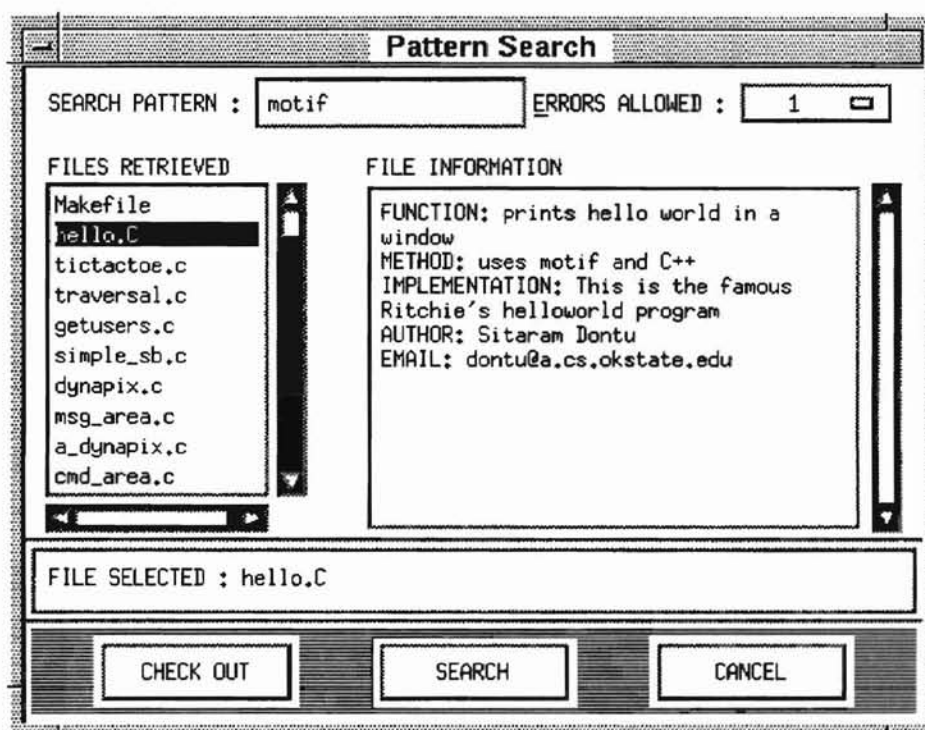


Figure 4. Pattern Search Dialog Box

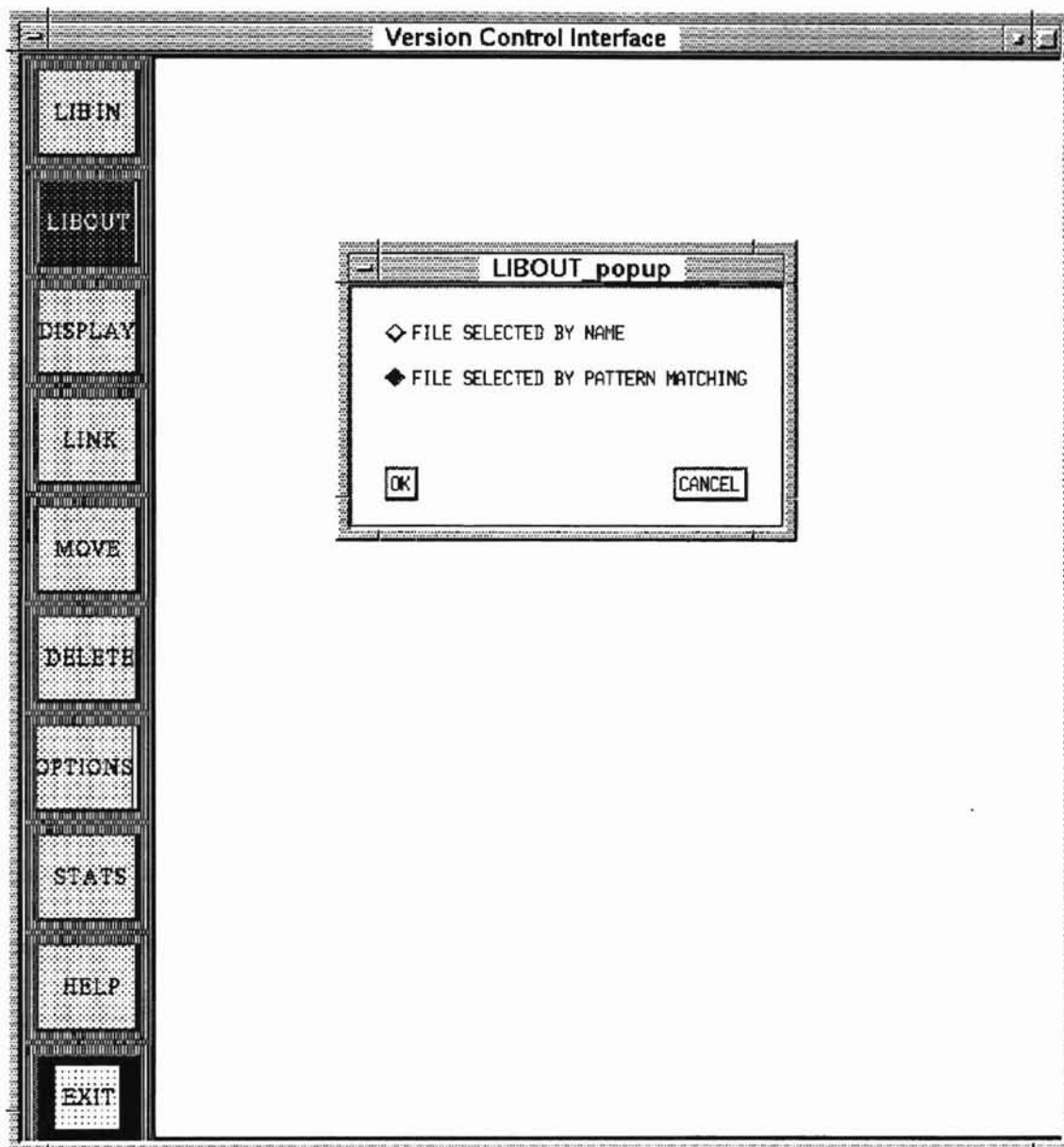


Figure 5. Dialog Box prompting the user to select one of the options for the Libout process [Source: Nadella 95]

The text box (see figure 4) allows the following types of search patterns.

a. Word sequence, e.g.,

Sitaram Dontu

which constitutes a query as a sequence of words. The resulting query will be a search according to the criteria chosen (exact or approximate pattern matching) by matching a selection from the Option menu which is explained later in this subsection. Let us consider the above query, i.e., *Sitaram Dontu*. If exact pattern matching is selected, a file name is retrieved only if it contains the query exactly as given, unlike the case for approximate pattern matching, where the case sensitivity of the search pattern is ignored and the number of errors allowed in the search pattern is controlled.

b. Pattern formed by the logical connectives AND and OR, e.g.,

motif;unix (where ';' represents the AND logical connective)

motif,unix (where ',' represents the OR logical connective)

Thus, a query can be formed by using logical connectives between words. For example, the query, *motif;unix* means that the file to be searched must contain both of the words motif and unix. Two or more words can be connected logically, however the AND and OR logical connectives cannot be intermixed in a single pattern. The case sensitivity of the pattern is ignored in approximate pattern matching.

Using the above two patterns, we can form a complex query such as the one given below.

Pattern1;Pattern2;<199[1-5]>

As a result of such a query, a file is retrieved only if it contains *Pattern1*, *Pattern2*, and a year

between 1990 and 1996. The notation '<>' indicates that the pattern does not allow for any errors in the string between the corner brackets.

The Option menu (see Figure 4) is similar to a Pulldown menu. When it is clicked with the left mouse button, it presents a list of choices. In the Pattern Search dialog box (see Figure 4), the Option menu contains five choices. The first choice is *exact* for exact pattern matching and the rest of the choices, 0, 1, 2, and 3, are for approximate pattern matching, which control the number of errors allowed in the search pattern.

Based on a repository of 100 programs and 27 searches conducted, *agrep* took approximately 0 - 5 seconds to search a file for a given query. As a basis for comparison, an assorted set of 100 files was searched sequentially, the time taken was approximately 500 seconds (over 8 minutes). That is a comparatively long time to wait, and it increases as the number and size of files in the repository increases. So, to reduce the time taken to search the repository, each search should be made independent of the other searches. This is possible by forking a separate process for each search. We should be able to collect the results of all such independent searches carried out by different forked processes. For this we need shared memory. Obviously, no two search processes must write in the shared memory at the same time. This was achieved by a binary semaphore. Hence shared memory, a binary semaphore, and fork system calls were made use of in the search process. This brought down the search time for 100 files to approximately 40 seconds.

The following data (provided by a user while checking in a software component into the repository) is searched in every file for a given query.

1. Author

2. E-mail address of the author
3. The function of the file
4. A description about the objects manipulated by the file
5. A brief description of the implementation details of the file

Graphs were drawn using BLT [Mclennan 93] (a language built on Tcl/Tk) which compare approximate pattern matching and exact pattern matching. All the necessary parameters required for each graph were calculated and then a file containing all the BLT commands was created which was run to create the graph. An option is provided in each graph which automatically generates a postscript file, if so desired and selected. An example of such a graph is shown in Figure 6.

5.1.2 Usage of the Integrated Search Tool

The repository was filled with 100 'C' files obtained from various sources such as O'Reilly's Motif, Volume 1, written by Heller and Ferguson [Heller and Ferguson 91]; the '/contrib/src' directory from the Oklahoma State University Computer Science Department's Sequent multi-processor machine; personal files, etc. After approximate grep was introduced into VCI, the search process retrieved more files than it used to retrieve using exact pattern matching. This made the search process generally more productive.

A dialog box is provided (see Figure 4) for the retrieval process. The following steps can be followed in the process of retrieving a desired file from the repository.

1. Type a query in the Text Box as explained in the previous subsection (see Subsection 5.1.1). The size of the query must be greater than the number of errors allowed when

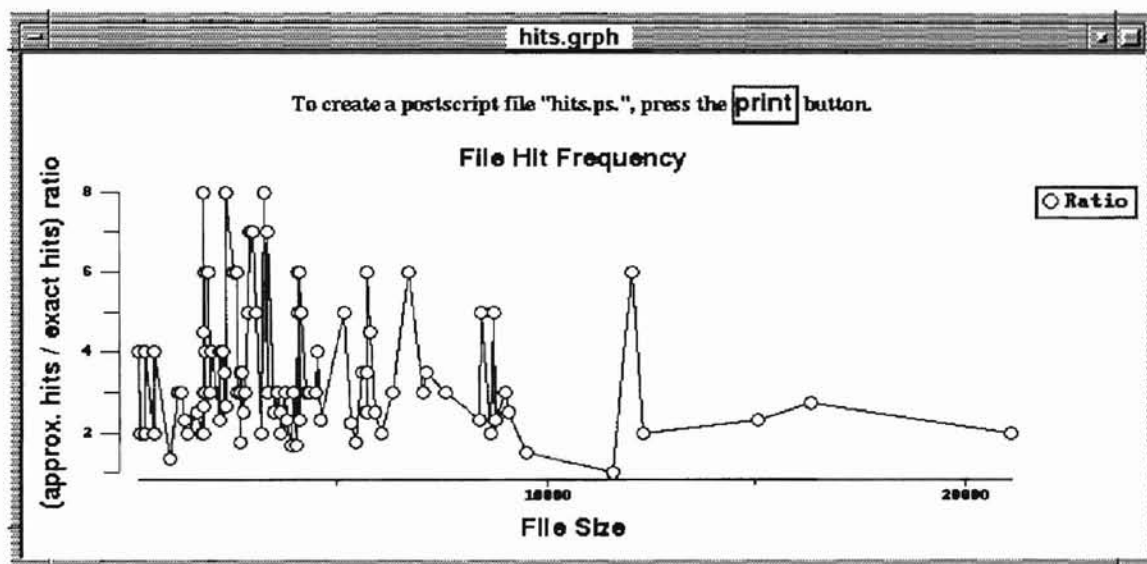


Figure 6. Comparison between approximate pattern matching and exact pattern matching

approximate pattern matching is selected.

2. Click the Search button in the Action Area with the left mouse button, or press Enter in the Text Box after the query is typed to start the search process.
3. Select the desired file by double clicking it.
4. Select the Check Out button in the Action Area to checkout the file into the current directory.

The cursor shape is changed to a watch once the search process begins. This means that the user has to wait for some time before the user can interact with the dialog box again. The retrieved files are displayed in the Scrolled List Box named Files Retrieved. Double clicking a file shows the description of the file in the Scrolled Text Box named File Information, and the Frame Box just below the Scrolled List Box displays the name of the file selected. Pressing the Check Out button in the Action Area checks out the selected file into the current directory from which VCI is run. Appropriate error dialogs pop up, if necessary, during the retrieval process.

5.2 Evaluation

5.2.1 Background

The following discussion is adapted from Grewal's Higher Engineering Mathematics text [Grewal 90].

Statistics deals with methods for collection, classification, and analysis of numerical data for drawing valid conclusions and making reasonable decisions. It has meaningful

applications in production engineering, in the analysis of experimental data, etc. The importance of statistical methods in engineering is generally on the increase.

5.2.1.1 Collection of Data The collection of data constitutes the starting point of any statistical investigation. It should be carried out systematically with a definite aim in view. Also data collection should be conducted with as much accuracy as is desired in the final results, for detailed analysis would not compensate for the bias and inaccuracies in the original data. Data may be collected for each and every unit of the whole lot (*population*), for it would ensure greater accuracy. But complete enumeration is prohibitively expensive and time-consuming. As such, out of a very large number of items, a few of them (*a sample*) are collected and conclusions drawn on the basis of that sample are taken to hold for the population. A sample should however be a *random sample*, i.e., it should be obtained without bias or showing preferences in selecting sample items from the population.

5.2.1.2 Classification of Data The data collected in the course of an inquiry is not in an easily assimilable form. As such, its proper classification is necessary for making meaningful inferences. The classification is done by dividing the raw data into a convenient number of groups according to the values of the variable and finding the frequency of the variable in each group.

Let us, for example, consider the raw data relating to the marks obtained in a Thermodynamics course by a group of 64 students (see Table I). The data can be grouped and shown in a tabular form (see Table II). Table II shows that there is one student getting marks between 50 - 54, two students getting marks between 55 - 59, nine students getting

marks between 60 - 64, and so on. Thus the 64 figures have been put into 10 groups, called the classes. The width of the class is called the *class interval* and the number in that interval is called its *frequency*. The mid-point or the mid-value of a class is called the *class mark*. Table II, showing the classes and the corresponding frequencies, is called a frequency table. Thus a set of raw data summarized by distributing it into a number of classes along with their frequencies is known as a *frequency distribution*.

TABLE I

Marks obtained by 64 students in a Thermodynamics class [Source: Grewal 90]

79	88	75	60	93	71	59	85
84	75	82	68	90	62	88	76
65	75	87	74	62	95	78	63
78	82	75	91	77	69	74	68
67	73	81	72	63	76	75	85
80	73	57	88	78	62	76	53
62	67	97	78	85	76	65	71
78	89	61	75	95	60	79	83

While forming a frequency distribution, the number of classes should not ordinarily exceed 20, and should not, in general, be less than 10. As far as possible, the class intervals should be of equal width.

In some investigations, the number of items is required to be less than a certain value. We add up the frequencies of the classes up to that value and call this number the *cumulative frequency*. In Table II, the third column shows the cumulative frequencies, i.e., the number of students getting less than 54 marks, less than 59 marks, and so on.

The condensation of data in the form of frequency distribution is very useful as far as it converts a long series of observations into a compact form. But in practice we are generally interested in comparing two or more series. The inherent inability of the human mind to grasp in its entirety even data in the form of a frequency distribution compels us to seek for certain constants which could concisely give an insight into the important characteristics of the series. The chief constants which summarize the fundamental characteristics of a series are *Measure of central tendency, Measure of dispersion, and Measure of skewness*.

5.2.1.3 Measures of Central Tendency A frequency distribution, in general, shows clustering of the data around some central value. Finding this central value or the average is of importance, as it gives a most representative value of the whole group. Different methods give different averages which are known as the *measures of central tendency*. The commonly used measures of central value are mean, median, and mode.

Mean is calculated as follows. If X_1, X_2, \dots, X_n are a set of n values of a variate, the mean is given by

$$\text{Mean} = \frac{X_1 + X_2 + \dots + X_n}{n}, \text{ i.e., } \frac{\sum X_i}{n}.$$

In a *frequency distribution*, where the frequencies for the values X_1, X_2, \dots, X_n are F_1, F_2, \dots, F_n respectively, we have

$$\text{Mean} = \frac{F_1X_1 + F_2X_2 + \dots + F_nX_n}{n}, \text{ i.e., } \frac{\sum F_iX_i}{\sum F_i}.$$

The same formula will also hold good for a grouped distribution except that the values X_1, X_2, \dots, X_n will then correspond to the mid-points of the classes.

TABLE II

Frequency Distribution Table for the marks obtained by 64 students in a Thermodynamics class [Source: Grewal 90]

Class	Frequency	Cumulative Frequency
50 - 54	1	1
55 - 59	2	3
60 - 64	9	12
65 - 69	7	19
70 - 74	8	27
75 - 79	17	44
80 - 84	6	50
85 - 89	8	58
90 - 94	3	61
95 - 99	3	64

5.2.2 Test Suite

In the retrieval of software components, as far as this thesis is concerned, we are interested in how fast we can retrieve a desired component and how much (whether) the approximate pattern matching is an improvement over exact pattern matching. We take samples (data obtained from pattern searches), classify them into small groups, and deduce the average number of hits, say 'm' hits for any desired file in 'n' searches. It means that any desired file in the repository can be brought 'm' times on the average in 'n' searches. From this we can calculate the average number of searches required to retrieve a desired file from the repository, and it can be used for comparisons between approximate and exact pattern matching.

The discussion in the background subsection (see Subsection 5.2.1) can now be applied to the data obtained from pattern searches. The tables containing the data are given in the following pages.

TABLE III

Number of hits for all the files in the repository
for approximate and exact pattern matches

File Name	Number of Hits (Approximate Pattern Matching)	Number of Hits (Exact Pattern Matching)
Makefile	3	0
a_dynapix.c	5	0
action_area.c	5	2
alpha_list.c	4	0
app_scroll.c	10	3
arrow.c	4	1
arrow_timer.c	5	1
ask_user.c	6	1
ask_user_simple.c	5	0
ask_user_simple1.c	4	0
build_menu.c	5	1
build_option.c	5	1
cmd_area.c	5	1
color_draw.c	6	2
color_slide.c	5	0
confcli.c	5	1
confd.c	5	1
copy_by_name.c	6	3
copy_retrieve.c	6	2
corners.c	7	3

TABLE III (Continued)

Number of hits for all the files in the repository
for approximate and exact pattern matches

File Name	Number of Hits (Approximate Pattern Matching)	Number of Hits (Exact Pattern Matching)
cut_paste.c	5	1
dialog.c	5	0
draw2.c	8	3
drawing.c	9	5
drawn.c	6	3
dynapix.c	4	1
editor.c	6	2
entry_cb.c	6	2
error_test.c	4	0
expand.c	2	0
file_browser.c	6	1
file_sel.c	8	1
fill.c	0	0
fillmake	1	0
fontlist.c	3	0
form_corners.c	7	2
frame.c	5	2
free_hand.c	7	3
friends.c	2	0
getusers.c	9	3

TABLE III (Continued)

Number of hits for all the files in the repository
for approximate and exact pattern matches

File Name	Number of Hits (Approximate Pattern Matching)	Number of Hits (Exact Pattern Matching)
hello.C	3	2
hello_dialog.c	5	0
help_text.c	7	3
incr_retrieve.c	6	2
inet.h	3	0
inquire.c	4	1
ircmake	3	0
main_list.c	7	3
map_dlg.c	5	1
modal.c	6	0
modify_btn.c	5	0
modify_text.c	9	1
monitor_sb.c	4	1
msg_area.c	5	3
msgcli.c	5	0
msgd.c	4	0
multi_click.c	5	1
multi_font.c	5	0
paned_win1.c	7	2
paned_win2.c	7	2

TABLE III (Continued)

Number of hits for all the files in the repository
for approximate and exact pattern matches

File Name	Number of Hits (Approximate Pattern Matching)	Number of Hits (Exact Pattern Matching)
password.c	4	0
pixmap.c	7	0
popups.c	6	2
prompt_dlg.c	7	0
prompt_phone.c	6	1
prompt_phone2.c	6	0
pushb.c	7	1
query_retrieve.c	6	2
radio_box.c	3	0
reason.c	6	0
replace.c	5	1
rowcol.c	6	2
rtr.C	2	0
rtrtree.dat	1	0
rtrtreefunc.C	1	0
rtrtreemake	1	0
script.c	2	0
search_list.c	4	1
search_text.c	5	1
select_dlg.c	6	1

TABLE III (Continued)

Number of hits for all the files in the repository
for approximate and exact pattern matches

File Name	Number of Hits (Approximate Pattern Matching)	Number of Hits (Exact Pattern Matching)
select_text.c	5	1
show_files.c	5	1
show_pix.c	5	0
simple_list.c	4	1
simple_popup.c	11	3
simple_pullright.c	11	2
simple_radio.c	3	0
simple_sb.c	5	1
simple_scale.c	5	0
spreadsheet.c	5	2
string.c	7	0
text_entry.c	5	2
text_form.c	6	1
tictactoe.c	6	1
toggle.c	3	0
traversal.c	5	1
undo.c	4	2
unit_types.c	8	1
warning.c	5	1
xcal.c	4	0

5.2.3 Test Results

The total number of searches conducted was 27. The data (see Table III) is meaningfully classified below. The following table is for exact pattern matching.

TABLE IV
Frequency Distribution Table for exact pattern matching

Class Interval	Frequency (F _i)	Mid-point of Interval (X _i)	F _i X _i
0 - 2	70	1	70
2 - 4	29	3	87
4 - 6	1	5	5
6 - 8	0	7	0
8 - 10	0	9	0
10 - 12	0	11	0
12 - 14	0	13	0
14 - 16	0	15	0
16 - 18	0	17	0
18 - 20	0	19	0
$\sum F_i = 100$		$\sum (F_i X_i) = 162$	

$$\text{Mean} = \frac{\sum (F_i X_i)}{\sum F_i} = \frac{162}{100} = 1.62$$

That means that for any file an average of 1.62 hits is obtained in 27 searches. From this we

can calculate the number of searches needed to find any desired file. Average number of searches per file (Total number of searches / Average number of hits) = $27 / 1.62 \approx 17$.

The following table is for approximate pattern matching.

TABLE V

Frequency Distribution Table for approximate pattern matching

Class Interval	Frequency (F_i)	Mid-point of Interval (X_i)	$F_i X_i$
0 - 2	5	1	5
2 - 4	12	3	36
4 - 6	44	5	220
6 - 8	30	7	210
8 - 10	6	9	54
10 - 12	3	11	33
12 - 14	0	13	0
14 - 16	0	15	0
16 - 18	0	17	0
18 - 20	0	19	0
$\sum F_i = 100$		$\sum(F_i X_i) = 558$	

$$\text{Mean} = \frac{\sum(F_i X_i)}{\sum F_i} = \frac{558}{100} = 5.58$$

So, an average of 5.58 hits for any file in the repository is noted. Average number of searches per file (Total number of searches / Average number of hits) = $27 / 5.58 \approx 5$.

The following graph is drawn based on the above two tables.

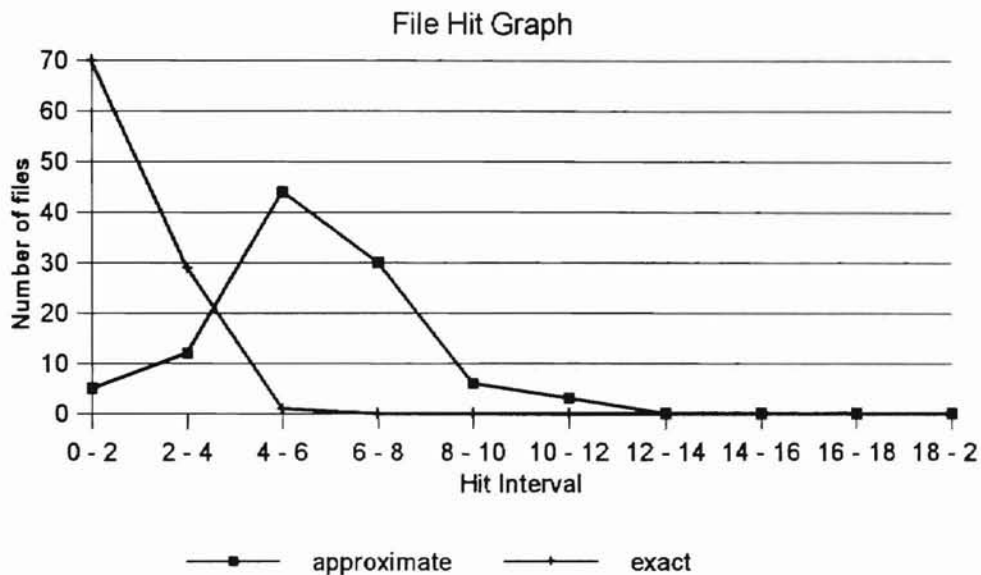


Figure 7. Frequency Distribution Graph showing the number of files in each hit interval for exact and approximate pattern matching

5.2.4 Observations

From the above results, we note that a desired file from the repository is retrieved faster with approximate pattern matching than exact pattern matching. The number of searches in approximate pattern matching can be improved if the number of errors in the search pattern is increased, but sometimes this will bring up files which are a lot different from the user specifications.

CHAPTER VI

SUMMARY, CONCLUSIONS, AND FUTURE WORK

Software reuse helps increase software productivity and quality. But it also involves storage, retrieval, and classification of software objects. Retrieval of software components from a repository is a recurring problem. Exact pattern matching is used in most of the repository schemes, which reduces the effectiveness of retrieving a software document from a repository. Approximate pattern matching overcomes the above deficiency by reducing the users' effort in the retrieval significantly. Agrep is a fast approximate pattern search algorithm. It was integrated into an existing repository scheme (VCI), which employed exact pattern matching. A Dialog Box was provided for the retrieval process using agrep.

A prototypical experiment was conducted that involved storing 100 files in the repository and comparing approximate pattern search over exact pattern search. The same search patterns were used for both approximate and exact pattern matching. The number of searches conducted was 27. The number of hits for all the files in the repository was noted in both cases (approximate and exact pattern matching). Graphs were drawn comparing approximate and exact pattern matching. From the graphs, it was deduced that on an average 5 searches were needed to retrieve a desired file, unlike exact pattern matching where an average of 17 searches were needed. Hence the time spent in approximate matching to

retrieve a desired file was approximately 70% less than the time spent in exact pattern matching.

The storage structure of the files, stored as RCS files in the repository, can still be finely tuned to reduce the access times for the files to be checked out. This can be done by dynamically changing the storage structure. Those files which are frequently accessed can be brought up closer to the root, which can potentially reduce future file access times . To capture this essence, a controlled study is needed to explore the different ways that the storage structure should be restructured dynamically.

REFERENCES

- [Baeza-Yates and Gonnet 89] R. A. Baeza-Yates and G. H. Gonnet, "A New Approach to Text Searching", *Proceedings of the 12th Annual ACM-SIGIR Conference on Information Retrieval*, pp. 168-175, Cambridge, MA, June 1989.
- [Boehm 87] B. Boehm, "Improving Software Productivity", *IEEE Software*, Vol. 4, No. 5, pp. 43-57, September 1987.
- [Boyer and Moore 77] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm", *Communications of the ACM*, Vol. 20, No. 10, pp. 762-772, October 1977.
- [Burton et al. 87] B. A. Burton, R. W. Aragon, S. A. Bailey, K. D. Koehler, and L. A. Mayes, "The Reusable Software Library", *IEEE Software*, Vol. 4, No. 4, pp. 25-33, July 1987.
- [Cox 90] B. J. Cox, "Planning the Software Revolution", *IEEE Software*, Vol. 7, No. 6, pp. 25-35, November 1990.
- [Devanbu et al. 91] P. Devanbu, R. Brachman, P. Selfridge, and B. Ballard, "LaSSIE: A Knowledge-Based Software Information System", *CACM*, Vol. 34, No. 5, pp. 34-49, May 1991.
- [Embley and Woodfield 87] D. W. Embley and S. N. Woodfield, "A Knowledge Structure for Reusing Abstract Data Types in Ada Software Production", *Proceedings of the Joint Ada Conference, Fifth National Conference on Ada Technology and Washington Ada Symposium*, pp. 27-34, U.S. Army Communications-Electronics Command, Fort Monmouth, NJ, 1987.
- [Fernandez-Chamizo et al. 95] Carmen Fernandez-Chamizo, Pedro A. Gonzalez-Calero, Luis Hernandez-Yanez, and Alvaro Urech-Baque, "Case-Based Retrieval of Software Components", *Expert Systems with Applications*, Vol. 9, No. 3, pp. 397-405, 1995.
- [Frakes and Nejme 87] W. B. Frakes and B. A. Nejme, "Software Reuse Through Information Retrieval", *The 32nd IEEE Computer Society International Conference, Digest of Papers: Intellectual Leverage*, pp. 380-384, San Francisco, CA, February 1987.

- [Girardi and Ibrahim 95] M. R. Girardi and B. Ibrahim, "Using English to Retrieve Software", *The Journal of Systems and Software*, Vol. 30, No. 3, pp. 249-270, September 1995.
- [Grewal 90] B. S. Grewal, "Statistical Methods", *Higher Engineering Mathematics*, Khanna Publishers, pp. 758-760, New Delhi, India, 1990.
- [Heller and Ferguson 91] Dan Heller and Paula M. Ferguson, *Motif Programming Manual*, O'Reilly and Associates, Inc., Sebastapol, CA, 1991.
- [Knuth et al. 77] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast Pattern Matching in Strings", *SIAM Journal on Computing*, Vol. 6, No. 2, pp. 323-350, June 1977.
- [Krueger 92] Charles W. Krueger, "Software Reuse", *ACM Computing Surveys*, Vol. 24, No. 2, pp. 131-179, June 1992.
- [Maarek et al. 91] Y. Maarek, D. Berry, and G. Kaiser, "An Information Retrieval Approach For Automatically Constructing Software Libraries", *IEEE Transactions on Software Engineering*, Vol. 17, No. 8, pp. 800-813, August 1991.
- [McLennan 93] Michael J. McLennan, BLT, Copyright © 1993 AT&T Bell Laboratories.
- [Mili et al. 95] Hafedh Mili, Fatma Mili, and Ali Mili, "Reusing Software: Issues and Research Directions", *IEEE Transactions on Software Engineering*, Vol. 21, No. 6, pp. 528-559, June 1995.
- [Myers and Miller 89] E. W. Myers and W. Miller, "Approximate matching of regular expressions", *Bull. Of Mathematical biology* 51, pp. 5-37, 1989.
- [Nadella 95] S. C. Nadella, "A User-Friendly Interface to RCS and Its Use as a Software Repository", Master's Thesis, Computer Science Department, Oklahoma State University, Stillwater, OK, 1995.
- [Preito-Diaz 89] R. Preito-Diaz, "Classification of Reusable Modules", *Software Reusability*, Vol. I, Ted J. Biggerstaff and Alan J. Perlis, Eds., pp. 99-124, Addison-Wesley Publishing Company, NY 1989.
- [Prieto-Diaz and Freeman 87] R. Prieto-Diaz and P. Freeman, "Classifying Software for Reusability", *IEEE Software*, Vol. 4, No. 1, pp. 6-16, January 1987.
- [Samadzadeh et al. 96] Mansur H. Samadzadeh, S. C. Nadella, S. Dontu, and M. K. Zand, "Software Repository: Efficient Storage and Retrieval", *Proceedings of the Sixth International Conference on Information Processing and Management of*

Uncertainty in Knowledge Based Systems (IPMU'96), Special Track on Software Reusability, Vol. III, pp. 1129-1135, Granada, Spain, July 1996.

- [Schluebier 95] Alan C. Schluebier, "The Future Is Reuse", *Computer World*, Vol. 29, Issue 19, pp. 77, May 1995.
- [Sobell 95] Mark G. Sobell, *A Practical Guide to the UNIX System*, Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1995.
- [Swanson and Samadzadeh 92] J. E. Swanson and Mansur H. Samadzadeh, "A Reusable Software Catalog Interface", *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing*, pp. 1076-1086, Kansas City, MO, March 1992.
- [Ukkonen 85] Esko Ukkonen, "Algorithms for Approximate String Matching", *Information and Control*, Vol. 64, Nos. 1-3, pp. 100-118, January-March 1985.
- [Wood and Sommerville 88] M. Wood and I. Sommerville, "An Information Retrieval System for Software Components", *ACM SIGIR Forum*, Vol. 22, Nos. 3-4, pp. 11-25, Spring/Summer 1988.
- [Wu and Manber 92] Sun Wu and Udi Manber, "Agrep: A Fast Approximate Pattern-Matching Tool", *Proceedings of the Winter 1992 USENIX Conference*, pp. 153-162, San Francisco, CA, January 1992.
- [Zadeh 65] L. A. Zadeh, "Fuzzy Sets", *Information and Control*, Vol. 8, pp. 338-353, June 1965.
- [Zand and Samadzadeh 95] Mansour Zand and Mansur H. Samadzadeh, "Software Reuse: Current Status and Trends", *The Journal of Systems and Software*, Vol. 30, No. 3, pp. 167-170, September 1995.

APPENDICES

Appendix A

GLOSSARY

ADT	Abstract Data Type, a user defined type with a set of operations defined on it.
BLT	A library of extensions to the Tk toolkit.
Delta	When one version is stored fully, the other versions are represented based on differences from this version; the differences are called deltas.
Dialog	A secondary (transient) window in a graphical user interface which has reduced functionality unlike the main (top-level) window.
GUI	Graphical User Interface, a visual representation of some of the functionality of a program that can be manipulated in a friendly, easy-to-use, and non-programmatic manner.
Pattern	A string supplied by a user as (part of) a query.
RCS	Revision Control System, a configuration management tool.
Repository	A virtual storage or depository for software components.
Software Component	A piece of code (or software artifact in general).
Tcl	Tool Command Language, a scripting language that is used for developing and using graphical user interface applications.
Tk	A toolkit based on Tcl that helps users create graphical user interfaces for the X11 Window System by writing Tcl scripts.
VCI	Version Control Interface, a tool (developed by Sunil Nadella [Nadella 95]) which provides a graphical user interface to a repository.

Appendix B

TRADEMARK INFORMATION

DYNIX/ptx	A registered trademark of Sequent Computer Systems, Inc.
Motif	A registered trademark of Open Software Foundation (OSF).
NCD	A registered trademark of Network Computing Devices, Inc.
Sequent Symmetry S/81	A registered trademark of Sequent Computer Systems, Inc.
UNIX	A registered trademark of UNIX System Laboratories, Inc.
X	A registered trademark of Massachusetts Institute of Technology (MIT).

WINDUPTA STATE UNIV. LIBRARY

Appendix C

PROGRAM LISTING

The program files are presented in this appendix. The following files contain the new code that is added to Nadella's program files (old code) [Nadella 95]. As a whole (new code + old code), functions as a single entity. The new code depends on some of the data structures defined in the old code. The pattern search functionality is completely replaced by the new code.

The order of the files as given in the following pages is given below:

`app_grep.c`

`app_sear.c`

`hits.c`

`graph.c`

`head.h`

```

/*
 * * * * *
 *   Filename      : app_grep.c
 * * * * *
 *   Programmed by : Dontu Sitaram
 * * * * *
 *   Last updated on : Oct 30 1996
 * * * * *

```

This file contains the code to display the window and controls in it during the check out process using pattern search (approximate pattern search).

```
*/
```

```

#include <Xm/DialogS.h>
#include <Xm/Frame.h>
#include <Xm/Text.h>
#include <Xm/TextF.h>
#include <Xm/Form.h>
#include <Xm/Label.h>
#include <Xm/List.h>
#include <Xm/PushButton.h>
#include <Xm/PanedW.h>

```

```
/*
```

```

* This function creates the pattern search dialog and its controls.
* A search pattern is typed in a text box after selecting the
* the number of errors allowed. The retrieved files are displayed
* in a scrolled list window and the information about a file can be
* displayed in a scrolled text window. The action area contains
* 3 buttons. The check out button checksout any file selected. A
* file can be selected by double clicking on it in the scrolled
* text window. The search process can be initiated either by
* clicking on the search button or pressing return in the pattern
* text box after typing the pattern. Cancel button pops down the
* dialog.

```

```
*/
```

MADHURIA STATE UNIV. LIBRARY


```

void option_cb(), annul();

extern void a_search(), sel_callback(), a_checkout();
extern void a_checkout();

/*
 * A static dialog is created and remains in memory
 * until the application is terminated.
 */
if (!top)
{
    /*
     * arr is a static array of widgets used to store handles
     * to needed widgets in this dialog box. The reason we go
     * for this type of storage is none of the widgets are made
     * global and it is also not desirable.
     */
    arr = (Widget *) XtMalloc (6 * sizeof(Widget));

    /*
     * Get the x and y coordinates of the top left corner of
     * the top level window.
     */
    XtVaGetValues (toplevel, XmNx, &x, XmNy, &y, NULL);
    x = x + 140;
    y = y + 125;

    /*
     * Dialog shell.
     */
    top = XtVaCreatePopupShell ("Pattern Search", xmDialogShellWidgetClass,
                                toplevel,
                                XmNx,          x,
                                XmNy,          y,
                                NULL);

    /*
     * Manager widget - panedWindow.
     */
    pane = XtVaCreateWidget ("pane", xmPanedWindowWidgetClass, top,
                            XmNsashWidth, 1,
                            XmNsashHeight, 1,
                            NULL);

    /*
     * Link the help callback function.
     */
    XtAddCallback(pane, XmNhelpCallback, help, 25);

    arr[5] = pane;

    /*
     * Pane widget is used to store the initial default
     * value for the number of errors allowed in the pattern.
     */
    XtVaSetValues (pane, XmNuserData, 0, NULL);

    /*
     * Manager widget - form widget in panedWindow.
     */
    form = XtVaCreateWidget ("form", xmFormWidgetClass, pane, NULL);

```

```

/*
 * Manager widget - form1 (form) widget in
 * form widget.
 */
form1 = XtVaCreateWidget ("form1", xmFormWidgetClass, form,
                        XmNleftAttachment, XmATTACH_FORM,
                        XmNtopAttachment, XmATTACH_FORM,
                        XmNrightAttachment, XmATTACH_FORM,
                        XmNfractionBase, 16,
                        NULL);

str = XmStringCreateSimple ("SEARCH PATTERN :");

/*
 * Label widget "SEARCH PATTERN" in form1 widget.
 */
label = XtVaCreateManagedWidget ("label", xmLabelWidgetClass, form1,
                                  XmNlabelString, str,
                                  XmNleftAttachment, XmATTACH_FORM,
                                  XmNleftOffset, 7,
                                  XmNtopAttachment, XmATTACH_FORM,
                                  XmNbottomAttachment, XmATTACH_FORM,
                                  NULL);

/*
 * Free the string after it is used.
 */
XmStringFree (str);

/*
 * Text widget class in form1 widget, to type
 * the user's query.
 */
a_text = XtVaCreateManagedWidget ("a_text", xmTextWidgetClass, form1,
                                   XmNleftAttachment, XmATTACH_WIDGET,
                                   XmNleftWidget, label,
                                   XmNleftOffset, 2,
                                   XmNrightAttachment, XmATTACH_POSITION,
                                   XmNrightPosition, 9,
                                   XmNtopAttachment, XmATTACH_FORM,
                                   XmNbottomAttachment, XmATTACH_FORM,
                                   NULL);

arr[0] = a_text;

/*
 * If return is pressed in the text widget class,
 * the search routine is activated.
 */
XtAddCallback (a_text, XmNactivateCallback, a_search, arr);

str = XmStringCreateSimple ("ERRORS ALLOWED :");
zero = XmStringCreateSimple (" exact ");
one = XmStringCreateSimple (" 0 ");
two = XmStringCreateSimple (" 1 ");
three = XmStringCreateSimple (" 2 ");
four = XmStringCreateSimple (" 3 ");

/*
 * Option menu for selecting the number of errors
 * in the query.
 */
option = XmVaCreateSimpleOptionMenu (form1, "option", str, 'E',
                                     0 /*initial menu selection*/, option_cb,
                                     XmVaPUSHBUTTON, zero, 'e', NULL, NULL,

```



```

XmVaPUSHBUTTON, one, '0', NULL, NULL,
XmVaPUSHBUTTON, two, '1', NULL, NULL,
XmVaPUSHBUTTON, three, '2', NULL, NULL,
XmVaPUSHBUTTON, four, '3', NULL, NULL,
XmNrightAttachment, XmATTACH_FORM,
XmNrightOffset, 7,
XmNtopAttachment, XmATTACH_FORM,
XmNbottomAttachment, XmATTACH_FORM,
NULL);

/*
 * Free all the strings no longer needed.
 */
XmStringFree (str);
XmStringFree (zero);
XmStringFree (one);
XmStringFree (two);
XmStringFree (three);
XmStringFree (four);

/*
 * Manger widget - form2 (form) widget class
 * under form widget.
 */
form2 = XtVaCreateWidget ("form2", xmFormWidgetClass, form,
                        XmNleftAttachment, XmATTACH_FORM,
                        XmNtopAttachment, XmATTACH_WIDGET,
                        XmNtopWidget, form1,
                        XmNtopOffset, 10,
                        XmNrightAttachment, XmATTACH_FORM,
                        XmNbottomAttachment, XmATTACH_FORM,
                        XmNfractionBase, 16,
                        NULL);

str = XmStringCreateSimple ("FILES RETRIEVED");

/*
 * Label widget "FILES RETRIEVED" in form2
 * form widget.
 */
label = XtVaCreateManagedWidget ("label", xmLabelWidgetClass, form2,
                                  XmNlabelString, str,
                                  XmNleftAttachment, XmATTACH_FORM,
                                  XmNleftOffset, 7,
                                  XmNtopAttachment, XmATTACH_FORM,
                                  NULL);

XmStringFree (str);

n = 0;

/*
 * The scrolled list widget shows a maximum of
 * 10 files. If there more than 10 files, vertical
 * and horizontal scrollbars show up.
 */
XtSetArg (args [n], XmNscrollBarDisplayPolicy, XmSTATIC); n++;
XtSetArg (args [n], XmNlistSizePolicy, XmRESIZE_IF_POSSIBLE); n++;
XtSetArg (args [n], XmNvisibleItemCount, 10); n++;
XtSetArg (args [n], XmNleftAttachment, XmATTACH_FORM); n++;
XtSetArg (args [n], XmNleftOffset, 7); n++;
XtSetArg (args [n], XmNrightAttachment, XmATTACH_POSITION); n++;
XtSetArg (args [n], XmNrightPosition, 5); n++;
XtSetArg (args [n], XmNtopAttachment, XmATTACH_WIDGET); n++;

```

```

XtSetArg (args[n], XmNtopWidget,          label); n++;
XtSetArg (args[n], XmNbottomAttachment,   XmATTACH_FORM); n++;

/*
 * Scrolled list widget in form2 (form) widget.
 */
s_list = XmCreateScrolledList (form2, "s_list", args, n);
arr[1] = s_list;

/*
 * Callback registered for double clicking on
 * an item in the scrolled list. The callback
 * function is called to provide description
 * of the item (file) selected here.
 */
XtAddCallback (s_list, XmNdefaultActionCallback, sel_callback, arr);

str = XmStringCreateSimple ("FILE INFORMATION");

/*
 * Label widget "FILE INFORMATION" in
 * form2 (form) widget.
 */
label = XtVaCreateManagedWidget ("label", xmLabelWidgetClass, form2,
                                XmNlabelString,      str,
                                XmNleftAttachment,   XmATTACH_POSITION,
                                XmNleftPosition,     6,
                                XmNtopAttachment,    XmATTACH_FORM,
                                NULL);

XmStringFree (str);

n = 0;

/*
 * The scrolled text widget has the following
 * resources set. The list cannot be edited,
 * has a vertical scrollbar, and doesn't provide
 * a horizontal scrollbar.
 */
XtSetArg (args[n], XmNscrollVertical,      True); n++;
XtSetArg (args[n], XmNscrollHorizontal,    False); n++;
XtSetArg (args[n], XmNeditMode,            XmMULTI_LINE_EDIT); n++;
XtSetArg (args[n], XmNeditable,            False); n++;
XtSetArg (args[n], XmNcursorPositionVisible, False); n++;
XtSetArg (args[n], XmNwordWrap,            True); n++;
XtSetArg (args[n], XmNtopAttachment,       XmATTACH_WIDGET); n++;
XtSetArg (args[n], XmNtopWidget,           label); n++;
XtSetArg (args[n], XmNbottomAttachment,    XmATTACH_FORM); n++;
XtSetArg (args[n], XmNleftAttachment,      XmATTACH_POSITION); n++;
XtSetArg (args[n], XmNleftPosition,        6); n++;
XtSetArg (args[n], XmNleftWidget,         s_list); n++;
XtSetArg (args[n], XmNrightAttachment,     XmATTACH_FORM); n++;
XtSetArg (args[n], XmNrightOffset,         7); n++;

/*
 * Scrolled text widget where the file information
 * is displayed. Child of a form widget(form2).
 */
s_text = XmCreateScrolledText (form2, "s_text", args, n);
arr[2] = s_text;

```

```

/*
 * Frame widget to form a border around a text field
 * widget which is created below. This is used only
 * for good visual appearance. Child of pane widget.
 */
frame = XtVaCreateManagedWidget ("frame", xmFrameWidgetClass, pane,
                                XmNshadowType,          XmSHADOW_ETCHED_OUT,
                                NULL);

/*
 * Text field widget, which displays the name of
 * the file being checked out. When no file is
 * selected for check out, it displays "FILE SELECTED: None".
 */
text_output = XtVaCreateManagedWidget ("text_output",
                                       xmTextFieldWidgetClass, frame,
                                       XmNvalue,          "FILE SELECTED : None",
                                       XmNeditable,      False,
                                       XmNcursorPositionVisible, False,
                                       XmNshadowThickness, 0,
                                       NULL);

arr[3] = text_output;

/*
 * The height of the text field widget is stored in 'h'.
 */
XtVaGetValues (text_output, XmNheight, &h, NULL);

/*
 * The panes (which contains the frame widget, and the
 * frame widget is the parent of text_output widget) height
 * is made constant. It remains the same even if the whole
 * dialog window is resized.
 */
XtVaSetValues (frame, XmNpaneMaximum, h+8, XmNpaneMinimum, h+8, NULL);

/*
 * Form widget (form3) is created as a child of pane
 * widget. It will contain the action area buttons
 * 'check out', 'search', and 'cancel' buttons.
 */
form3 = XtVaCreateWidget ("form3", xmFormWidgetClass, pane,
                         XmNfractionBase, 16,
                         NULL);

/*
 * A pixmap is created inside the form3 widget using
 * the window foreground and background colours and
 * "horizontal" pixmap is internal to motif.
 */
XtVaGetValues(form3, XmNforeground, &fg, XmNbackground, &bg, NULL);
pixmap = XmGetPixmap(XtScreen(form3), "horizontal", fg, bg);
XtVaSetValues (form3, XmNbackgroundPixmap, pixmap, NULL);

i = 1;
for ( n=1; n<=3; n++ )
{
    if ( n == 1 )
        str = XmStringCreateSimple (" CHECK OUT ");
    else if ( n == 2 )
        str = XmStringCreateSimple (" SEARCH ");
    else
        str = XmStringCreateSimple (" CANCEL ");
}

```

VAICUWID JIOTE LINDI

```

/*
 * Each action area button is created as a child of
 * form widget (form3).
 */
buttons = XtVaCreateManagedWidget ("buttons", xmPushButtonWidgetClass, form3,
                                   XmNlabelString,      str,
                                   XmNsensitive,        n == 1?False : True,
                                   XmNshowAsDefault,    n == 2?True : False,
                                   XmNdefaultButtonShadowThickness, 1,
                                   XmNleftAttachment,   XmATTACH_POSITION,
                                   XmNleftPosition,     i,
                                   XmNrightAttachment,  XmATTACH_POSITION,
                                   XmNrightPosition,    i+4,
                                   XmNtopAttachment,     XmATTACH_FORM,
                                   XmNbottomAttachment, XmATTACH_FORM,
                                   NULL);

XmStringFree (str);
i += 5;

if ( n == 2 )

    /*
     * Callback for the search button.
     */
    XtAddCallback (buttons, XmNactivateCallback, a_search, arr);
else if ( n == 1 )
{
    arr[4] = buttons;

    /*
     * Callback for the check out button.
     */
    XtAddCallback (buttons, XmNactivateCallback, a_checkout, arr);
}
else if ( n == 3 )

    /*
     * Callback for the cancel button.
     */
    XtAddCallback (buttons, XmNactivateCallback, annul, NULL);
}

XtVaGetValues (buttons, XmNheight, &h, NULL);

/*
 * The action area height is made constant.
 */
XtVaSetValues (form3, XmNpaneMaximum, h+9, XmNpaneMinimum, h+9, NULL);
}

XtManageChild (s_text);
XtManageChild (s_list);
XtManageChild (option);
XtManageChild (form1);
XtManageChild (form2);
XtManageChild (form3);
XtManageChild (form);
XtManageChild (pane);

XtPopup (top, XtGrabNone);
}

```

```
/*
 * This function stores the selection (no of errors) as the value
 * of XmUserData of PanedWindow.
 */
void
option_cb(menu_item, client_data, call_data)
Widget menu_item;
XtPointer client_data;
XtPointer call_data;
{
    int item_no;

    item_no = (int) client_data;
    XtVaSetValues (XtParent(XtParent(XtParent(XtParent(menu_item))))),
                  XmUserData, item_no,
                  NULL);
}

/*
 * This function pops down the dialog when the cancel button
 * is clicked. The dialog is not destroyed. It is popped up
 * when the pattern search is selected in the libout process.
 */
void
annul(menu_item, client_data, call_data)
Widget menu_item;
XtPointer client_data;
XtPointer call_data;
{
    Widget shell;

    shell = XtParent (XtParent (XtParent (menu_item)));

    XtPopdown (shell);
}
```

LIBRARY OF THE UNIVERSITY OF TORONTO

```

/*
    * * * * *
    *   Filename      : app_sear.c   *
    * * * * *
    *   Programmed by : Dontu Sitaram *
    * * * * *
    *   Last updated on : Oct 30 1996 *
    * * * * *
*/

This file contains the code to find the files whose description
has the pattern given by the user in the Libout process.
*/

#include <Xm/Text.h>
#include <Xm/List.h>
#include <Xm/MessageB.h>
#include <X11/cursorfont.h> /* For changing the cursor shape */

#include <stdio.h> /* For popen() */
#include <time.h>
#include <string.h>
#include <sys/types.h>
#include "sem.h" /* Library containing semaphore system calls */
#include "shared_mem.h" /* Library containing shared mem. sys. calls */

#define SEMKEY_VAL 50091 /* Semaphore key (e.g., last 5 digits of SSN) */
#define SHMEM_KEY1 17346 /* Shared memory key */
#define SHMEM_KEY2 44099 /* Shared memory key */

int shmid1; /* Id of shared memory segment1 */

int shmid2; /* Id of shared memory segment2 */

int shared_created = 0;

char *shared; /* Shared memory for holding files containing the pattern */
int *found; /* Shared integer */

int semid; /* Semaphore id */

int no_of_processes = 0;

extern
struct file_node {
    char name[20];
    char rcsfile[80];
    char rcsno[2000];
    char author[40];
    char email[40];
    char function[100];
    char method[100];
    char implementation[1000];
    float saved;
    int outno;
    int status;
    int filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

typedef struct h_list hits_list;
extern
struct h_list {

```

```

    char file_name[30];
    int count;
    hits_list *next;
};

extern XmStringCharSet charset;
extern struct file_node *head, *point;
extern Widget toplevel; /* toplevel widget */
extern hits_list *approx_hd;
extern hits_list *exact_hd;
extern int sys_adm;

extern errordialog();
extern void hits_count();
extern void checkout();

void match_display();
void match_traverse();
void match_check();
void compare();
void create_sharedmem();
void TimeoutCursor();
void shared_kill();

/*
 * This function removes the semaphore and the shared
 * memory segments created.
 */
void
shared_kill()
{
    /*
     * Remove the semaphore, semid; shared memory, shmid1
     * and shmid2.
     */
    sem_rm(semid);
    shmkill(shmid1);
    shmkill(shmid2);
}

/*
 * This function creates a semaphore and the shared
 * memory segments needed.
 */
void
create_sharedmem()
{
    if ((shmid1 = shminit((key_t)SHMEM_KEY1, sizeof(int))) == -1)
    {
        printf("Shared memory segment initialization failed\n");
        exit(1);
    }

    /*
     * Found is a shared pointer to an integer.
     */
    if ((found = (int *) shmat(shmid1, (char *)0, 0)) == (int *)-1)
    {
        perror("shmat");
        exit(1);
    }

    if ((shmid2 = shminit((key_t)SHMEM_KEY2, 2000*sizeof(char))) == -1)

```

```

{
    printf("Shared memory segment initialization failed\n");
    exit(1);
}

/*
 * Shared is a shared character array of size 2000.
 */
if ((shared = (char *) shmat(shmid2, (char *)0, 0)) == (char *)-1)
{
    perror("shmat");
    exit(1);
}

/*
 * Create the binary semaphore.
 */
semid = sem_create((key_t)SEMKEY_VAL, 1); /* initialising semid = 1 */
if (semid == -1)
{
    printf("Semaphore initialization failed.\n");
    exit(1);
}

/*
 * Initialize shared memory segments.
 */
*found = 0;
strncpy (shared, "", 2000);
}

/*
 * This function is called when the search button in the
 * pattern search dialog window is clicked. It searches
 * all the files in the library and displays them in a
 * scrolled list window. When a file is double clicked
 * it's info. is displayed in a scrolled text window.
 * If any error is encountered or no pattern is typed
 * appropriate messages are given.
 */
void
a_search (w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
    char *text;
    Widget *arr = (Widget *) client_data;
    Widget a_text, s_list, text_output, s_text, ok_button, pane;
    Display *dpy;
    int n = 0, i;
    int status;
    static int firsttime = 0;
    hits_list *hits_ptr;

    time_t time1, time2;

    a_text = arr[0];
    s_list = arr[1];
    s_text = arr[2];
    text_output = arr[3];
    ok_button = arr[4];
    pane = arr[5];

```



```

no_of_processes = 0;

if (firsttime == 0)
{
    firsttime++;

    /*
     * Create the shared memory only the first time
     * approximate search is called.
     */
    create_sharedmem ();
    shared_created = 1;
}
else
{
    /*
     * Initialize the shared memory everytime search
     * is started so that a new set of files matched
     * are stored in this shared segment.
     */
    strncpy (shared, "", 2000);
    *found = 0;
}

/*
 * Initialize or reset the widgets to the following
 * values everytime searching is started.
 */
XmTextSetString (text_output, "FILE SELECTED : None");
XmTextSetString (s_text, NULL);
XmListDeleteAllItems (s_list);
XtSetSensitive (ok_button, False);

/*
 * The following 3 lines uodate the window for the
 * above changes.
 */
dpy = XtDisplay (XtParent (pane));
XFlush (dpy);
XmUpdateDisplay (XtParent (pane));

/*
 * (n-1) is the number of errors
 * allowed in the query.
 */
XtVaGetValues (pane, XmUserData, &n, NULL);
text = XmTextGetString (a_text);

if ( !*text || !text )
{
    errordialog ("No pattern typed");
    return;
}
if ( (int)strlen(text) <= (n-1) )
{
    errordialog ("Size of pattern must be greater than the number of errors");
    return;
}

TimeoutCursor (True, XtParent (pane));

time1 = time(NULL);

```

```

/* Check the tree recursively to see if any file description has the
 * required pattern. Make a linked list of all matching files.
 */
match_traverse (head->left, text, n);

for (i=0; i<no_of_processes; i++)
    wait(&status);

time2 = time(NULL);

printf ("The time taken for match_traverse function = %g\n", difftime (time2, time1));

XtFree (text);
match_display (s_list, n);

TimeoutCursor (False, XtParent (pane));

if (!*found)
    errorDialog ("Pattern not found");
}

/*
 * This function displays all the files retrieved in the scrolled
 * list window.
 */
void
match_display(s_list, n)
Widget s_list;
int n;
{
    XmString listname;
    int i = 1;
    char *p;

    /* Traverse the linked list of file names already chosen,
     * and add one by one to the scrolled list widget.
     */

    XmListDeleteAllItems (s_list);

    p = strtok (shared, " ");
    while (p) {
        if (sys_admin)
            hits_count (p, n);

        listname = XmStringCreateSimple(p);
        XmListAddItemUnselected(s_list, listname, i++);
        XtFree(listname);
        p = strtok (NULL, " ");
    }
}

/* This function is used to check the RCS tree structure
 * recursively to see if any file's description has the
 * required pattern. At each node in the tree, it forks
 * off a process which does the work of searching the given
 * pattern in that node. This expedites the search process.
 */
void
match_traverse(cur, str, n)
struct file_node *cur;
char *str;
int n;

```

```

{
    int childpid;
    if (cur) {

        if ((childpid =fork()) == -1)
        {
            /* Fork failed. */
            perror("fork failed");
            exit(1);
        }

        if (childpid == 0) /* child process */
        {
            compare (cur, str, n);
            exit(1);
        }
        else
            no_of_processes++;

        match_traverse(cur->left, str, n);
        match_traverse(cur->right, str, n);
    }
}

/*
 * This function is called by each forked process
 * to search the query given by the user in the node.
 * It calls agrep to search the given pattern with
 * the given no. of errors in that node. If found,
 * the filename of the node is written in a shared
 * memory using a semaphore. The semaphore is a binary
 * semaphore and allows only one process to write into
 * the shared memory at a time.
 */
void
compare(cur, str, n)
struct file_node *cur;
char *str;
int n;
{
    char cmd[2000];
    char str1[2000];
    char buf[102];
    FILE *ptr;
    int ret_val = 0;

    sprintf (str1, "%s\n%s\n%s\n%s\n%s", cur->function, cur->method,
            cur->implementation, cur->author, cur->email);

    if ( n == 0 )
        sprintf(cmd, "echo \"%s\" | agrep -d '$$' -w '%s'", str1, str);
    else
        sprintf(cmd, "echo \"%s\" | agrep -d '$$' -%d -i '%s'", str1, n-1, str);

    if ((ptr = popen(cmd, "r")) != NULL)
        while (fgets(buf, 100, ptr) != NULL)
            ret_val = 1;

    pclose (ptr);

    if (ret_val)
    {

```

```

sem_wait (semid);

*found = 1;
strcat (shared, cur->name);
strcat (shared, " ");

sem_signal (semid);
}
}

/*
 * This function is called when any filename in the
 * scrolled list window is double clicked or retrun
 * is pressed when that filename is highlighted. It
 * displays that file's info. in a scrolled text
 * window.
 */
void
sel_callback(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
    XmListCallbackStruct *cbs = (XmListCallbackStruct *) call_data;
    char *choice, str[80], file_selected[120];
    Widget *arr = (Widget *) client_data;
    Widget s_text, text_output, ok_button;

    s_text = arr[2];
    text_output = arr[3];
    ok_button = arr[4];

    XmStringGetLtoR (cbs->item, charset, &choice);
    strcpy (str, choice);

    XtFree (choice);
    sprintf (file_selected, "FILE SELECTED : %s", str);
    XmTextSetString (text_output, file_selected);
    XtSetSensitive (ok_button, True);
    match_check (s_text, str);
}

/*
 * This function does the job displaying a files
 * information in a scrolled text window.
 */
void
match_check(s_text, str)
Widget s_text;
char str[80];
{
    char string[2000];

    point = NULL;
    traverse(head->left, str);

    if (point) {
        sprintf(string,
            "FUNCTION: %s\nMETHOD: %s\nIMPLEMENTATION: %s\nAUTHOR: %s\nEMAIL: %s",
            point->function, point->method, point->implementation, point->author,
            point->email);

        XmTextSetString (s_text, string);
    } else {

```

```

        errordialog("FILE NOT FOUND IN LIBRARY .....");
        return;
    }
}

/*
 * This function is called when the check out button
 * is clicked. The selected file is checked out into
 * the directory from which 'vci' is run.
 */
void
a_checkout(w, client_data, call_data)
Widget w;
XtPointer client_data;
XtPointer call_data;
{
    Widget *arr = (Widget *) client_data;
    char *choice, str[120], *file_name, string[80];
    Widget text_output;

    text_output = arr[3];

    choice = XmTextGetString (text_output);
    strcpy (str, choice);

    XtFree (choice);

    strtok (str, ":");
    file_name = strtok (NULL, " \\0");
    checkout (file_name);

    sprintf (string, "%s checked out", file_name);
    errordialog (string);
}

/*
 * The search process takes some time. This function
 * indicates that by changing the cursor to clock
 * shape, i.e., the user is not supposed to do anything
 * until the cursor is turned back to normal.
 */
void
TimeoutCursor(on, top)
Boolean on;
Widget top;
{
    static Cursor cursor1, cursor2;
    XSetWindowAttributes attrs1, attrs2;
    Display *dpy = XtDisplay (top);
    XEvent event;

    if (!cursor1 && !cursor2);
    {
        cursor1 = XCreateFontCursor (dpy, XC_watch);
        cursor2 = XCreateFontCursor (dpy, 58);
    }

    /*
     * If on is true, then turn on watch cursor, otherwise,
     * return the top shell's cursor to normal and toplevel's
     * cursor to its original cursor.
     */
    attrs1.cursor = on ? cursor1 : None;
    attrs2.cursor = on ? cursor1 : cursor2;
}

```

```
/*
 * Change the main application shell's cursor to be the
 * timeout cursor or reset it to normal. If other shells
 * exist in this application, they have to be listed
 * here in order for them to have timeout cursors too.
 */
XChangeWindowAttributes (dpy, XtWindow (top), CWCursor, &attrs1);
XChangeWindowAttributes (dpy, XtWindow (toplevel), CWCursor, &attrs2);
XFlush (dpy);

/*
 * Get rid of all button and keyboard events that occurred during
 * the searching process. The user shouldn't have done anything
 * during this time, so flush for button and keypress events.
 * KeyRelease events are not discarded because accelerators
 * require the corresponding release event before normal input
 * can continue. Note: XCheckMaskEvent removes the events happened
 * from the event queue, i.e., they are lost. If you want to
 * process any particular event later you need to set an
 * application defined flag or variable that notifies the
 * application that it must eventually should deal with that event.
 */
if (on == False)
    while (XCheckMaskEvent (dpy,
        ButtonPressMask | ButtonReleaseMask | ButtonMotionMask
        | PointerMotionMask | KeyPressMask, &event))
    {
        XBell (dpy, 50);
    }
}
```

```

/*
    * * * * *
    *   Filename      : hits.c           *
    * * * * *
    *   Programmed by : Dontu Sitaram   *
    * * * * *
    *   Last updated on : Oct 30 1996  *
    * * * * *
*/

    This program contains all the functions needed to draw the File Hit graph.
*/

#include <stdio.h>
#include <string.h>

void hits_count();
void load_hits();
void hitlist_kill();
void cleanup ();

extern int sys_admin;
extern int shared_created;
extern char lib_d[100];

extern void shared_kill();

typedef struct h_list hits_list;
struct h_list {
    char file_name[30];
    int count;
    hits_list *next;
};

hits_list *approx_hd;
hits_list *exact_hd;

/*
 * This function is called from match_display in app_sear.c.
 * This function makes a linked list of all the files found
 * in the search process. Each file in the list has a counter
 * attached to it which has the no of times the file appeared
 * in all the search process till now. There are 2 linked lists
 * one for exact matching and the other for approximate pattern
 * matching.
 */
void
hits_count (str, n)
char *str;
int n;
{
    hits_list *hits_ptr;
    int file_found = 0;

    if (n != 0) /* n = 1 - 4 */
        hits_ptr = approx_hd;
    else /* n = 0 */
        hits_ptr = exact_hd;

    /* Check if the file is present in the linked list. */
    while (hits_ptr->next)
    {
        if (strcmp (str, hits_ptr->next->file_name) == 0)
        {
            hits_ptr->next->count++;
        }
    }
}

```

```

        file_found = 1;
        break;
    }
    hits_ptr = hits_ptr->next;
}

/*
 * Add the file to the linked list and make its counter 1.
 */
if (!file_found)
{
    hits_ptr->next = (hits_list *) malloc (sizeof (hits_list));
    hits_ptr = hits_ptr->next;
    hits_ptr->next = NULL;

    strcpy (hits_ptr->file_name, str);
    hits_ptr->count = 1;
}
}

/*
 * This function loads the hits.dat file in the library
 * directory. The hits.dat file contains the hits in the
 * previous runs of vci.
 */
void
load_hits()
{
    FILE *fp;
    char temp[50];
    hits_list *hits_ptr;

    approx_hd = (hits_list *) malloc (sizeof (hits_list));
    approx_hd->next = NULL;

    exact_hd = (hits_list *) malloc (sizeof (hits_list));
    exact_hd->next = NULL;

    sprintf (temp, "%shits.dat", lib_d);

    fp = fopen (temp, "r");
    if (!fp)
    {
        printf ("Error in opening file hits.dat\n");
        exit(1);
    }

    /*
     * Load the approximate hits.
     */
    hits_ptr = approx_hd;
    while (fscanf (fp, "%s", temp) != EOF)
    {
        if (strcmp (temp, "-----") != 0)
        {
            hits_ptr->next = (hits_list *) malloc (sizeof (hits_list));
            hits_ptr = hits_ptr->next;
            hits_ptr->next = NULL;

            strcpy (hits_ptr->file_name, temp);
            fscanf (fp, "%d", &hits_ptr->count);
        }
        else
            break;
    }
}

```



```

}

/*
 * Load the exact hits.
 */
hits_ptr = exact_hd;
while (fscanf (fp, "%s", temp) != EOF)
{
    hits_ptr->next = (hits_list *) malloc (sizeof (hits_list));
    hits_ptr = hits_ptr->next;
    hits_ptr->next = NULL;

    strcpy (hits_ptr->file_name, temp);
    fscanf (fp, "%d", &hits_ptr->count);
}
fclose (fp);
}

/*
 * This function deletes the linked lists allocated to
 * exact matching and approximate matching at the end of
 * execution of vci.
 */
void
hitlist_kill()
{
    FILE *fp;
    char temp[50];
    hits_list *hits_ptr;

    hits_ptr = approx_hd->next;

    sprintf (temp, "%shits.dat", lib_d);

    fp = fopen (temp, "w");

    /*
     * Write the hits in the file hits.dat in the lib directory.
     */
    while (hits_ptr)
    {
        fprintf (fp, "%s %d\n", hits_ptr->file_name, hits_ptr->count);
        free (approx_hd);
        approx_hd = hits_ptr;

        hits_ptr = hits_ptr->next;
    }
    free (approx_hd);

    fprintf (fp, "%s\n", "-----");
    hits_ptr = exact_hd->next;
    while (hits_ptr)
    {
        fprintf (fp, "%s %d\n", hits_ptr->file_name, hits_ptr->count);
        free (exact_hd);
        exact_hd = hits_ptr;

        hits_ptr = hits_ptr->next;
    }
    free (exact_hd);

    fclose (fp);
}

```

```
/*
 * This function calls the necessary cleaning functions
 * to clean up the semaphore and the shared memory.
 */
void cleanup ()
{
    if ( shared_created )
        shared_kill();

    if (sys_adm)
        hitlist_kill();

    exit(0);
}
```

```

/*
    * * * * *
    *   Filename      : graph.c
    *
    *   Programmed by  : Dontu Sitaram
    *
    *   Last updated on : Oct 30 1996
    * * * * *

    This program contains all the blt code necessary to draw
    the File Hit graph using BLT.
*/

#include <stdio.h>
#include "head.h"

void set_coor ();

/*
 * This function contains the BLT code necessary to draw
 * the graphs showing the relationship between approxima-
 * te pattern matching over exact pattern matching.
 */
void
hits_graph(mode)
int mode;
{
    FILE * ofp;
    char temp[80];
    int *X; /* X coordinates */
    float *Y; /* Y coordinates */
    int j;

    sprintf(temp, "%shits.grph", lib_d);
    ofp = fopen(temp, "w");
    if (!ofp) {
        errordialog("Unable to open hits.grph file ..");
        return;
    }

    /* Lock the hits.grph file in the library directory to avoid
     * simultaneous updates.
     */
    if (!lock(ofp)) {
        errordialog("Unable to lock hits.grph file ..");
        return;
    }

    /*
     * BLT code is written in file hits.grph,
     * which is executed once all the necessary
     * data required to draw the graph is written
     * in the file hits.grph.
     */
    fprintf(ofp, "#!/contrib/bin/blt_wish -f\n\n");
    fprintf(ofp, "if [file exists /contrib/library] {\n");
    fprintf(ofp, "    set blt_library /contrib/library\n");
    fprintf(ofp, "option add *Blt_htext.Font *Times-Bold-R*14*\n");
    fprintf(ofp, "option add *Blt_text.Font *Times-Bold-R*12*\n");
    fprintf(ofp, "option add *graph.xTitle \"File Size\"\n");
}

```

```

/*
 * Frequency Hit Graph - shows the number of
 * hits for all files in the repository for
 * approximate and exact pattern matching.
 */
if (mode == 2)
    fprintf(ofp, "option add *graph.yTitle \"No of hits\\n\\n\");
else
/*
 * Hit Ratio graph - shows the ratio of approximate
 * hits over exact hits for all the files.
 */
if (mode == 3)
    fprintf(ofp, "option add *graph.yTitle \"(approx. hits / exact hits) ratio\\n\\n\");

/*
 * Graph title in BLT code.
 */
fprintf(ofp, "option add *graph.title \"File Hit Frequency\\n\\n\");
fprintf(ofp, "option add *Blt_graph.legendFont *Times-**-8*\\n\\n\");
fprintf(ofp, "set visual [wininfo screenvisual .]\\n\");

/*
 * Background colours for the buttons in header
 * are set.
 */
fprintf(ofp, "if { $visual != \"staticgray\" } { \\n\");
fprintf(ofp, "    option add *print.background yellow \\n\");
fprintf(ofp, "    option add *quit.background red \\n\");
fprintf(ofp, "}\\n\\n\");

/*
 * The header text is set and a button for
 * creating a postscript file of the graph
 * is set.
 */
fprintf(ofp, "global graph\\n\");
fprintf(ofp, "set graph .graph\\n\");
fprintf(ofp, "blt_htext .header -text {\\n\");
fprintf(ofp, "To create a postscript file \"hits.ps.\", press the %%%\\n\");
fprintf(ofp, "button $blt_htext(widget).print -text print -command {\\n\");
fprintf(ofp, " .graph postscript hits.ps -pagewidth 6i -pageheight 4i\");
fprintf(ofp, " -landscape false \\n }\\n\\n\");
fprintf(ofp, "$blt_htext(widget) append $blt_htext(widget).print\\n\");
fprintf(ofp, "%%% button.}\\n\\n\");

/*
 * Footer text and a button for quitting is set.
 */
fprintf(ofp, "blt_htext .footer -text {Hit the %%%\\n\");
fprintf(ofp, "button $blt_htext(widget).quit -text quit -command {destroy .}\\n\");
fprintf(ofp, "$blt_htext(widget) append $blt_htext(widget).quit\\n\");
fprintf(ofp, "%%% button when you are done.%%%\");
fprintf(ofp, "$blt_htext(widget) -padx 20\\n\");
fprintf(ofp, "%%%\");

/*
 * Count the number of files in the repository.
 */
tcount = 0;
nofiles (head->left);

```

```

/*
 * X and Y coordinates of the BLT graph.
 */
X = (int *) malloc ((tcount+1)*sizeof(int));
Y = (float *) malloc ((tcount+1)*sizeof(float));

/*
 * This function calculates the X and Y coordinates
 * and puts them in X and Y arrays created dynamically.
 */
set_coor (X, Y, 0, mode);

/*
 * All the coordinates are written down
 * in the BLT file hits.grph.
 */
fprintf(ofp, "set X1 {\n");
for (j = 0; j < tcount; j++)
    fprintf(ofp, " %d", X[j]);
fprintf(ofp, "\n\n set Y1 {\n");
for (j = 0; j < tcount; j++)
{
    if (mode == 2)
        fprintf(ofp, " %.0f\n", Y[j]);
    else
        if (mode == 3)
            fprintf(ofp, " %.2f\n", Y[j]);
}
fprintf(ofp, "\n\n");

/*
 * BLT code necessary to draw the graph
 * for the obtained coordinates.
 */
fprintf(ofp, "blt_graph $graph\n\n");
if (mode == 3)
{
    /*
     * The Hit ratio graph has a thin line connecting
     * all the points in the graph. The points are
     * represented by circles on the line.
     */
    fprintf(ofp, "$graph element create Ratio -xdata $X1 -ydata $Y1 \\n");
    fprintf(ofp, " -symbol circle -linewidth 1\n");
}
else
if (mode == 2)
{
    /*
     * The Frequency hit graph has a thin line connecting
     * all the points in the graph for approximate pattern
     * matching. The points are represented by circles on
     * the line.
     */
    fprintf(ofp, "$graph element create approx -xdata $X1 -ydata $Y1 \\n");
    fprintf(ofp, " -symbol circle -linewidth 1\n");

    /*
     * Get the X and Y coordinates for all the
     * points on the Frequency Hit graph.
     */
    set_coor (X, Y, 1, mode);

    fprintf(ofp, "set X1 {\n");

```

```

for (j = 0; j < tcount; j++)
    fprintf(ofp, " %d", X[j]);
fprintf(ofp, "\n\n set Y1 {\n");
for (j = 0; j < tcount; j++)
    fprintf(ofp, " %d\n", Y[j]);
fprintf(ofp, "\n\n\n");

/*
 * The Frequency hit graph has a thin line connecting
 * all the points in the graph for exact pattern
 * matching. The points are represented by squares on
 * the line.
 */
fprintf(ofp, "$graph element create exact -xdata $X1 -ydata $Y1 \\n");
fprintf(ofp, "    -symbol square -linewidth 1\n");
}

/*
 * Free the memory after X and Y dynamic arrays
 * are no longer needed.
 */
free (X);
free (Y);

fprintf (ofp, "set coor 0\n");
fprintf (ofp, "label .l -textvariable coor\n");

fprintf(ofp, "pack append . \\n .header { padx 20 pady 10 } \\n");
fprintf(ofp, "    .graph { fill expand } \\n");
fprintf (ofp, " .l { padx 20 pady 20 } \\n .footer { padx 20 pady 10 } \\n\n ");

fprintf(ofp, "wm min . 0 0\n\n");

fprintf(ofp, "bind $graph <B1-ButtonRelease> { %%W crosshairs toggle }\n\n");

fprintf(ofp, "proc TurnOnHairs { } {\n");
fprintf(ofp, "bind .graph <Any-Motion> {\n");
fprintf(ofp, ".graph crosshairs configure -position @%%x,%%y\n");
fprintf(ofp, "set coor [.graph invtransform %%x %%y]\n\n\n");

fprintf(ofp, "bind .graph <Enter> { TurnOnHairs }\n");

memset(temp, '\0', 80);
sprintf(temp, "chmod 777 %shits.grph", lib_d);
system(temp);

memset(temp, '\0', 80);

/*
 * Execute the hits.grph graph at the shell to show the graph.
 */
sprintf(temp, "%shits.grph &", lib_d);
system(temp);

unlock(ofp);

fclose(ofp);
}

```

```

/*
 * This function gets the X and Y coordinates of the number of
 * hits for all files in the repository from the linked
 * lists, necessary to draw the Frequency Hit graph or
 * Hit Ratio graph.
 */
void
set_coor(X, Y, n, mode)
int *X;
float *Y;
int n;
int mode;
{
    /*
     * hits_ptr is used to traverse the linked list,
     * which contains the number of hits for the files
     * which came up as a result of pattern searches.
     */
    hits_list *hits_ptr;

    /* Needed for drawing ratio graph. */
    hits_list *approx_ptr, *exact_ptr;

    struct list *list_ptr;
    int file_found = 0;
    int x1, y1;
    int numerator;
    int denominator;

    x1 = 0; y1 = 0;
    list_ptr = listhead->right;

    /*
     * If mode=2, then the X and Y coordinates for
     * the number of hits for all files should be
     * calculated to draw the Frequency Hit Graph.
     */
    if (mode == 2)
    {
        if (n == 0)
            hits_ptr = approx_hd->next;
        else
            hits_ptr = exact_hd->next;

        /*
         * List_ptr is a linked list containing information
         * about all the files in the repository. For both
         * approximate and exact pattern matchings, the # of
         * hits for all the files are obtained and put in the
         * X and Y arrays.
         */
        while (list_ptr)
        {
            X[x1++] = list_ptr->filesize;

            /*
             * For files with (hits > 0), calculate the
             * Y coordinate.
             */
            while (hits_ptr)
            {
                if (strcmp (hits_ptr->file_name, list_ptr->name) == 0)
                {

```

```

        file_found = 1;
        Y[y1++] = hits_ptr->count;
        break;
    }
    hits_ptr = hits_ptr->next;
}

/*
 * For files with (hits = 0),
 * Y coordinate is 0 (zero).
 */
if (file_found == 0)
    Y[y1++] = 0;

if (n == 0)
    hits_ptr = approx_hd->next;
else
    hits_ptr = exact_hd->next;

list_ptr = list_ptr->right;
file_found = 0;
}
}

else
/*
 * If mode=3, then the X and Y coordinates for
 * the number of hits for all files should be
 * calculated to draw the Hit Ratio Graph.
 */
if (mode == 3)
{
    exact_ptr = exact_hd->next;
    approx_ptr = approx_hd->next;

    while (list_ptr)
    {
        X[x1++] = list_ptr->filesize;

        /*
         * Number of hits using approximate matching.
         */
        while (approx_ptr)
        {
            if (strcmp (approx_ptr->file_name, list_ptr->name) == 0)
            {
                numerator = approx_ptr->count;
                break;
            }
            approx_ptr = approx_ptr->next;
        }

        /*
         * Number of hits using exact matching.
         */
        while (exact_ptr)
        {
            if (strcmp (exact_ptr->file_name, list_ptr->name) == 0)
            {
                denominator = exact_ptr->count;
                break;
            }
            exact_ptr = exact_ptr->next;
        }
    }
}
}

```



```
/*
 * Calculate the (approximate/exact) hits ratio.
 * It should be ensured that the denominator
 * should be atleast one to avoid 'division
 * by zero'. A pattern suchs as '#' brings up
 * all the files during a search, and ensures
 * that all the files have >= zero hits.
 */
Y[y1++] = (float)numerator/denominator;

exact_ptr = exact_hd->next;
approx_ptr = approx_hd->next;

list_ptr = list_ptr->right;
}
}
```

```

/*
    * * * * *
    *   Filename       : head.h           *
    *   * * * * *
    *   Programmed by  : Dontu Sitaram    *
    *   * * * * *
    *   Last updated on : Oct 30 1996    *
    * * * * *
*/
    This file consists of all the declarations needed by the file graph.c.
*/

extern char lib_d[100];
extern lock();
extern unlock();

/*
 * The following structure contains information
 * about an RCS file in the repository.
 */
extern struct file_node {
    char name[20];
    char rcsfile[80];
    char rceno[2000];
    char author[40];
    char email[40];
    char function[100];
    char method[100];
    char implementation[1000];
    float saved;
    int outno;
    int status;
    int filesize;
    struct file_node *left;
    struct file_node *right;
    struct file_node *parent;
};

extern struct file_node *head;

extern int tcount; /* tcount is the total # of files in the repository */
extern void nofiles(); /* This function calculates the total # files */

/*
 * This structure contains information about
 * the distance, the name of file to be attached
 * to, etc., which is given by the user in the
 * LINK process.
 */
extern
struct list{
    char name[20];
    char filename[20];
    int type;
    int distance;
    int filesize;
    struct list * left;
    struct list * right;
};

extern struct list *listhead;

```

```
/*
 * The following structure is made use
 * of in noting down the # of hits for
 * all files in the repository.
 */
typedef struct h_list hits_list;
extern
struct h_list {
    char file_name[30];
    int count;
    hits_list *next;
};

extern hits_list *approx_hd;
extern hits_list *exact_hd;
```

VITA

Sitaram Dontu

Candidate for the Degree of

Master of Science

Thesis: EFFICIENT RETRIEVAL OF SOFTWARE COMPONENTS FROM A
REPOSITORY

Major field: Computer Science

Biographical:

Personal Data: Born in Cement Nagar, Andhra Pradesh, India, July 3, 1972, son of Venkatapathy Dontu and Venkata Lakshmi Dontu.

Education: Graduated high school from Sarada Junior College, Vijayawada, India in May 1990; received Bachelor of Technology in Electronics and Communication Engineering from Regional Engineering College, Kakatiya University, Warangal, India in May 1994; completed the requirements for the Master of Science degree in Computer Science at the Computer Science Department at Oklahoma State University in December 1996.

Experience: Employed by Oklahoma State University, Computing and Information Services, as a Lab Consultant from January 1995 to October 1996.