Relevance Feedback in the Retrieval of Reusable Software Components

M.Sc. Thesis

Lan Jin

Supervisor: Prof. Anestis A. Toptsis

York University
Department of Computer Science
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-22831-2
Relevance Feedback in the Retrieval of Reusable Software Components

by

Lan Jin

a thesis submitted to the Faculty of Graduate Studies of York University in partial fulfillment of the requirements for the degree of

Master of Science

© November 1996

Permission has been granted to the LIBRARY OF YORK UNIVERSITY to lend or sell copies of this thesis, to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film, and to UNIVERSITY MICROFILMS to publish an abstract of this thesis. The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author’s written permission.
Relevance Feedback in the Retrieval of Reusable Software Components

Abstract

In this thesis we focus on the issue of retrieving software from a collection of reusable software components. The proposed method is a variation of the relevance feedback process. The relevance feedback process is a popular retrieval mechanism typically used for test document environments. Here we describe how this method can be adapted for reusable software collections. Two methods of the relevance feedback process are presented. Also, experimental results of the methods' performance evaluation on two different software collections are presented and analyzed.

To the best of our knowledge, reusable software component retrieval is an area of study which has drawn little (or no) attention from the software engineering and information systems community. Most research reports on the technical aspects of software reusability deal with domain analysis, classification of software components, interoperability of software repositories, adaptation of software components for reuse, reuse of systems designs and architectures, and software metrics that quantify eligibility for reuse. All implicitly assume that the retrieval of software components is somehow taken care automatically, or at best, it is done using some ad-hoc retrieval mechanism. The latter approach is typically quite ineffective since changes in the querying habits
and/or modifications of the underlying software collection require major manual efforts for reorganization of the collection. Our proposal constitutes an attempt to put the software retrieval problem on the map of technical obstacles for software reuse. The proposed methods require no prior organization of the software collection. Instead, the collection is organized dynamically in response to the user's querying activity, until satisfactory values of recall and precision are achieved. With the exception of the phase in which the user has to relate her satisfaction of the retrieved results to the system, this organization is performed automatically and it is transparent to the user.

**Keywords**

software reuse, information retrieval, relevance feedback, threshold, degree of relevancy.
Acknowledgment

I would like to thank my thesis supervisor Prof. Anestis Toptsis who first introduced to me the area of software reuse, for his ideas, his insight, his guidance, his patience, and for the financial support he provided me for the duration of this thesis. He also read many drafts of the thesis, and always provided valuable suggestions that greatly improved the content and presentation of this work. I would also like to thank Prof. Jia Xu who provides constructive feedback that has improved the presentation of the thesis. My thanks also go to Prof. Machael Jenkin who has provided advice throughout my study at York, and Ms. Patricia Plumber who has given me constant help and support. I would also like to thank my husband and my parents, without their hardworking and contributions, this thesis would be impossible.
## Contents

**Acknowledgement**  
vi

1. **Introduction**  
1

2. **Related Work**  
4
   2.1 The Formal Specification Approach  
   2.2 The Thesaurus Approach  
   2.3 The Clustered Data Approach  
   2.3.1 The Top-down Approach  
   2.3.2 The Bottom-up Approach  
   2.3.3 The Nearest Neighbor Clusters Approach  
   2.4 Artificially synthesized query method  

3. **Methodology**  
9
   3.1 Background  
   3.2 Prepresentation of software components  
   3.3 The vector method  
   3.4 The structure method  

4. **Evaluation**  
36
   4.1 Test Cases  

vii
4.2 Evaluation Methodology .................................................. 36

4.3 The Structure Method ..................................................... 45
   4.3.1 Quality of retrieval ............................................... 45
   4.3.2 General Performance ............................................. 63

4.4 Vector Method .............................................................. 74

4.5 Comparison of the two methods ..................................... 96
   4.5.1 Comparison of the results for two methods ................. 96
   4.5.2 Conclusion of the comparison .................................. 111

5. Future Work Directions .................................................. 112

Appendix A: Definitions and Notations .................................. 113

Appendix B: Proof of the theorem ...................................... 115

References ................................................................. 118
List of Tables

1 Recall Improvement at the end of 5th iteration for different combinations of DR and T. (C++) .............................................................. 58

2 The average recall increase for the same relevance feedback session ..................60

3 Average decrease of recall when T=0.02-0.03 and 0.08-0.09 .......................... 71

4 Average recall and precision of DR values for different collections and query sets . 72

5 Average recall and precision of the two methods ........................................... 96

6 Recall and precision for both methods with different combinations of T and DR ... 108
List of Figures:

A-1  Recall for structure method for each iteration, (T=0.01,DR=5%,Q2) ......... 45
A-2  Precision for structure method for each iteration, (T=0.01,DR=5%,Q2) ......... 46
A-3  Recall for structure method for each iteration, (T=0.01,DR=5%,Q1) ......... 46
A-4  Precision for structure method for each iteration, (T=0.01,DR=5%,Q1) ......... 47
A-5  Recall for structure method for each iteration, (T=0.10,DR=100%,Q2) ......... 50
A-6  Precision for structure method for each iteration, (T=0.10,DR=100%,Q2) ......... 51
A-7  Recall for structure method for each iteration, (T=0.10,DR=100%,Q1) ......... 51
A-8  Precision for structure method for each iteration, (T=0.10,DR=100%,Q1) ......... 52
A-9  Recall for structure method for each iteration, (T=0.06,DR=50%,Q2) ............. 53
A-10 Recall for structure method for each iteration, (T=0.07,DR=50%,Q2) ............. 54
A-11 Recall for structure method for each iteration, (T=0.09,DR=50%,Q2) ............. 54
A-12 Precision for structure method for each iteration, (T=0.08,DR=50%,Q2) ............. 55
A-13 Recall for structure method for each iteration, (T=0.01,DR=100%,Q2) ............. 55
A-14 Precision for structure method for each iteration, (T=0.01,DR=100%,Q2) ............. 56
A-15 Recall for structure method for each iteration, (T=0.01,DR=100%,Q1) ............. 56
A-16 Precision for structure method for each iteration, (T=0.01,DR=100%,Q1) ............. 57
A-17 Precision for structure method for each iteration, (T=0.10,DR=5%,Q1) ............. 59
A-18 Precision for structure method for each iteration, (T=0.10,DR=5%,Q2) ........... 59
A-19 Recall for structure method for each iteration, (T=0.06,DR=20%,Q2) ............ 61
A-20 Recall for structure method for each iteration, (T=0.07,DR=5%,C++) ............ 62
A-21 Recall for structure method of iteration avg., (T=0.01,Q1) ....................... 64
A-22 Recall for structure method of iteration avg., (T=0.01,Q2) ....................... 64
A-23 Recall for structure method of iteration avg., (T=0.05,Q1) ....................... 65
A-24 Recall for structure method of iteration avg., (T=0.05,Q2) ....................... 65
A-25 Recall for structure method of iteration avg., (T=0.10,Q1) ....................... 66
A-26 Recall for structure method of iteration avg., (T=0.10,Q2) ....................... 66
A-27 Precision for structure method of iteration avg., (T=0.01,Q1) ................... 67
A-28 Precision for structure method of iteration avg., (T=0.01,Q2) ................... 68
A-29 Precision for structure method of iteration avg., (T=0.05,Q1) ................... 68
A-30 Precision for structure method of iteration avg., (T=0.05,Q2) ................... 69
A-31 Precision for structure method of iteration avg., (T=0.10,Q1) ................... 69
A-32 Precision for structure method of iteration avg., (T=0.10,Q2) ................... 70
B-1 Recall for vector method at end of 10th iteration, App.A. (T=0.01) ............ 74
B-2 Recall for vector method at end of 10th iteration, App.B. (T=0.01) ............ 74
B-3 Precision for vector method at end of 10th iteration, App.A. (T=0.01) ........ 75
B-4 Precision for vector method at end of 10th iteration, App.B. (T=0.01) ........ 76
B-5 Recall for vector method at end of 10th iteration, App.A. (T=0.07) ............ 78
B-6 Recall for vector method at end of 10th iteration, App.B. (T=0.07) ............ 78
B-7 Precision for vector method at end of 10th iteration, App.A. (T=0.07) ............ 79
B-8 Precision for vector method at end of 10th iteration, App.B. (T=0.07) ............ 79
B-9 Recall for vector method at end of 10th iteration, App.A. (DR=20%) ............ 80
B-10 Recall for vector method at end of 10th iteration, App.B. (DR=20%) ............ 81
B-11 Precision for vector method at end of 10th iteration, App.A. (DR=20%) ........ 82
B-12 Precision for vector method at end of 10th iteration, App.B. (DR=20%) ........ 82
B-13 Recall for vector method for each iteration, App.A. (T=0.04, DR=10%) ......... 84
B-14 Recall for vector method for each iteration, App.B. (T=0.04, DR=10%) ......... 84
B-15 Precision for vector method for each iteration, App.A. (T=0.04, DR=10%) .... 86
B-16 Precision for vector method for each iteration, App.B. (T=0.04, DR=10%) .... 86
B-17 Recall for vector method for each iteration, App.A. (T=0.1, DR=10%) .......... 87
B-18 Recall for vector method for each iteration, App.B. (T=0.1, DR=10%) .......... 87
B-19 Precision for vector method for each iteration, App.A. (T=0.1, DR=10%) .... 88
B-20 Precision for vector method for each iteration, App.B. (T=0.1, DR=10%) .... 88
B-21 Precision for vector method for each iteration and T.(DR=10%) ............... 91
B-22 Similarity for vector method for each iteration and T.(DR=10%) ............... 91
B-23 Precision for vector method for each iteration and T.(DR=20%) ............... 92
B-24 Similarity for vector method for each iteration and T.(DR=20%) ............... 93
B-25 Precision for vector method for each iteration and T.(DR=50%) ............... 94
B-26 Similarity for vector method for each iteration and T.(DR=50%) ............... 94
C-1 Recall for both methods of iteration avg. (T=0.01, C++) ......................... 98
C-2 Precision for both methods of iteration avg. (T=0.01,C++) ....................... 98
C-3 Recall for both methods for each iteration. (T=0.01,DR=5%,C++) .............. 99
C-4 Precision for both methods for each iteration. (T=0.01,DR=5%,C++) .......... 99
C-5 Recall for both methods for each iteration. (T=0.01,DR=100%,C++) ........... 100
C-6 Precision for both methods for each iteration. (T=0.01,DR=100%,C++) ........ 100
C-7 Recall for both methods of iteration avg. (T=0.05,C++) .......................... 101
C-8 Precision for both methods of iteration avg. (T=0.05,C++) ....................... 102
C-9 Recall for both methods for each iteration. (T=0.05,DR=50%,C++) ............ 102
C-10 Precision for both methods for each iteration. (T=0.05,DR=50%,C++) ......... 103
C-11 Recall for both methods of iteration avg. (T=0.10,C++) .......................... 104
C-12 Precision for both methods of iteration avg. (T=0.10,C++) ....................... 105
C-13 Recall for both methods for each iteration. (T=0.10,DR=5%,C++) ............. 106
C-14 Precision for both methods for each iteration. (T=0.10,DR=5%,C++) ........... 106
C-15 Recall for both methods for each iteration. (T=0.10,DR=100%,C++) .......... 107
C-16 Precision for both methods for each iteration. (T=0.10,DR=100%,C++) ....... 108
Chapter 1.

Introduction

Effective practice of software reuse (the "art" of reusing existing software to construct new software) can impact multiple aspects of software engineering in the large. During the past five years, reuse has enjoyed an unprecedented attention from both academia and industry. Recent studies estimate that a $30 dividend is paid for $1 invested on software reuse over a four-year period [Jone94]. Beyond its traditional role of using old software to produce new software, benefits of software reuse can be realized in software maintenance, enhancement, re-engineering, and reverse engineering tasks [Topt95]. A recent projection states that software reuse will be totally and naturally integrated in the software construction process by the year 2000 [Diaz94]. At the same time, although there have been a large number of success stories for software reuse (e.g., [NASA91, Tirs91, Griss91, D’Ale93, Sind93]), there are a large number of technical, managerial, and legal issues to be resolved before software reuse becomes effective. The key technical issues are in the areas of domain analysis, classification of software components, interoperability of software repositories, adaptation of software components, reuse of systems designs and architectures, and software metrics that quantify eligibility for reuse [USGA93]. Reportedly, no standards exist for any of these areas.
An issue that is little addressed is the retrieval of software components from a software library. All methods dealing with technical issues of software reuse (and especially those for software classification) implicitly assume that the retrieval of software components is somehow taken care of automatically, or at best, is done using some ad-hoc retrieval mechanism. The latter approach may be quite ineffective since changes in the querying habits and/or modifications of the underlying software collection require major manual efforts for reorganization of the collection. The problem arises from the fact that the organization and maintenance of the software library requires a significant investment in manual effort. This is, for example, the case in the classification schemes reported in [Diaz87a, Maar91, Frak88, Cho90, Arno87, Oste92]. A recently appeared software classification method [Daud94] performs the software library organization automatically, however it still requires significant manual effort for the domain analysis task, prior to organization. In general, we feel that the software retrieval problem has been given little attention, or at best, it has been addressed indirectly as byproduct of research endeavors whose primary focus was the software library organization. In the latter case, the software retrieval task is subject to the difficulties outlined above.

We propose two major methods for software retrieval. Both methods are based on the relevance feedback process of information retrieval. The relevance feedback method is a popular retrieval mechanism typically used for text document environments. The main characteristic of our methods is that during their endeavors for query answering, they
require no prior manual effort for classifying the underlying software collection. Instead, the collection is organized dynamically in response to the users' querying activity, until satisfactory values of recall and precision are achieved. With the exception of the phase in which the user has to relate her satisfaction of the retrieved results to the system, this organization is performed automatically and it is transparent to the user. In our methods, we start with a software collection which is completely unorganized. Then, given a user's query $Q$, the collection is iteratively (but automatically) organized until the user receives a satisfactory set of components as the result of the query. Extensive experimental results on two different software collections will evaluate the performance of the proposed relevance feedback methods. By attaining high recall and precision values, the test results will show that both methods perform well.
Chapter 2.

Related work

Several relevant feedback approaches have been reported in the literature. The methods outlined below provide an overall coverage in terms of the different methodologies and techniques.

2.1 The Formal Specification Approach

Patrick Chen [Chen93] proposed the formal specification approach. A reusable component consists of an abstract algebraic specification and at least one concrete implementation. The abstract formal specifications of components are mapped into database objects. Reusable components are thus represented by objects in the database where they can be retrieved by the knowledge base management system. The goal specification to be implemented is mapped into a query to the database which can be processed by the knowledge base management system. The retrieved components should be adapted before reuse, since the new program systems are in general different from the stored ones. Hence best match of goal specification and reusable components is supported.
The specifications are written by using the algebraic specification language ASL. Based on the semantics of specifications, an implementation relation is defined which satisfies the requirements of the abstract specification. The query rules in the data modeling language Telos are used to formulate queries to the database. The translate-comp maps signatures of reusable components (consists of a set of sorts and a set of operations, each being equipped with a particular functionality) to Telos objects. The goal specifications can be mapped by translate-goal into Telos query classes. From the definition of the mappings translate-comp and translate-goal follows that the transformation of a goal specification yields a query class which retrieves exactly those objects of the class component which represent signatures of reusable components that can be matched with the signature. The formal specification approach can be extended by allowing navigation through a component library according to the vertical and/or horizontal structuring properties of a component.

2.2 The Thesaurus Approach

M.G. Fugini [Fugi93] proposed a software retrieval mechanism that is based on the Functional Description attribute attached to reusable classes, and on the elaboration of the FD to obtain a measure of the similarity between two classes. Two issues are being currently considered to illustrate the approach, the thesaurus facility and the weights.
A disciplined approach to keyword insertion is needed for constructing the thesaurus. The system should be able to "normalize" the user query by substituting synonyms with unique terms. Thesaurus structuring needs basically to organize keywords according to application-domain specific vocabularies. The thesaurus is re-organized into different sets of verbs and nouns pertaining to the different application domains. Thesaurus keywords are the main help for query formulation. Additionally, the system uses them to create the list of candidates to be examined through a comparison algorithm, in order to compute the confidence values between the reusable classes which indicate the percentage of successful matching of the application requirements when one class substitutes the other. Numerical weights are arbitrary and are being substituted by fuzzy values.

2.3 The Clustered Data Approach

This approach has concentrated on the effectiveness of retrieval from hierarchically clustered document collections, based on the cluster hypothesis, which states that associations between documents convey information about the relevance of documents to requests.

2.3.1 The Top-down Approach

A top-down search [Will88] involves entering a preconstructed tree of clusters at the root and matching the query against the cluster at each node, moving down the tree following the path of greater similarity. The search is terminated according to some criterion, for
instance when the cluster size drops below the number of documents desired, or when the query-cluster similarity drops below an acceptable threshold. A single cluster is retrieved when the search is terminated. Since it is difficult to adequately represent the clusters in the very large top-level clusters, a useful modification is to eliminate the top-level clusters by applying a threshold clustering level to the hierarchy to obtain a partition, and using the best of these mid-level clusters as the starting point for the top-down search.

### 2.3.2 Bottom-Up Approach

A bottom-up search [Will88] begins with some software components or cluster at the base of the tree and moves up until the retrieval criterion is satisfied. The beginning software components may be items known to be relevant prior to the search, or they can be obtained by a best match search of documents or lowest-level clusters. Comparative studies suggest that the bottom-up search gives the best results, particularly when the search is limited to the bottom-level clusters. Output may be based on retrieval of a single cluster, or the top-ranking clusters may be retrieved to produce a predetermined number of either components or clusters; in the latter case, the documents retrieved may themselves be ranked against the query.

### 2.3.3 The Nearest Neighbor Clusters Approach

This approach [Will88] retrieves a component and components that are most similar to it. It was determined that for a variety of test collections, search performance comparable to
or better than that obtainable from non-clustered collections could be obtained using this method.

2.4 Artificially synthesized query method

This method [Oomm88] considers the adaptive reorganization of data, which is achieved not by using the user’s query stream but rather by using a synthesized query stream which has more concentrated statistical information about the user’s query than the original query stream. More formally, let \( R=(R_1, R_2, ..., R_n) \) be a set of data elements. The elements of \( R \) are accessed by the users of the system according to a fixed but unknown distribution \( S=(S_1, S_2, ..., S_n) \), referred to as the user’s query distribution. However, rather than organizing the data according to \( Q \), the stream of queries presented by the user is reorganized based on a synthesized query stream \( Q' \). This synthesized stream possesses an underlying distribution \( S' \). Observe that by doing this we effectively modify the user’s query distribution without his knowing it. If this transformation is done appropriately, the data storage achieved according to \( S' \) will be superior to that achieved if the data was stored according to the distribution \( S \).
Chapter 3.

Methodology

3.1 Background

3.1.1 The traditional text document retrieval system

In the traditional text document retrieval system developed by G. Salton and Michael J. McGill [Sal83, Sal89], the documents are retrieved by performing similarity computations between stored items and incoming queries, and by ranking the retrieved items in decreasing order of their similarity with the query. Terms are used to express the concepts included in each document, and each document is characterized by a collection of individual terms. The retrieval system is based on the vector space model, in which each record, or document, is represented in a vector format. Each element in the vector represents the weight, or importance, of term \( j \) assigned to document \( i \). For example, we have two documents:

- \( D_1 = \) information science
- \( D_2 = \) retrieval systems

Query \( Q = \) retrieval of information.

the terms used in the documents and the query are defined as \( t_1 \) "information" and \( t_2 \) "retrieval", \( D_1 \) is represented as a vector \((d_{11}, d_{12})\), \( D_2 \) is represented as vector \((d_{21}, d_{22})\), Where \( d_{ij} \) \( (i=1,2;j=1,2) \) is assumed to represent the weight, or importance, of term \( j \)
assigned to document \( i \). Each element in the vector format represents the importance of term \( t_i \) within each document.

The query \( Q \) can be similarly identified as a vector \((q_1, q_2)\), where \( q_i \) \((i = 1, 2)\) represents the weight, or importance, of term \( i \) assigned to the query.

Upon transforming the query and each document into the vector format, the retrieval of a stored item can be made to depend on the magnitude of a similarity computation measuring the similarity between a particular document vector and a particular query vector. The similarity measure is defined as

\[
\text{Sim}(\text{DOC}, \text{query}) = \sum \sigma \times \sigma'.
\]

where \( \sigma \) and \( \sigma' \) are the vector representation of the document and the query. The retrieval of a document can be made to depend on a particular threshold in the similarity measure.

The advantage of generating similarity between the query and software components is that:

1. The size of the retrieved set can be adapted to the user’s requirements by retrieving only the top few items in the ranked order when casual users are involved, while providing a more exhaustive group of items to specialized users who may require high-recall performance.
2). Items retrieved early in a search, which are most similar to the queries, may help generate improved query formulations using relevance feedback.

One of the most important and difficult operations in information retrieval is generating useful query statements that can extract materials wanted by users and reject the remainder. Since usually an ideal query representation cannot be generated without knowing a great deal about the composition of the collection, it is customary to conduct searches iteratively, first operating with a tentative query formulation, and then improving formulations for subsequent searches based on evaluations of the previously retrieved materials. One method for automatically generating improved query formulations is the well-known relevance-feedback process, in which the relevance assessments supplied by the users for previously retrieved documents are returned to the system and used to construct new query vectors. The reformulated queries can then be compared with the stored documents in a new search operation. The aim is to construct new queries exhibiting a greater degree of similarity with the documents previously identified as relevant by the user than the original queries; at the same time, the new queries are expected to be less similar to the documents identified as nonrelevant by the user than the originals.

The basic process of relevance feedback is carried out as follows. The user constructs the original query and submits it to the system against an existing document collection. The
system responds with a set of documents deemed as relevant to the submitted query. The user examines the documents and expresses her opinion as to which of those are relevant and which are not. The user’s opinion is fed back to the system, which in turn performs the following steps:

1. modifies the current query or the retrieval mechanism.
2. retrieves a new set of (deemed to be) relevant documents, and
3. presents the new set of retrieved documents to the user.

The user examines the newly retrieved set of documents, and the above process is repeated. The process stops when the user is satisfied by the most recently retrieved set of documents. Clearly, the above process is not completely automatic since it requires the user’s evaluation regarding the relevance of the retrieved documents through every iteration.

The modified $Q_{j+1}$ is expressed as $Q_{j+1} = Q_j + \frac{1}{n_1} \sum_{i=1}^{n_1} R_i - \frac{1}{n_2} \sum_{i=1}^{n_2} S_i$, where $Q_j$ is the current (before modification) query, $R_i$ is a relevant document, $S_i$ is a non-relevant document, $n_1$ is the number of most recently retrieved documents that the user marked as relevant, and $n_2$ is the number of most recently retrieved documents that the user marked as non-relevant. $Q_{j+1}$, $Q_j$, $R_i$, and $S_i$ are all “documents”, and are represented as vectors.

3.1.2 The vector method and the structure method
Similar retrieval mechanisms used in the text document retrieval system can be applied to the software component retrieval system. This thesis presents two methods, one is the vector method, the other is the structure method. Both of them are based on the vector space model and combined with the relevance feedback process described above. The process requires that all the software components and the queries be converted into the vector format. When combined with the relevance feedback process, each query is progressively modified based on the user’s feedback upon each retrieval.

In the vector method, we will present two algorithms. The first one (algorithm A) is the direct implementation of the retrieval mechanism used in the document retrieval system. However, as the representation of the software components and the text documents is different, the process of converting the software components into the vector format is different from the process of converting the text documents, in which text documents are identified by “terms”, whereas when converting software components, they are identified by “reference components”. The detailed converting process is described in Section 3.2. Upon converting all the software components into the vector format, the rest part is the same as the document retrieval process. The description of the whole process is presented in Approach A of Section 3.3, along with Algorithm A.

The query modification formula used in the algorithm A has shortcomings when used in retrieving software components, as shown by analysis and test results. Analysis of the
query modification formula in approach A shows that the similarity between the new query constructed and the relevant software components can be further improved by a slightly different modification to the current query. Approach B of section 3.2 presents the extensive analysis, along with the algorithm which is a slightly modification of algorithm A.

While in the vector method, improved query formulation is generated without modifying the vector format of the software components, another approach, however, can also be used to change the vector format of the software components. The way of changing the vector format of both query and software components can further improve the similarity of the query component and the relevant components deemed by the user; at the same time, decrease the similarity of the query component and the nonrelevant components. Section 3.4 describes the structure method along with an algorithm, in which, new vector formats are generated by relocating/merging the features in the reference components.

Extensive experimental comparison of approach A and B for the vector method, as well as the comparison of the vector method with the structure method is provided in chapter 5. Evaluation methodology is provided in chapter 4.
3.2 Representation of software components

When we apply the relevance feedback method outlined in the subsection 3.1 for software components, we have to present to the user a set of components deemed as relevant. In other words, we have to present to the user a list of components which have the similarity with the query greater than or equal to a certain threshold $T$.

The similarity is calculated based on the vector format of both query and the software components. Therefore, we must first convert the software components and the query into the vector format.

In text documents, we have a set of keywords (terms) that capture the entire collection, i.e., each document is expressed as a vector in terms of these keywords (and a weight is associated to each keyword of the particular document, to show the importance of the keyword within that document).

We use the representation technique and structure outlined in [Faus92, Fugi93] to represent our components. Each component is expressed as a functional description (FD). A FD consists of one or more features. A feature is a tuple (verb, noun, weight), where the verb is the action or operation performed, the noun is the object upon which the
operation is performed, and the weight is a number indicating the relative importance of the feature within its FD. In other words, the weight represents the degree of functionality that a particular verb-noun pair contributes to a software component. A sample functional description (FD) with three features is

FD: < open-file H; read-file M; close-file L >.

The letters H, M, L represent weights for High, Medium and Low respectively. The mapping between the letters and weight values is as shown in the following table:

<table>
<thead>
<tr>
<th>Very High(VH)</th>
<th>High(H)</th>
<th>Medium(M)</th>
<th>Low(L)</th>
<th>Very Low(VL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1/2</td>
<td>1/4</td>
<td>1/8</td>
<td>1/16</td>
</tr>
</tbody>
</table>

In case of synonym verbs or nouns, a thesaurus is used to store those synonyms. In order to convert each component into a format that conforms to the vector space model described in subsection 2.1, we have to designate a set of keywords (terms) that describe the entire software collection, and then, express each software component as a vector of these keywords. Also, a weight that represents the importance of each keyword within a particular component vector should be associated with each keyword within that component.

The equivalent of keywords (terms) in our representation is *reference components*. Each of them is a FD consisting of a single feature whose weight is always fixed to M.
(medium). There are as many reference components as the number of distinct features in our collection of FDs. For example, assume our collection consists of components $C_1, C_2, C_3$ as follows:

$$C_1 = (f_1, f_2, f_3, f_4, f_5), \quad C_2 = (f_1, f_2, f_6, f_7, f_8), \quad C_3 = (f_5, f_9, f_{10})$$

Each $f_i$ is a feature consisting of a verb-noun pair and a weight $w$ which may be one of $(VH, H, M, L, VL)$. There are 10 reference components $R_j = (g_j'), j = 1, \ldots, 10$. The feature $g_j'$ is defined to have verb-noun pair identical to the one of features $f_j$ and fixed weight equal to $M$ (medium).

In the software collection, the reference components play the role of keywords (terms). Each software component is then represented as a vector of these keywords, using the mapping $M$ defined as follows. Given $k$ reference components $R_1, R_2, \ldots, R_k$, for each component $C_i = (f_1, \ldots, f_m)$, define

$$M: (f_1, f_2, \ldots, f_m) \rightarrow (R_1, R_2, \ldots, R_k),$$

$$M(f_1, f_2, \ldots, f_m) = (\text{sim}(f_1, f_2, \ldots, f_m), g_1'), \text{sim}(f_1, f_2, \ldots, f_m, g_2'), \ldots, \text{sim}(f_1, f_2, \ldots, f_m, g_9'), \text{sim}(f_1, f_2, \ldots, f_m, g_{10}'))$$

$$= (s_1, s_2, \ldots, s_{10})$$

So component $C_i$ is represented as the vector $(s_1, s_2, \ldots, s_{10})$ since

$$M(f_1, f_2, \ldots, f_5) = (\text{sim}(f_1, f_2, \ldots, f_5), g_1'), \text{sim}(f_1, f_2, \ldots, f_5, g_2'), \ldots, \text{sim}(f_1, f_2, \ldots, f_5, g_{10}'))$$
What differentiates the components is the weight \( w_i \) assigned to each keyword-feature in the vector representation of a component. The weight \( w_i \) of the \( i \)-th coordinate of a component \( C \) is defined as \( w_i = \text{similarity}(C^{\text{original}}, R_j) \), where \( C^{\text{original}} \) is the representation of component \( C \). For example, \( C_i^{\text{original}} = (f_1, f_2, f_3, f_4, f_5) \). The function \( \text{similarity}(C^{\text{original}}, R_j) \) is computed as in [Faus92, Fugi93]. The quantity \( \text{similarity}(C^{\text{original}}, R_j) \) is a number between 0 and 1 and determines how functionally similar the components \( C^{\text{original}} \) and \( R_j \) are. The more similar the components are, the higher the similarity value is. We can represent the vector format of component \( C \) or query \( Q \) as \( C(w_1, w_2, ..., w_k) \) or \( Q(w_1, w_2, ..., w_k) \) after the similarity computation.

The reference components \( (R_1, R_2, ..., R_k) \) can be restructured; i.e., the features in \( R_i \) can be relocated to \( R_j \), \( i, j = 1, ..., k \). As indicated in Section 3.1, the relocating of the feature of the reference components will lead to different vector formats of software components as well as the query. By relocating the features, the new reference component set may contain more or less reference components. Therefore, the mapping of reference components

\[
\text{Mrr}: \text{Set}_k(R_1, R_2, ..., R_k) \rightarrow \text{Set}_m(R'_1, R'_2, ..., R'_m)
\]

cconverts the vector format of the software components from \( (w_1, w_2, ..., w_k) \) to \( (w_1, w_2, ..., w_m) \).
3.3. The vector method

As indicated in Section 3.1, we present two approaches in this section. Approach A and Approach B.

Approach A is the direct implementation of the retrieval mechanism used in the text document retrieval system, which is combined with the relevant feedback process. System converts each software component and the query into the vector format as described in the previous section. It then modifies the query based on the user’s feedback.

By adding $\frac{1}{n} \sum_{i=1}^{n} R_i - \frac{1}{n} \sum_{i=1}^{n} S_i$ to the query vector, the system deliberately increases the similarity between the query and the components user deemed as relevant, decreases the similarity between the query and the components user deemed as non-relevant. After the query modification, most of the components that are similar to the relevant ones deemed by the user will be retrieved by the next retrieval, while at the same time, the components that are similar to the non-relevant ones identified by the user will not be retrieved by the next retrieval. It modifies the query formula based on the feedback from the user again and starts the next retrieval. The retrieving process stops when the user is satisfied with the retrieving result.
The query modification formula used in the algorithm A has shortcomings when used in retrieving software components, as shown by analysis and test results. The aim of query modification is to construct new queries exhibiting a greater degree of similarity with the software components previously identified as relevant by the user than the original one; at the same time, the new queries are expected to be less similar to the software components identified as nonrelevant. Analysis of the query modification formula in approach A shows that the similarity between the new query and the relevant software components can be further improved by a slightly different modification to the current query. In approach B, instead of adding \( \frac{1}{n_1} \sum_{i=1}^{n_1} R_i - \frac{1}{n_2} \sum_{i=1}^{n_2} S_i \) to the query vector, it adopts a different query modification method which is described in approach B. Apart from this modification, the rest remains the same as in approach A.

The standard approach (approach A):

Define \( \sigma \) as the vector format of the software component, \( \sigma' \) as the vector format of the query.

- Let \( N \) software components \( C_1, C_2, ..., C_n \) be in the collection.
- Identify all distinct verb-noun pairs in the collection. Let there be \( k \) such pairs \( (v_i, n_i) \), \( i=1,...,k \)
- Create \( k \) reference components \( R_1, R_2, ..., R_k \) such that \( R_i = (v_i, n_i, M) \), \( i=1,...,k \). (note, all
reference components have fixed weight $M$.)

- Convert all software components $C_i$, $i=1,...,N$ into vector format:

- Associate a weight $s_{ij}$, for each term $f_{ij}$, where $s_{ij} = similarity(C_i, g_j')$, where $g_j'$ is the verb-noun pair of reference component $R_j$ (as defined by mapping $M$ in section 2).

  $i=1,...,N$ and $j=1,...,k$. Then, component $C_i$ can be expressed as $(s_{i1}, s_{i2},..., s_{ik})$.

- Given query $Q=(q_{fl}, q_{f2},..., q_{fk})$, express $Q$ in vector format (similar to the way we did for the components). $Q$ becomes $(s_{1}, s_{2},..., s_{k})$, where $s_j = similarity(Q, R_j)$, $j=1,...,k$.

  /* Query the collection, using relevance feedback */

- Set the threshold $T$;

  $Q_{old} = Q$

  LOOP

  - System retrieves all components $C_j$ such that $\sum_{\sigma} \sigma \geq T$

  - User inspects all retrieved components $C_j$.

    If $C_j$ is relevant, then it is marked as relevant;

    If $C_j$ is non-relevant, then it is marked as non-relevant.

    Exit LOOP if user is satisfied by the latest set of retrieved components;

- User announces Relevant and Non-relevant components to the system.

- System does:

  Update query $Q_{old}$ to $Q_{new}$ using the query modification formula:

  $$Q_{new} = Q_{old} + \frac{1}{m} \sum_{i=1}^{m} R_i - \frac{1}{m} \sum_{i=1}^{m} S_i$$
Here following is an example to illustrate the algorithm.

The verb-noun pairs of the software components are:

$C_1$: Top-stack
  Pop-stack

$C_2$: Top-stack

$Q$: Top-stack

The reference components are initialized as:

$R_1$: Top-stack

$R_2$: Pop-stack

Suppose $C_1$, $C_2$ and $Q$ can be represented in vector format after computing the similarity with each reference component: $C_1 (a_1, b_1)$, $C_2 (a_2, b_2)$ and $Q(q_1, q_2)$.

The similarities between $(C_1, Q)$ and $(C_2, Q)$ all exceed the threshold $T$, therefore both $C_1$ and $C_2$ are retrieved by the first retrieval and presented to the user. User identifies that $C_2$ is relevant to the system while $C_1$ is non-relevant.

The system then modifies the query $Q$ in the following way:

$Q = Q + C_2 - C_1 = (q_1 + a_2 - a_1, q_2 + b_2 - b_1)$. 

$Q_{old} = Q_{new}$

END LOOP
The similarities between each component and query $Q$ are re-computed. The system retrieves the components whose similarity with $Q$ exceeded the threshold $T$, and then lets the user identify the relevant and non-relevant components for the second retrieval.

**Approach B:**

The two main ingredients of approach A are the retrieval criterion $\sum \sigma \times \sigma' \geq T$ and the query modification formula $Q_{new} = Q_{old} + \frac{1}{n} \sum_{i=1}^{n} R_i - \frac{1}{n} \sum_{i=1}^{n} S_i$.

Obviously, for the relevance feedback method to be effective, the query modification formula should produce new queries such that the dot product $\sum \sigma \times \sigma'$ exceeds $T$ for the relevant components, while at the same time it falls below $T$ for the non-relevant components. This leads to the following observation. Suppose some coordinates in the initial query $Q$ are equal to 0 (zero). This means that the query contains a verb-noun pair which does not match any of the verb-noun pairs of the reference components. (Equivalently, since the verb-noun pairs of the reference components cover all verb-noun pairs of the collection, we can conclude that the corresponding verb-noun pair in the query does not match any verb-noun pair appearing anywhere in the collection.) Looking at the query modification formula, we observe that in order to produce the new query $Q_{new}$, the quantity $\frac{1}{n} \sum_{i=1}^{n} R_i - \frac{1}{n} \sum_{i=1}^{n} S_i$ is added to all coordinates of the current query.
\( Q_{old} \) including those coordinates that are equal to zero. In case that \( \frac{1}{m} \cdot \sum_{i=1}^{m} R_i - \frac{1}{n} \cdot \sum_{i=1}^{n} S_i \) is a positive number, then the new query \( Q_{new} \) will appear to have a positive coordinate in the place where the old query \( Q_{old} \) used to have a zero coordinate. This will subsequently increase the value of the dot product \( \Sigma \sigma \times \sigma' \) and as a result, \( \Sigma \sigma \times \sigma' \) may exceed \( T \) for some components and therefore those components would be retrieved. However, it is important to observe that increases in the value of \( \Sigma \sigma \times \sigma' \) which result from an increase of a zero coordinate of \( Q_{old} \) to a positive coordinate in \( Q_{new} \) are not desirable. This is because the zero coordinate in \( Q_{old} \) means that a particular verb-noun pair (the one that corresponds to the zero coordinate of the query) is not desirable in a retrieved component.

On the other hand, if the quantity \( \frac{1}{m} \cdot \sum_{i=1}^{m} R_i - \frac{1}{n} \cdot \sum_{i=1}^{n} S_i \) happens to be negative, then a zero coordinate in query \( Q_{old} \) will become a negative coordinate in the new query \( Q_{new} \), and this will desirably result in the corresponding coordinate contributing in reducing the value of the dot product \( \Sigma \sigma \times \sigma' \). Out of experimental results [Brau68], it is better to modify the query vector by just adding relevant components rather than adding and subtracting relevant and non-relevant components. Therefore, we adopt the following query modification method. For the non-zero coordinates of \( Q_{old} \), the corresponding coordinates of the query \( Q_{new} \) are computed as \( Q_{new} = Q_{old} + \frac{1}{m} \cdot \sum_{i=1}^{m} R_i - \frac{1}{n} \cdot \sum_{i=1}^{n} S_i \). For the zero coordinates of \( Q_{old} \), the corresponding coordinates of the new query \( Q_{new} \) remain zero. Apart from this modification, the reset remains the same as in approach A. Therefore, approach B is identical to approach A except that the query modification in the

24
bottom of the relevance feedback LOOP of approach A is applicable only to non-zero entries of the current query, as described above. Taking the same example given out for approach A, if vector of \( Q \) becomes \( Q(q_1, 0) \), when modifying the query \( Q \) after identifying \( C_2 \) is relevant and \( C_1 \) is non-relevant, 
\[
Q_{\text{new}} = (q_1 + a_2 - a_1, 0).
\]

3.4. Structure Method

As described in Section 3.1 of Chapter 3, by relocating/merging the features in the reference components, we effectively modify the query and component vectors. Perceptually, by modifying the query itself, we move the query towards the relevant components, and away from the non-relevant components identified by the user. While by modifying both query and the user deemed relevant components, we move query and the relevant components towards each other.

In the structure method, we effectively implement the idea of moving the query and the user deemed relevant components towards each other by relocating/merging the features in the reference components, while in the vector method, we simply move the query towards the relevant components by modifying the query itself. As when the features in the reference components are relocated, the vector representation of both query and software components are changed, perceptually causing the vector movement of both
query and the relevant components. Furthermore, when the relevant components approach a common query, these components will resemble each other more closely than before, as a result, they can be retrieved more easily in response to a similar query submitted later. Therefore we can implement the structure method to the multiple query system, and this is another advantage over the vector method.

As described in the previous section, the mapping of reference components

\[ Mrr: Set_k(R_1, R_2, ..., R_k) \rightarrow Set_m(R'_1, R'_2, ..., R'_m) \]

converts the vector format of the software components from \((w_1, w_2, ..., w_k)\) to \((w'_1, w'_2, ..., w'_m)\). It is implemented by merging features in different reference components. When the features to be merged are contained by the query and relevant components deemed by the user, the system increases the similarity of the components that are similar to the deemed relevant ones but have not yet been retrieved. Therefore, these components would be in the retrieval list by the next retrieval, and the effectiveness of the retrieval would be increased. The retrieving process stops when the user is satisfied with the retrieval results.

The method is illustrated through a small example using two reference components.

Let \( C \) be a software component. Let \((R_1, R_2)\) be the reference component set. The reference component \( R_1 \) contains two features: \((f_{11}, f_{12})\), the reference component \( R_2 \)
contains two features: \((f_{21}, f_{22})\). When we relocate feature \(f_{21}\) from \(R_2\) to \(R_1\), we accomplish a mapping that maps the reference component set \(1: (R_1, R_2)\) to set \(2: (R'_1, R'_2)\) as described by section 3.2. Now \(R'_1\) contains feature \((f_{11}, f_{12}, f_{21})\) and \(R'_2\) contains feature \((f_{22})\). As the features in the reference components are relocated, the mapping also causes the vector formats of software components and query to change. Suppose that the vector format of query component under the reference component set \((R_1, R_2)\) is represented as \((a, b)\), where \(a = \text{similarity}(Q, R_1)\), \(b = \text{similarity}(Q, R_2)\). The software component \(C\) under the reference component set \((R_1, R_2)\) is represented as \((c, d)\), where \(c = \text{similarity}(C, R_1)\), \(d = \text{similarity}(C, R_2)\). Let \(k_1 = \text{similarity}(Q, (f_{21}, f_{22}))\), \(k'_1 = \text{similarity}(Q, (f_{22}))\), i.e., the similarity between the query component \(Q\) and the feature \(f_{21}, f_{22}\) of reference component \(R_2\) separately. Let \(k_2 = \text{similarity}(C, (f_{21}, f_{22}))\), \(k'_2 = \text{similarity}(C, (f_{22}))\), i.e., the similarity between \(C\) and the feature \(f_{21}, f_{22}\) of reference component \(R_2\) respectively. The similarities \(a, b, c, d, k_1, k'_1, k_2, k'_2\) are computed as in [Faus92, Fugi93]. We can show that the query component vector \((a, b)\) will be changed to \((a+k_1, b-k_2)\), the software component vector \((c, d)\) will be changed to \((c+k_1, d-k_2)\) under the mapping of the reference component set \((R_1, R_2) \rightarrow (R'_1, R'_2)\) using the following theorem.

**Theorem:** The similarity between the software component and the reference component is the summation of the similarity between the software component and each feature of the reference component, i.e.,
\[ \text{similarity}(C, R^n) = \sum_{j=1}^n \text{similarity}(C, f_j) \]  
(See Appendix B for the proof)

According to the theorem,

\[ a = \text{Similarity}(Q, R_1) \]
\[ b = \text{Similarity}(Q, R_2) = \text{Similarity}(Q, f_{21}) + \text{Similarity}(Q, f_{22}) = k_1 + k_1' \]
\[ c = \text{Similarity}(C, R_1) \]
\[ d = \text{Similarity}(C, R_2) = \text{Similarity}(Q, f_{21}) + \text{Similarity}(Q, f_{22}) = k_2 + k_2' \]

When \( f_{21} \) is moved from \( R_2 \) to \( R_1 \), the reference component set is \( (R'_1, R'_2) \)

\[ \text{Similarity}(Q, R'_1) = \text{Similarity}(Q, R_1) + \text{Similarity}(Q, f_{21}) = a + k_1 \]
\[ \text{Similarity}(C, R'_1) = \text{Similarity}(C, R_1) + \text{Similarity}(C, f_{21}) = c + k_2 \]
\[ \text{Similarity}(Q, R'_2) = \text{Similarity}(Q, R_2) - \text{Similarity}(Q, f_{21}) = b - k_1 \]
\[ \text{Similarity}(C, R'_2) = \text{Similarity}(C, R_2) - \text{Similarity}(C, f_{21}) = d - k_2 \]

Therefore, the vector representation of the query component is changed from \((a, b)\) to \((a + k_1, b - k_1)\), and the vector representation of the software component is changed from \((c, d)\) to \((c + k_2, d - k_2)\).

The mapping from the old reference component set to the new reference component set also causes the similarity between the query \( Q \) and software component \( C \) to change. The mapping \( MN \), defined below, shows how this change is accomplished.

\[ MN: \text{similarity}_\text{OLD} \rightarrow \text{similarity}_\text{NEW} \], defined as
\[ MN(ac + bd) = (a + k_1)(c + k_2) + (b - k_1)(d - k_2) \]  \( (1) \)

Note, the right-hand-side of (1) (i.e., the new similarity) is equal to
\[ ac + ak_2 + ck_1 + k_1k_2 + bd + k_1k_2 - bk_2 - dk_1 , \] which is equal to
\[ ac + bd + 2k_1k_2 + (a-b)k_2 + (c-d)k_1 \]  \( (2) \).

Obviously, when \( c > d \) and \( a > b \), expression (2) is greater than \( ac + bd \), which is the old similarity between \( C \) and \( Q \). Formula (2) can be implemented by moving features from reference component which corresponds to the smaller weight of both query and relevant component to the reference component which corresponds to the larger weight of both query and relevant component. For example, move feature \( f_{2l} \) from \( R_2 \) to \( R_1 \) in the above case, since \( R_2 \) corresponds to \( (b, d) \) and \( R_1 \) corresponds to \( (a, c) \), and \( (b, d) \) is smaller than \( (a, c) \). Therefore, the above mapping produces a new similarity which is greater than the old similarity. This means that the similarity of the query and component is increased.

On the other hand, if we assume that the mapping is the other way around:
\[ M: (R'_1, R'_2) = (R_1, R_2), \]
it will cause the similarity between the query \( Q \) and software component \( C \) to be decreased.

---

1 The Similarity between \( C \) and \( Q \) is computed as the dot product, whereas the other similarities are computed as in [Faus92, Fugi93]
The above discussion gives us the following intuition. The restructuring of reference components will change the similarity between query and software components. There are some ways of increasing it and some other ways of decreasing it. This can be applied to the relevance feedback method. For those components that are identified by the user through similarities between those components and the query, whereas for the non-relevant components we restructure the reference components to decrease those components and the query. However, how to select the features to move and how to select the reference component that the selected features to move to? When query and relevant components are presented in the vector formats, we select the reference component which corresponds to the largest coordinate of query vectors, and that is the reference component that the selected features should move to. For each feature that appears in the reference component set and query as well, we count \( t_{rel} \) as the total number of relevant components that contain this feature, \( t_{nrel} \) as the total number of non-relevant components that contain this feature. We select the features which have \( t_{rel} - t_{nrel} > 0 \). This can guarantee that the verb-noun pairs of the selected features are contained by the majority of relevant components instead of non-relevant components. Therefore, relocating of the features could increase the similarities of most of the relevant components as well as the components that are similar to the relevant components. However, for the features that have \( t_{nrel} - t_{rel} \leq 0 \), we either do not move them or move them back to their previous places, had the features ever been moved.
As the above example shows, we select component $R_i$ as the component that feature $f_{21}$ should be moved to, since $R_i$ corresponds to the largest coordinate of the query vector $(a>b)$. Feature $f_{21}$ is selected to be moved as we assume that the number of relevant components that contain this feature is greater than the number of non-relevant components that contain $f_{21}$.

One advantage of the structure method is that it can restructure the reference components when retrieving software components for multiple queries. By collecting the user’s feedback for multiple queries at each iteration, we can select features of the reference component which are contained by the majority of queries and most of the relevant components of each query. Therefore relocating those features could improve the retrieving performance for the majority of queries, especially when the similarities of query components with the reference components are relatively high.

When multiple queries are presented to the system, i.e., the user is given the retrieval lists for multiple queries at each iteration, the relocation of reference components could result in different vector formats for different queries and software components. Therefore, the relocation has different impact on retrievals for different queries. We could find a way of relocating the reference components that would optimize the retrieval results for the
query stream. In other words, the average recall and precision for the query stream would reach the optimum value.

In the relocation process for multiple queries, we calculate \( n_{rel}[i] \) and \( n_{nrel}[i] \) of each reference component for each query, and then count the total \( tn_{rel} = \sum n_{rel}[i] \) and total \( tn_{nrel} = \sum n_{nrel}[i] \). We choose query, for which \( n_{rel}[j]=\max\{n_{rel}[j]\} \). This is the query that shares the verb-noun pair of the feature with the largest number of relevant components. Presumably, the retrieval performance improvement of this query plays a very important role in the over all retrieval results for the query stream. Therefore the reference component which corresponds to the largest coordinate of the vector format of this query is the component that other features should move to. For each feature of the reference component, when \( tn_{rel} - tn_{nrel} > 0 \), we define the feature as moveable, otherwise it is defined as non-moveable. The features that are identified as movable by the above process are then moved to the identified reference component. For the features that are identified as non-removable by the above process, the positions of the features either do not change or change back to their previous places, had the features ever been moved. Therefore, the reference components are restructured. It is interesting to observe that the larger the value of \( tn_{rel} \) is, the better the retrieval results will be. The large value of \( tn_{rel} \) shows that the verb-noun pair of the feature is shared by many relevant components, so by relocating this feature, the similarities of many components that are similar to the relevant components and contain the verb-noun pair of the feature are
increased. Besides, the large value of $tn_{rel}$ also shows that many queries contain the verb-noun pair of the feature, therefore the larger the value $tn_{rel}$ is, the larger the number of queries that would benefit from the relocation of the feature.

Here is another example for the structure method with multiple queries presented:

Let $C_1$, $C_2$ and $C_3$ be software components in the system,

$Q_1$, $Q_2$ be the queries,

$R_1$, $R_2$ be the reference components.

The verb-noun pairs in each component are as follows,

$C_1$: Push-element

   Top-element

$C_2$: Push-element

$C_3$: Push-element

$R_1$: Push-element

$R_2$: Top-element

$Q_1$: Push-element

   Top-element

$Q_2$: Top-Stack

   Push-element

Suppose $\text{Sim}(C_1, Q_1)$, $\text{Sim}(C_1, Q_2)$, $\text{Sim}(C_2, Q_3)$, $\text{Sim}(C_2, Q_2)$ are greater than threshold $T$, $C_1$ and $C_2$ are retrieved by both queries $Q_1$ and $Q_2$, and $C_1$ and $C_2$ are identified as
relevant for both queries. $C_3$ has a lower weight and $\text{Sim}(C_3, Q_i)$, $\text{Sim}(C_3, Q_j)$ are below the threshold $T$; therefore it cannot be retrieved by both queries. The verb-noun pair $f_i$ in $R_i$ of the reference component push-element is shared by $(C_i, Q_i)$ and $(C_2, Q_i)$ as well as $(C_i, Q_2)$ and $(C_2, Q_2)$. Therefore $\text{rel}_n[1]=2$ and $\text{nrel}_n[1]=0$, $\text{rel}_n[2]=2$ and $\text{nrel}_n[2]=0$. Consequently, $\text{tn}_\text{rel}=4$ and $\text{tn}_\text{nrel}=0$, $\text{tn}_\text{rel}-\text{tn}_n=4 > 0$ and $f_i$ is defined as movable, and we have $\text{rel}[1]-\text{nrel}[1]=\max\{\text{rel}_n[j] - \text{nrel}_n[j]\}$. When we convert $Q_i$ into vector format, it indicates that $f_i$ is the place that other features should be moved to.

The verb-noun pair $f_j$ of the reference component $R_j$ top-element is shared by both $C_i$ and $Q_i$. Therefore $\text{rel}_n[1]=1$ and $\text{nrel}[1]=0$. Consequently, $\text{tn}_\text{rel}-\text{tn}_\text{nrel}=1 > 0$ and $f_j$ is defined as movable, and we have $\text{nrel}[1]-\text{rel}[1]=\max\{\text{rel}_n[j] - \text{nrel}_n[j]\}$. When we convert $Q_j$ into vector format, it indicates that $f_j$ is the place that $f_2$ should be moved to.

When we move $f_2$ to $f_j$, we have the similarities of $C_i$ and $C_2$ with queries increased, as shown by the previous example. At the same time, the similarity of $C_3$ with the queries is also increased, as the verb-noun pair of $f_i$ is also contained by $C_3$. Therefore $C_3$ can be retrieved by queries when the similarities exceed the threshold $T$. The system's retrieval effect is improved.

The restructuring process is summarized below:

**LOOP** on each feature of all the reference components:

- Calculate $\text{tn}_\text{rel} = \sum \text{tn}_\text{rel}[i]$  
- Calculate $\text{tn}_\text{nrel} = \sum \text{tn}_\text{nrel}[i]$

IF($\text{tn}_\text{rel} > \text{tn}_\text{nrel}$) THEN
Select query_j where \( n_{rel}[jj] - n_{nrel}[jj] = \max (n_{rel}[i] - n_{nrel}[i]) \)

Convert query_j into vector format

- Identify the reference component which corresponds to the largest weight of the vector format of query_j
- Move the feature to the reference component

ENDIF
ENDLOOP

The entire relevance feedback and retrieval process can be summarized as follows:

Set system parameters which are the threshold \( T \) and the degree of relevancy \( DR \);

LOOP

- System retrieves all components \( C_j \) such that \( \sum \sigma \cdot \sigma' \geq T \)
- User inspects all retrieved components \( C_j \).
  - If \( C_j \) is relevant, then it is marked as relevant;
  - If \( C_j \) is non-relevant, then it is marked as non-relevant.

Exit LOOP if user is satisfied by the latest set of retrieved components;

- User announces Relevant and Non-relevant components to the system
- System does: restructure the reference components

END LOOP.
Chapter 4.

Evaluation

4.1. Test Cases

The described relevance feedback methods (vector method and structure method) are evaluated using two software collections. The first collection (UNIX collection) is derived from the source code for the commands for the BSD flavor of the UNIX OS. The second collection (C++ collection) is derived from the source code of the C++ compiler. The UNIX collection has 212 FDs, and a total of 720 features, of which 524 contain distinct verb-noun pairs. (i.e., 524 reference components are created). The average number of features per FD in the UNIX collection is 3.40. The C++ collection has 144 FDs, and a total of 1,068 features, of which 641 contain distinct verb-noun pairs (i.e., 641 reference components are created). The average number of features per FD in the C++ collection is 7.42.

4.2. Evaluation Methodology

The performance of the system is evaluated in terms of fluctuation in recall and precision as the relevance feedback process progresses.
Recall is defined as the ratio
\[
\frac{\text{number of retrieved and relevant components}}{\text{total number of relevant components in the collection}} \quad (1)
\]

Precision is defined as the ratio
\[
\frac{\text{number of retrieved and relevant components}}{\text{number of retrieved components}} \quad (2)
\]

The values of recall and precision are between “0” and “1”.

As we explained in Chapter 3, software components are retrieved by computing a similarity for each software component-query pair. A cut is made through the software component collection to distinguish retrieved components from nonretrieved ones, when the \( \text{Sim}(\text{COMP}_n, \text{query}) = \sum \sigma \times \sigma' \geq T \), where \( T \) is defined as the system threshold.

The recall measurement requires knowledge of the total number of relevant components in the collection with respect to the query. Therefore, relevant judgments by the user for all the software components with respect to the query need to be obtained. However, as the number of software components in each collection is relatively large, it is unrealistic to have user identify each component in the collection as relevant or not. For the testing purposes, system evaluates each component automatically by adopting the methodology described below.
The system defines each software component as relevant or not based on the *degree of relevance* between the software component and the query. Degree of relevance is defined as the percentage of verb-noun pairs of software component matching with verb-noun pairs of the query. A system threshold $DR$ is then given to distinguish relevant components from nonrelevant ones. Basically, a component $COMP_i$ is defined as relevant when

$$DR(COMP_i, \text{query}) \geq DR$$

Where $DR(COMP_i, \text{query})$ is the percentage of verb-noun pairs of the software component $COMP_i$ matching with verb-noun pairs of the query and calculated by the system. Furthermore, the system even expands the evaluation for components that are retrieved. Such operation saves tremendous manual effort.

The mechanical process of simulating user's relevant feedback can be summarized in the following steps;

1. User is given a list of components retrieved by the system which have $(COMP_i, \text{query}) = \sum \sigma x \sigma' \geq T$, where $T$ is defined as the system threshold. Count the number of retrieved relevant components and nonrelevant components respectively, as well as the total number of components retrieved.
2. System simulates user's feedback by calculating $DR(RETR_i, \text{query})$, where $RETR_i$ is one of the components retrieved and given to the user. When $DR(RETR_i, \text{query}) \geq DR$, it is identified as relevant, otherwise, as nonrelevant.

3. System calculates the number of relevant components in the collection by computing $DR(COMPi, \text{query})$. If $DR(COMPi, \text{query}) \geq DR$, the counter of number of relevant components increases by 1, otherwise, remains unchanged.

4. The recall and precision are calculated based on the formula (1) and (2) by using the result of 1 and 3.

5. Repeat 1-4 until the retrieval process finishes.

In the user's feedback process, query must be presented in a way that describes the functionality of the software components user required. User is satisfied when the components retrieved perform most of the functionality described and presented by the query. As the verb-noun pairs identify the software components and describe the basic functionality of the software components as well as the query, user would identify each component as relevant or not based on the percentage of verb-noun pairs that match with the query. Therefore, when $DR$, the percentage of verb-noun pairs of the component which match with that of the query exceeds a certain threshold, the system identifies it as relevant component as users would do. Certainly, this mechanism closely resembles the way that users identify each component manually.
The methodology described above has some weakness, and the theory that the methodology is based on may not be absolutely correct. Therefore the test result might be taken with grain of salt. While user might also take into account the weight of each verb-noun pair in the feature of the software component, the simulating mechanism of the system ignores them. However, while user is judging each component in the system, she gives preference on seeing the verb-noun Pairs in the feature that she is looking over the relative importance of the verb-noun within the feature. Only when there are multiple components in the system having the exact same percentage of the v-n pairs matching with the query, user may give preference of some components over others based on weight of the v-n paris in the feature. The simulating mechanism of the system might generate some bias only under the above circumstances, since the system would treat those component equally. However, preliminary study of the collection shows that it is very rare to have multiple components having the exact same percentage of the v-n pairs matching with the query, especially when the average number of v-n paris in the components is relatively large (C++ collection), therefore the above methodology is accurate enough to simulate the user’s feedback (especially for the C++ collection).

Please note, the definition of degree of relevance and threshold DR are generated by the simulating mechanism is only for the purpose of performing the experiment. They are used to obtain the knowledge of the number of relevant components in the collection, in order to calculate recall. They are used to evaluate the retrieval effect of the structure and
vector method, which are based on the vector space model and combined with the relevance feedback process. The criteria of retrieving the software components is the similarity of the components with the query.

The other parameters involved in evaluating the method is the number of iterations performed during a relevance feedback session. The following table shows the parameters that will be used for testing:

<table>
<thead>
<tr>
<th>DR-Degree of relevance</th>
<th>T-threshold</th>
<th>#Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>5%, 10%, 20%, 50%, 100%</td>
<td>0.01, 0.02, 0.03,..., 0.09, 0.10</td>
<td>5-10</td>
</tr>
</tbody>
</table>

To test the vector method, we use the same query for all relevance feedback sessions for each collection. The query used for the UNIX collection consists of 7 features. The query used for the C++ collection consists of 20 features. The queries were constructed by randomly selecting features from the corresponding collection. Note that the query for the C++ collection has size about 3 times the average number of features per FD in that collection, while the query for the UNIX collection has size about 2 times the average number of features per FD in that collection. In this respect, the query for the C++ collection can be considered "big", whereas the query for the UNIX collection can be considered "normal".
To test the structure method with multiple queries presented, we select two query sets for each collection. Each query set contains 20 queries. Query set\textsubscript{1} contains queries with 2 features in each component, query set\textsubscript{2} contains queries with 4 features in each component. The features in the query components are randomly selected.

Test results are analyzed through different sets of graphs. Set A1 represents the recall and precision of both collections for the structure method within the same relevance feedback session when both T and DR and query set are fixed. The X axis represents different iterations within one feedback session. The Y axis represents recall or precision. In the title area, the value of T, DR and the query set number are displayed. The recall or precision values for each iteration shown in the graph are calculated in the following way. After the first retrieval, the recall and precision for each query in the query set are calculated based on the formula provided, the average of recall and precision of the query set are then obtained. This is taken as the recall and precision of the first retrieval. The system then relocates the features in the reference components and recalculates the similarities between each query and the fds in the collection based on the user's feedback. The next retrieval is made for each query and both the recall and the precision are calculated after the retrieval. Then the average recall and precision of the query set is obtained and we plot the dots in the graphs of recall and precision in the positions that corresponds to the next iteration. Test results in terms of recall and precision are
presented in the separate graphs. The above process applies to both Unix and C++ collections and the results for both collections are shown in the same graph.

Set A2 represents the average recall and precision of iterations for different DR values when T is fixed and query set number is given. The X axis represents different DR values and Y axis represents recall or precision. Therefore, the recall and precision for certain T and DR can be taken as the average of recall and precision along the X axis in the corresponding graph (with the same T and DR values) in the Set A1. We plot the values of recall in the positions that correspond to different DR values in the same graph, and results in terms of precision are plotted in a separate graph. Information for both collections are also shown in the same graph.

Set B1 and B2 are represented in a similar way as Set A1 and A2, except the fact that there is only one query set and only a singly query in the query set, therefore the recall and precision in the Set B1 are not averaged over the different queries in the query set.

Query Set C1 represents the test results in terms of recall and precision when comparing the structure method with the vector method. In a similar way, set C1 contains all graphs that represent the recall and precision within one relevance feedback session when T, DR are given. Instead of plotting values of different collections in the same graph, we plot the different values for both methods (vector and structure). It is not necessary to average the
recall and precision for the query set as there is only one query presented. The test results for different collections are presented in different graphs.

Query set C2 contains all graphs that represent the average recall and precision of iterations for different DR values when T is fixed. The X axis represents the different DR values and the Y axis represents recall or precision. Therefore the recall and precision for certain T and DR can be taken as the average of recall and precision along the X axis in the corresponding graph (with the same T and DR values) in the Set C1. Test results for different collections are presented in the separate graphs while test results in terms of different methods are presented in the same graph.
4.3 Structure Method

4.3.1 Quality of retrieval

As the effectiveness of any relevance feedback method is determined by the quality of retrieval (in terms of recall and precision) within the same relevance feedback session, test results are shown when both DR and T are kept fixed and successive iterations are performed. Test results are analyzed through different combinations of T and DR, with T and DR taking high, medium and low values.

Recall for structure method for each iteration, (T=0.01, DR=5%, Q2)

*Figure A-1*
Precision for structure method for each iteration, $(T=0.01, DR=5\%, Q2)$

**Figure A-2**

Recall for structure method for each iteration, $(T=0.01, DR=5\%, Q1)$

**Figure A-3**
Figures A-1, A-2, A-3 and A-4 show that when both $T$ and $DR$ are low, recall gets increased by the second retrieval while precision drops. Note, the increase in recall for the C++ collection is more obvious. However, the recall in third retrieval drops slightly in the C++ collection with query set 2 as shown by figure A-1. This is due to the following.

As both $T$ and $DR$ are low, the number of relevant and retrieved components is relatively large and many of those components can be with low degree of relevancy. In the relocation process as described by the Methodology, the $tn\_rel$ value of many features in the reference components could be larger than $tn\_nrel$, as the verb-noun pair in the feature would be contained by more relevant components than non-relevant ones, and we have $t = n\_rel[i] - n\_nrel[i] > 0$. Therefore many features in the reference components are involved in the relocation process. This would increase the similarity of the components.
that have not yet been retrieved but are similar to the relevant and retrieved components. Therefore, we observe an increase in recall after the second retrieval. However, some of those components are not qualified as relevant as they are similar to the retrieved but marginally relevant components. (Those marginally relevant components are with very low degree of relevancy when DR is low). Therefore the precision decreases after the second retrieval. This potentially increases the value of \( m_{nrel} \) of the features in reference components. According to the algorithm of the relocation process, when the value of \( m_{nrel} \) of these features are greater than that of \( m_{rel} \), those features could be moved back to their previous places in order to decrease the similarity of the components that contain these features. Unfortunately, those components could also include a small number of relevant ones. Therefore, the similarity of those components could be decreased. When the similarities are well below the threshold \( T \), those components cannot be retrieved in the next run, causing decrease in recall in the third retrieval. This phenomenon also shows that, to some degree, the retrieval effect of the current run depends on the quality of retrieval of the previous run. As the number of features that are shared by the components in the C++ collection and query set 2 are the highest among all the combination of collections and query sets, the observed trend is more obvious in C++ collection with query set 2.

Figures A-5, A-6, A-7 and A-8 show the snapshot when both DR and T are high. Figures A-5 and A-7 show the recalls within one relevance feedback session for query set 2 and
query set 1 separately. Figure A-6, A-8 show the corresponding precision. The general observed trend is that improvement of both recall and precision are shown in the graphs. Since both T and DR are high, the relevant and retrieved components after the first retrieval have very high values of degree of relevancy with query. Therefore, relocating the features in the reference components can bring up the similarities of those components that are similar to the relevant and retrieved ones. Some of those components are more likely to be retrieved by the next retrieval and are qualified as relevant when the relevant and retrieved components after the first retrieval have very high values of degree of relevancy. Therefore, the recall and precision for the next retrieval can be increased.

It is worth noticing that with query set 2, the Unix collection has better performance than the C++ collection, which is an exception. When DR =100%, the qualified relevant components in the C++ collection are those FDs which contain 100% the verb-noun pairs in the query FD. Although many verb-noun pairs are shared by the query FD and FDs of the C++ collection, the similarities between the features in the query FD and features in some relevant FDs of the C++ collection could be small, especially when those FDs in the C++ collection contain relatively large amount of features (Fugi, et al), which is normally the case since the number of features per FD in the C++ collection is larger than that of the Unix collection. As a result, some relevant components with DR = 100% in the C++ collection do not have very high similarities with the query. Although moving the features in the reference components could increase the similarities for some
of those components, the retrieving effect in terms of recall is not impressive, as the similarity of those FDs can not to be increased high enough to exceed the high threshold T (0.10). Note, the above exception only happens to a small number of components in C++ collection.

Recall for structure method for each iteration, (T=0.10,DR=100%,Q2)

**Figure A-5**
Precision for structure method for each iteration, \((T=0.10, DR=100 \%, Q2)\)

Figure A-6

Recall for structure method for each iteration, \((T=0.10, DR=100 \%, Q1)\)

Figure A-7
Figures A-9, A-10, A-11 and A-12 show that fluctuation of both recall and precision happens with DR=50% when $T=0.06, 0.07, 0.08, 0.09$, i.e., when DR is taking the mid value. This is due to the following. Recall and precision are increased after the first retrieval. Unfortunately, most of the relevant components after the first retrieval have degree of relevancy around 50%, i.e., they are the marginally relevant components. These components can be retrieved just because some features of those components have very high value of similarity with query, thus making the similarity of those components high enough to be retrieved. The degree of relevancy of those components is not necessarily high. Therefore, there are not many verb-noun pairs in the features of reference components that are shared by relevant components and the query. Although some of the them are shared by more relevant components instead of non-relevant ones,
which can be shown by \( t = n_{\text{rel}} - n_{\text{nrel}} > 0 \), \( t \) would not be large. As moving the features would not have good performance when \( t \) is not large enough, therefore, both recall and precision decrease. (Proper value of threshold should be defined as whether to allow the feature be relocated or not in order to achieve the best performance, "see future research directions").

When the recall and precision decrease after the third retrieval, most of those marginally relevant components are excluded from the retrieval list. We then observe another increase in both recall and precision after the forth retrieval. When the above procedure repeats itself, the recall and precision fluctuate. Again, C++ collection and query set 2 show more obvious trend.

![Recall for structure method for each iteration, (T=0.06,DR=50%,Q2)](image)

**Figure A-9**
Recall for structure method for each iteration, \((T=0.07, DR=50\%, Q2)\)

**Figure A-10**

Recall for structure method for each iteration, \((T=0.09, DR=50\%, Q2)\)

**Figure A-11**
Figure A-12

Figure A-12 shows that there is also a fluctuation in precision when DR=50% and T=0.08 for the reasons similar to the ones explained above.

Figure A-13
Precision for structure method for each iteration, \((T=0.01, DR=100\%, Q2)\)

**Figure A-14**

Recall for structure method for each iteration, \((T=0.01, DR=100\%, Q1)\)

**Figure A-15**
Figures A-13, A-14, A-15 and A-16 show the retrieval status when T is low and DR is high. We do not observe an increase in recall but a slightly increase in precision for the C++ collection with query set 2 as shown by figure A-14. Since T is low and DR is high, it is very easy for the components to be retrieved, especially those components that cannot be qualified as relevant by the high DR value. Therefore, the precision of the first retrieval would be low. On the other hand, as DR is high, the number of relevant components is relatively small, and the relevant components can be easily retrieved by the first retrieval by the relatively low T values. Therefore, the recall turns out high on even the first retrieval. However, as DR is high, most of the retrieved components by the first retrieval have high degree of relevancy with the query. Therefore, moving the features in the reference components would decrease the similarity of the components.
that do not have high degree of relevancy, i.e., non-relevant components, and precision in the next retrieval increases. At the same time, we can observe that recall remains high.

In table 1, we display the recall improvement in percentage which is obtained by the recall value increase from the first retrieval to the end of the 5th iteration within the same relevance feedback session. It shows that the best recall improvement happens when $T$ is high and $DR$ is low. As C++ collection shows more obvious trend, the recall improvement of C++ collection is displayed. As the retrieved components have high similarities, most of their values of degree of relevancy are high enough to pass the DR value and are qualified as relevant. Therefore, the number of relevant components is much larger than that of the non-relevant components. The value of $tn_{rel}$ for each feature of the reference component is higher than that of $tn_{nrel}$ ($t = tn_{rel} - tn_{nrel}$ is very large). This causes more features of reference components to be involved in the relocation process and the recall to be substantially improved.

Table 1 Recall Improvement at the end of 5th iteration for different combinations of $DR$ and $T$: (C++)

<table>
<thead>
<tr>
<th>Query Set 2, C++ Collection</th>
<th>Query Set 1, C++ Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>T/DR</td>
<td>5%</td>
</tr>
<tr>
<td>0.01</td>
<td>8.13%</td>
</tr>
<tr>
<td>0.10</td>
<td>40%</td>
</tr>
</tbody>
</table>
At the same time, we observe that precision drops for query set 2 as shown by figure A-18, which is typical in information retrieval environment when there are substantial increases in recall.

**Figure A-17**

**Figure A-18**
Table 2 shows average recall improvement within the same feedback session for different combinations of collections and query sets.

Table 2. The average recall increase for the same relevance feedback session.

<table>
<thead>
<tr>
<th>ave.Incr.</th>
<th>Unix Collection</th>
<th>C++ Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Set 1.</td>
<td>2.996%</td>
<td>2.64%</td>
</tr>
<tr>
<td>Query Set 2.</td>
<td>3.17%</td>
<td>6.07%</td>
</tr>
</tbody>
</table>

From table 2, we observe that we could achieve the best retrieval increase in the same feedback session with the combination of C++ collection and query set 2. This is also shown by figures A-19 and A-20, in which we can observe a big increase in the C++ collection as compared with the Unix collection (Figure A-19), and in query set 2 as compared with query set 1 (Figure A-20). When there are more features in the FDs in the C++ collection than in that of the Unix collection, the verb-noun pairs in the features of the reference components are shared by more components in both C++ collection and the query set. Therefore tn_rel would obtain larger values for the C++ collection than for the Unix collection. Since query set 2 has 4 features per component, compared with 2 features per component in query set 1, more common verb-noun pairs are shared among
query set 2. This means that the components in query set 2 are more similar to each other. In the relocation process, we first get the feedback information for all the queries in the query set. For each feature in the reference component, we then sum \( m_{rel} \) as the total of \( n_{rel}[i] \) of each query \( i \), as we described in the Methodology. Generally, when more common verb-noun pairs are shared by the queries in the query set, it is more likely that the verb-noun pairs of the feature of the reference component could be shared by more queries in the query set. Therefore, we could obtain larger \( m_{rel} \) value for each feature in the reference component. Again, the larger the value of \( m_{rel} \) is, the larger the number of queries would benefit from the relocation of the feature, the better the overall retrieval performance would be. That explains the best retrieval improvement for the C++ collection with query set 2.

![Recall for structure method for each iteration, (T=0.06,DR=20%,Q2)](image)

**Figure A-19**
Recall for structure method for each iteration, \((T=0.07, DR=5\%, C++)\)

**Figure A-20**
4.3.2 General Performance

Figures A21-A26 show the test results in terms of the average recall and precision when DR varies from 5% to 100% while fixing T to the values of 0.01, 0.05, 0.10 separately. The general observed trend is that the average recall increases with the increase of DR. This can be explained by the fact that with the increase of DR, fewer components are qualified as relevant. Therefore, the denominator in the ratio of recall, the total number of relevant components, decreases, causing the value of recall to increase. The noticeable drop in recall when T increases is expected since as T increases, fewer components produce a dot product similarity that exceeds the larger threshold T and, therefore, fewer components qualify for retrieval. Consequently, as fewer components are retrieved, also fewer relevant components make it into the retrieved components pool.
Recall for structure method of iteration avg., \((T=0.01,Q1)\)

**Figure A-21**

Recall for structure method of iteration avg., \((T=0.01,Q2)\)

**Figure A-22**
Recall for structure method of iteration avg., \( (T=0.05,Q1) \)

**Figure A-23**

Recall for structure method of iteration avg., \( (T=0.05,Q2) \)

**Figure A-24**
Recall for structure method of iteration avg., \((T=0.10,Q1)\)

**Figure A-25**

Recall for structure method of iteration avg., \((T=0.10,Q2)\)

**Figure A-26**

We also can observe from the graphs that the performance in terms of Recall of the C++ collection is better than that of the Unix collection (The exception of small drop in recall...
when DR=100% shown on figure A-26 has been explained before). This is due to the fact that the query contains more common verb-noun pairs with the C++ collection than with the Unix collection, therefore the average similarity between the C++ collection and the query is higher as explained before. More detailed analysis on comparing the two collections will be given out later.

Figures A-27 to A-32 show the values of the average precision when DR varies from 5% to 100% while fixing T to 0.01, 0.05, 0.10 separately.

![Figure A-27](image-url)
Precision for structure method of iteration avg., \((T=0.01,Q2)\)

**Figure A-28**

Precision for structure method of iteration avg., \((T=0.05,Q1)\)

**Figure A-29**
Figure A-30

Figure A-31
The general observed trend is that with the increase of DR, precision drops. As precision is defined as the ratio: number of relevant and retrieved components / total number of retrieved components, the denominator in the formula does not change when T is kept fixed while the numerator decreases when DR increases.

We also observed that the precision of the C++ collection has poorer performance than that of the Unix collection when DR is high. This is also due to the fact that the average similarity between the C++ collection and the query is higher especially when DR is high.
Since the number of features in the components of the Unix collection is smaller than that in the C++ collection, it causes the query component to have fewer common verb-noun pairs with the components in the Unix collection. This translates to low similarity values (computed as dot products) between the query and components. Taking $T$ values from 0.01 to 0.10, we observe that for the Unix collection, the portion of relevant components with small $T$ values is larger than that for the C++ collection, whereas the portion of relevant components with large $T$ values is smaller than that for the C++ collection. The following table 3 shows the interesting phenomenon.

Table 3. Average decrease of recall when $T=0.02-0.03$ and $0.08-0.09$

<table>
<thead>
<tr>
<th>Ave. Decrease.</th>
<th>Unix</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T=0.02-0.03$</td>
<td>0.33</td>
<td>0.15</td>
</tr>
<tr>
<td>$T=0.08-0.09$</td>
<td>0.10</td>
<td>0.42</td>
</tr>
</tbody>
</table>

According to the definition of recall, the recall value when $T=\tau l$ ($\tau l$ is a random number within range of $T$) is the ratio of relevant components with similarities greater than or equal to $\tau l$ to the total number of relevant components. Therefore, when $T$ jumps from $\tau l$ to $\tau 2$ ($\tau 2$ is a random number within range of $T$), the decrease of recall value is the proportion of relevant components with similarities in the range of $\tau l$ to $\tau 2$ to the total of relevant components. The table shows that when $T$ jumps from 0.02 to 0.03, the decrease of recall in the Unix collection (0.33) is greater than that in the C++ collection (0.15);
When T jumps from 0.08 to 0.09, the decrease of recall in the Unix collection (0.10) is fewer than that of the .C++ collection (0.42). This means that the portion of relevant components with similarities in the range of 0.02 to 0.03 of the Unix collection is larger than that of the C++ collection, whereas the portion of relevant components with similarities in the range of 0.08 to 0.09 of the C++ collection is larger than that of the Unix collection. Therefore, with the increase of T, the different decrease of recall for the two collections shows that the average similarity between query and Unix collection is smaller than that between the query and C++ collection. Therefore when DR is high, more components from the C++ collection can be retrieved. As those components include both relevant and non-relevant ones, recall shows higher values while precision shows lower values.

Table 4 shows the average recall and precision of DR values for different combinations of collections and query sets.

Table 4. Average recall and precision of DR values for different collections and query sets:

<table>
<thead>
<tr>
<th>Recall</th>
<th>Unix</th>
<th>C++</th>
<th>Precision</th>
<th>Unix</th>
<th>C++</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>0.68</td>
<td>0.85</td>
<td>Q1</td>
<td>0.87</td>
<td>0.82</td>
</tr>
<tr>
<td>Q2</td>
<td>0.69</td>
<td>0.80</td>
<td>Q2</td>
<td>0.707</td>
<td>0.71</td>
</tr>
</tbody>
</table>
We can observe from the above table that recall for the C++ collection shows higher value than for the Unix collection. This is consistent with the analysis in the general performance that the average similarity between C++ collection and query set is higher than between Unix collection and query set, since when the average similarity is larger, more components can be retrieved by the same threshold $T$, therefore the numerator gets increased. It is expected to note that the average recall and precision for query set 1 are higher than that of query set 2. Since the number of features in query set 2 is doubled compared with query set 1, it is more difficult for the components to be qualified as 'satisfying the query requirement', therefore the average recall and precision would decline.
4.4 Vector Method

The results of our experiments are shown in Figures B-1 to B-26. Note, the collection I.D. ‘C’ shown in the legend of each graph represents the C++ collection, and ‘UNIX’ represents the UNIX collection.

Recall for vector method at end of 10th iteration, App.A. ($T=0.01$)

**Figure B-1**

Recall for vector method at end of 10th iteration, App.B. ($T=0.01$)

**Figure B-2**
In Figure B-2, we observe that there is a slight dip in recall values for DR=20% (for the UNIX collection) and DR=50% (for the C++ collection). This is due to the following. As DR increases, the definition of relevance changes. Specifically, for large DR values, a component is more difficult to qualify as relevant. Accordingly, the total number of relevant components decreases but also, the number of relevant components that are retrieved decreases. The rate of decrease of these two numbers determines the fluctuation of the ratio number of relevant and retrieved components / total number of relevant components, which is the recall. It is rather encouraging to see, as Figure B-2 illustrates that this fluctuation (in recall for different DRs) is no more than 10% for the UNIX collection and about 15% for the C++ collection. Similar comments apply for Figure B-1.

![Figure B-3](image)

Precision for vector method at end of 10th iteration, App.A. (T=0.01)

**Figure B-3**
As both Figures B-3 and B-4 illustrate, the precision values are significantly different for the two collections. In Figure B-4, the UNIX collection behaves well, maintaining high precision values. For the C++ collection, we observe a decreasing trend (except for DR = 100%). This is due to the nature of the query used for this experiment. The used query is quite large, consisting of 20 features. This causes the query to have many common verb-noun pairs with many components of the collection. This translates to high similarity values (computed as dot products, as described in subsection 2.3) between the query and components, and, therefore, many components are retrieved. At the same time, as DR increases, a retrieved component is harder to qualify as relevant. This results in a drop in precision. Incidentally, the above situation is not the case for the UNIX collection, where the used query consists of only 7 features. Accordingly, as the precision curves for the UNIX collection demonstrate, this does not cause the above anomaly and the precision values remain stable, in Figure B-4. A final note for Figure B-4 is the sharp increase in
precision for DR = 100%. Although this seems to be in contradiction to the above explanation of decreasing precision for increasing DR values, when the DR goes very high, the following phenomenon occurs. During the first few iterations within the relevance feedback session, several components are retrieved. However, because of the high DR value, only very few of them qualify as relevant. Accordingly, the user informs the system of this fact, that is, that most of the retrieved components are non-relevant. Then the query vector is modified. Since there is a large number of non-relevant components, the subtracted quantities from the corresponding entries of the query vector are quite large. This causes many entries of the query vector to become zero (or even negative). In subsequent iterations during the same relevance feedback session, those entries that became zero, remain zero (according to method B). It is important to note that the nullified entries of the query vector are those that correspond to verb-noun pairs which appear in non-relevant components. Therefore, when in later iterations the dot product (query vector) x (component vector) is calculated, the resulting value would be small due to the many zeroes appearing in the query vector. As a result, the corresponding non-relevant components would not be retrieved since their dot product similarity with the query would not exceed the threshold. Note, Figure B-4 shows the precision at the end of the 10th iteration. By then, for large DR values (100% in Figure B-4), most query vector entries are zero and, therefore, mostly/only relevant components are retrieved. The above effect of increasing DR value becomes more apparent in Figure B-3 for the C++ collection, where the upward precision trend starts for DR > 20%, and for the
UNIX collection for most values of DR. Although in method A also the zero entries of the query are modified, still the above discussion for Figure B-4 applies here. This is because negative entries of the query would also decrease the value of the dot product (query vector) \( \times \) (component vector). The drop in precision for DR = 10% and 20% for the C++ collection in Figure B-3, is attributed, as in Figure B-4, to the large query (20 features) that is used against this collection.

![Graph](image1.png)

Recall for vector method at end of 10th iteration, App.A. (T=0.07)

**Figure B-5**

![Graph](image2.png)

Recall for vector method at end of 10th iteration, App.B. (T=0.07)

**Figure B-6**
In Figure B-6, recall still has an increasing trend. However, it starts from lower values than the ones in Figure B-2. This is due to the higher T value (of 0.07 as compared to T=0.01 in Figure B-2). In Figure B-5, recall stays high. The occasional dips are due to the reasons already discussed for Figures B-1 and B-2.
Regarding the precision in Figure B-8, the same discussion done for Figure B-4 applies here. In addition, note, overall precision values in Figure 8 are slightly better than the ones in Figure B-4. This is expected and it is also consistent with the discussion of Fig. 4. Note, in Figure B-8 we have higher T value (0.07 as compared to T=0.01 in Figure B-4). This makes it harder for the dot product similarity between a component and the query to exceed the threshold, and, consequently, it makes it harder for the components to be retrieved. The precision curves in Figure B-7 are also very close to the ones of Figure B-3.

![Figure B-9](image)

Recall for vector method at end of 10th iteration, App.A. (DR=20%)
Recall for vector method at end of 10th iteration, App.B. (DR=20%)

Figure B-10

Figures B-9 and B-10 show recall values for varying T. The rather noticeable drop in recall in Figure B-10 is expected since as T increases fewer components produce a dot product similarity that exceeds the larger threshold T and, therefore, fewer components qualify for retrieval. Consequently, as fewer components are retrieved, also fewer relevant components make it into the retrieved components pool.

Tests for different DR values (5%, 10%, 50%, 100%) were also run (but not shown here in order to conserve space). The general observed trend is that the recall increases as DR increases. This is due to the following phenomenon. By setting a higher DR value, fewer overall components qualify as relevant in the collection. Therefore, the denominator total number of relevant components, in the ratio number of relevant and retrieved components / total number of relevant components (= recall), decreases, and this results in increased recall value. Tests for DR=50% and DR=100% (not shown here) support this trend.
Specifically, for method B, when DR=50%, the recall value for the UNIX collection was uniformly equal to 1, and the recall value for the C++ collection was 0.8 or better. For DR=100%, the recall value was uniformly equal to 1 for both collections. For method A, the recall value was uniformly equal to 1 for both DR = 50% and 100% and both collections, at the end of the 10th iteration.

![Figure B-11](image1)

**Figure B-11**

![Figure B-12](image2)

**Figure B-12**
Figures B-11 and B-12 show the precision curves for the same parameter values used for Figures B-9 and B-10. As T increases, although (as discussed in Figures B-9 and B-10) fewer components are retrieved, Figure B-12 shows that the overall quality of the retrieved components remains good and fairly stable for method B. On the other hand, an interesting phenomenon debuts in Figure B-11. As we see, the precision curves for either collection exhibit rather significant fluctuations (this is more apparent for the UNIX collection in Figure B-11). For example, for the UNIX collection, method A exhibits precision values fluctuating within a wide range (high 85%, low 55%). Also, there is no clear trend in the precision curves (this is more apparent for the C++ collection, but it is also vivid for the UNIX collection). Although the UNIX collection precision curve shows a downward trend after $T = 0.04$, it fluctuates up and down for all previous T values. For the C++ collection, there is no trend whatsoever. This puzzling behavior of precision for method A becomes even more apparent in Figures B-15 and B-19 below. For the moment, we just state that this behavior is intrinsic to the query modification formula used in method A. Following the discussion of Figures B-15 and B-19, we then present a detailed explanation of this phenomenon (supported by Figures B-21 to B-26).
In all above tests we presented results showing recall and precision when varying DR and T. In all those results, only a single snapshot of the relevance feedback process was presented, namely the status of recall and precision at the end of the 10th iteration. This
does not actually reveal the true merits of any relevance feedback method. This is because the effectiveness of any relevance feedback method is determined by the quality of retrieval (in terms of recall and precision) within the *same* relevance feedback session, i.e., when both DR and T are kept fixed and successive iterations are performed. Figures B-13 through B-20, show such results.

In Figure B-14 we observe that there is an increase in recall, followed by stabilization. Figure B-13 exhibits good behavior of method A in terms of recall since after persisting for 6 relevance feedback iterations, the recall stabilizes to 1 (for the UNIX collection). In Figure B-16, the precision is highest for the smallest recall values (which occur during the first few iterations in Figure B-14) and then falls (this is typical in information retrieval environments and it is also attributed to the reasons stated during discussion of Figures B-3 and B-4). In any case, this fall in precision is neither alarming nor damaging to the overall retrieval quality, since the precision never falls below 80%. In fact, at the 5th iteration (Figure B-14), the recall attains its highest value (and then stabilizes) for both collections while still the precision is above 85%. On the other hand, we see that method A has a very erratic behavior in precision (Figure B-15) (such behavior has been also observed earlier, in Figure B-11). This phenomenon is explained below.
In Figure B-16, the precision remains high for both collections.
Recall for vector method for each iteration, App.A. (T=0.1, DR=10%)

**Figure B-17**

Recall for vector method for each iteration, App.B. (T=0.1, DR=10%)

**Figure B-18**

In Figure B-18, the recall shows again non-decreasing trend. Note, however, it starts from lower values than the ones in Figure B-14. This is due to the higher T value (0.1, as compared to t=0.04 in Figure B-14), which makes it more difficult for the similarity values to exceed the threshold value.
In Figure B-20, the precision remains high for both collections.
Note, the recall and precision values in Figs B-14, B-16, B-18, and B-20 are for DR=10% (and only two values of T). Tests with higher DR values (not shown here) and the full tested range of T values (0.01 to 0.1) revealed even better recall and precision values. Specifically, for method B, when DR is 20% or above, we got an average recall value of 0.94 (over 300 runs, for values DR = 20%, 50%, 100%, and T = 0.01 to 0.1, for 10 iterations per relevance feedback session), and average precision of 0.9 for the UNIX collection; we got an average recall of 0.86 and precision of 0.69 for the C++ collection.

For method A, when DR = 20% or above, we got an average recall value of 0.93, and average precision of 0.86 for the UNIX collection; we got an average recall value of 0.91, and average precision of 0.78 for the C++ collection. These average precision values for method A are quite good. On the other hand, unlike Figures B-16 and B-20 (method B), in Figures B-15 and B-19 (method A) we observe that there is significant fluctuation in the precision curve for both collections. The fluctuation is so great that it is basically impossible to identify a pattern in the precision values and allow one to have any educated expectations of what the precision value will be from one iteration to the next. This makes method A quite unpredictable and unattractive for the relevance feedback process when the DR value is small.

The wide fluctuation in the precision values Figures B-15 and B-19 is attributed to the way that the query Q is modified in successive iterations within the same relevance feedback session. In method A, all entries of the query vector, including the zero ones,
are modified. This causes the overall similarity between $Q$ and a component to drop in some cases. As Figures B-15 and B-19 show, that drop causes a very erratic behavior in the precision.

Figures B-21 to B-26 next, show how this erratic behavior in precision is connected with the average similarity values. Figures B-21, B-23, and B-25 show the precision curves for both methods A and B, for the UNIX collection, for different DR values (10%, 20%, 50%), while Figures B-23, B-22, and B-26 show the corresponding *average similarity* curves. By average similarity we mean the average value of the dot product similarities of all retrieved components within one iteration of a relevance feedback session. In all Figures, the shown curves cover all tested $T$ values (0.01 to 0.1) and all 10 iterations of a relevance feedback session.
In Figure B-21 (small DR value) we see that there is wide fluctuation in precision values for method A within every relevance feedback session, while the precision for method B remains stable (at 1). We argue that this fluctuation is due to the way that the query is modified in method A. This is supported by the average similarity curve shown in Figure
B-22. We observe that while the average similarity for method B exhibits an uninterrupted growth, this is not the case for the average similarity curve of method A which drops in many occasions before it resumes its upward trend. These drops are due to the modification of the zero entries of the query during the relevance feedback process. What happens is that the subtracted quantity from the zero entries outweighs the added quantity in these cases.

![Graph showing precision for vector method for each iteration and T.(DR=20%)](image)

**Figure B-23**
The situation becomes a bit more stable in Figures B-23 and B-24, where the DR value has been increased. Figure B-24 shows that the fluctuation in average similarity is not as great as the one in Figure B-22. In addition, a careful look at Figure B-23 shows that the fluctuation in precision is not as wide as the one in Figure B-21. The reason for this change in behavior is the increased DR value. As DR increases from 10% (in Figs B-21 and B-22) to 20% (in Figs B-23 and B-24), the query vector contains fewer zero entries, since more reference components are now relevant to the query. Therefore, the opportunity of decreasing the average similarity by subtracting from zero entries, is now less. This results to fewer drops in average similarity for method A. In conjunction with Figure B-24, as Figure B-23 illustrates, the “correction” in the similarity curve is also reflected in the precision curve of method A.
Finally, in Figures 25 and 26, the differences between methods A and B are very minimal. Due to the additional increase in DR to 50%, even fewer entries of the query vector are zero. This results into even fewer opportunities of subtracting from zero entries of the query vector, and therefore, fewer opportunities for decreasing the average similarity. The result is an uninterrupted growth of the similarity curve of method A,
making it almost identical to the one of method B (Figure 26). Accordingly, this behavior is reflected into the corresponding precision curves (Figure 25), which are almost identical for both methods A and B. The same patterns occur if the DR is increased to 100% (those results are not shown here).

Overall, the following conclusions can be drawn from the above experiments.

- Both methods A and B constitute viable relevance feedback methodologies in terms of attained recall values (all Figures support this).
- Smaller T values result to better recall for both methods A and B (Figures 9, 10).
- A high threshold results to generally better precision for method B for any DR (Figures 7, 8). For a low threshold, method B is also generally better (in terms of precision), but more marginally so (Figures 3, 4).
- Low DR values result to erratic behavior of method A, in terms of precision (Figures 11, 15, 19, 21, 23), and good and predictable behavior (in terms of precision) of method B.
- For high DR values, both methods exhibit good and predictable behavior in terms of precision (Figures 25, 26).
4.5. Comparison of the two methods

In order to compare the retrieving performance of the ‘vector method’ with that of the ‘structure method’, the same sets of query and software collection are used to test both methods. The following figures are obtained by running the query set 3 for the Unix and C++ collections.

4.5.1 Comparison of the results for two methods.

The recall and precision at the end of 10th retrieval with $T=0.01, \ldots, 0.10$ and $DR=5\% \ldots 100\%$ are snapped for both methods and collections, and results are shown in figure set C. Table 5 shows the average recall and precision at the end of 10th iteration for both methods.

<table>
<thead>
<tr>
<th></th>
<th>Structure Method</th>
<th>Vector Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>Unix</td>
<td>0.77</td>
<td>0.70</td>
</tr>
<tr>
<td>C++</td>
<td>0.70</td>
<td>0.70</td>
</tr>
</tbody>
</table>

We can observe from the table that the average recall and precision values of the structure method are not as high as that of the vector method. We will use the following figures to observe the retrieval performance for both methods and analyze and explain the performance in the paragraphs following the observation.
The figures are presented in the order of T value: from low, medium to high. The graphs that show the general performance with fixed T value and different DR values come the first in each T category, followed by the graphs that show the performance in each relevance feedback session with fixed both T and DR, which are presented in the increasing order of DR. As the C++ collection shows a more obvious trend, all the graphs shown below are selected from the C++ collection.

Figures C-1 and C-2 show the recall values at the end of the 10th iteration for the two methods when T is small (0.01) and DR varies from 5% to 100%.

The generally observed trend is that the recall for the structure method remains in relatively high value while the precision shows a decreasing trend with the increase of DR. The decreasing trend for the precision can be explained by the following: With the increase of DR, fewer components are qualified as relevant. Therefore, the total number of relevant components decreases, and the total number of relevant and retrieved components decreases as well. As T is fixed, the total number of retrieved components does not change, therefore the denominator in the formula of precision does not change while the numerator decreases. For the vector method, there is a dip observed in recall while precision shows no trend at all.
Recall for both methods of iteration avg. (T=0.01, C++)

Figure C-1

Precision for both methods of iteration avg. (T=0.01, C++)

Figure C-2

Figures C-3 and C-4 show the retrieval behavior within one feedback session for both methods when T=0.01, DR=5%, i.e., when both T and DR are small.
Recall for both methods for each iteration. (T=0.01,DR=5%,C++)

**Figure C-3**

Precision for both methods for each iteration. (T=0.01,DR=5%,C++)

**Figure C-4**

From the graphs we can see that the precision of the vector method shows no trend and the recall in the third retrieval drops abruptly. Other graphs with lower DR values also support the argument. The recall and precision for the structure method shows very stable, and recall is improved in the second retrieval followed by stabilization.
Figures C-5, C-6 show the recall and precision for both methods within one relevance feedback session when $T=0.01$ and $DR=100\%$, i.e., when $T$ is low and $DR$ is high.

Recall for both methods for each iteration. ($T=0.01, DR=100\%, C++$)  
**Figure C-5**

Precision for both methods for each iteration. ($T=0.01, DR=100\%, C++$)  
**Figure C-6**
Figures C-5 and C-6 show that recall for both methods stays high. There is a big jump in precision for the vector followed by stabilization. The precision for the structure method stays very low without any improvement.

Figures C-7, C-8 show the recall and precision at the end of 10th iteration when T is medium (0.05), and DR varies from 5% to 100%.

Recall for both methods of iteration avg. (T=0.05,C++)

**Figure C-7**
Precision for both methods of iteration avg. (T=0.05,C++)

Figure C-8

Recall for both methods for each iteration. (T=0.05,DR=50%,C++)

Figure C-9
The recall of the vector method main relatively high and precision increases from 0.6 with DR=5% to 1.0 with DR=100%. But there is still no obvious trend for both recall and precision with the increase of DR anyway. The structure method shows more obvious trend: recall increases with the increase of DR while precision decreases with the increase of DR. Recall starts from very low point (0.02), and gradually increases to 1. Precision remains high when DR increases from 5% to 20%, and drops when DR increases from 20% to 100%. Let's look at the performance of recall for both methods within one feedback session when T=0.05 and DR=50%, i.e., when both T and DR is taking the middle value as figures C-9 and C-10 show.

**Figure C-10**

Precision for both methods for each iteration (T=0.05, DR=50%, C++)
We can observe from the figures that the recall for both methods show exactly the same behavior. The precision is still erratic for the vector method but very stable for the structure method.

Figures C-11 and C-12 show the recall and precision at the end of 10th iteration when $T$ is large (0.10) and DR varies from 5% to 100%.

![Graph showing recall and precision](image)

Recall for both methods of iteration avg. ($T=0.10, C++$)

Figure C-11
Figures C-11 and C-12 show a very similar behavior with C-5 and C-6 for both methods, except that the recall with DR=5% shown in figure C-11 is even lower than that in figure C-7. This is because when T is higher and DR is the same, there are fewer components that can be retrieved. Therefore the number of relevant and retrieved components, the numerator in the formula of recall, decreases, while the denominator remains the same as DR is the same. This causes a decrease in recall. Figures C-13 and C-14 show the performance within the same relevance feedback session with T = 0.10 and DR = 5%, i.e., when T is high and DR is low.

We can observe from figures C-13 and C-14 that the recall for the vector method shows a steady increase, while for the structure method, it keeps lower values without any
improvement. The precision is not stable for the vector method but remains high for the structure method.

Recall for both methods for each iteration. (T=0.10,DR=5%,C++)

**Figure C-13**

Precision for both methods for each iteration. (T=0.10,DR=5%,C++)

**Figure C-14**
Figures C-15 and C-16 show the performance within the same relevance feedback session with $T=0.10$ and $DR=100\%$, i.e. when both $T$ and $DR$ are large.

We observe from the figures C-15 and C-16 that the recall for both methods stays high and there is a big jump in the precision followed by stabilization for the vector method, while precision remains low in the graph of structure method.

![Graph showing recall for both methods for each iteration.](image)

Recall for both methods for each iteration. ($T=0.10, DR=100\%, C\leftrightarrow$)

**Figure C-15**
The following table 6 summaries the performance of recall and precision for both methods with different combinations of T and DR.

**Table 6  Recall and Precision for both methods with different combinations of T and DR**

**Recall**

<table>
<thead>
<tr>
<th></th>
<th>T = L</th>
<th>T = M</th>
<th>T = H</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR = L</td>
<td>Improved &amp; structure</td>
<td>Stay High</td>
<td>Stay Low</td>
</tr>
<tr>
<td>DR = H</td>
<td>Improved &amp; method Stable</td>
<td>Stay High</td>
<td>Gradually</td>
</tr>
<tr>
<td>DR = M</td>
<td>Fluctuate &amp; vector High</td>
<td>Stay High</td>
<td>Improved</td>
</tr>
<tr>
<td>DR = L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DR = H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Precision

<table>
<thead>
<tr>
<th></th>
<th>T=L</th>
<th>T=M</th>
<th>T=H</th>
</tr>
</thead>
<tbody>
<tr>
<td>DR=L</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DR=H</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Stay High</th>
<th>Stay Low</th>
<th>Stay Medium</th>
<th>Stay High</th>
<th>Stay Low</th>
</tr>
</thead>
<tbody>
<tr>
<td>Struct M.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector M.</td>
<td>Fluctuate</td>
<td>Large Improved &amp; Stable</td>
<td>Fluctuate</td>
<td>Largely Improved &amp; Stable</td>
<td>Fluctuate</td>
</tr>
</tbody>
</table>

From the table we observe that the structure method works well with low values of both T and DR, while the vector method fluctuates without showing any trend. This is due to the following. When T is low, many components can be retrieved by both methods, and many components can be identified as relevant with low DR values as well. As the query contains relatively large number of features (21), the features in the reference components are contained by the query and many relevant and retrieved components as well. Hence the value of tn_rel for each feature in the reference component is relatively large. When more features in the reference components are involved in the relocation process, it is more likely that the similarities of more components that are similar to the relevant and retrieved ones would get increased, therefore both recall and precision would be improved and show high values (Structure Method). For the vector method, as many retrieved components qualified as relevant are with low DR values, adding more relevant components which have lower relevancy with query would make the zero entries of the
query vector to be increased, making the average similarity between the components and query very erratic within one feedback session and attracting more non-relevant components into the retrieval pool. This would have impact on the performance of next retrieval, causing both recall and precision to fluctuate. Note, the unpredictable behavior of both recall and precision at the end of 10th iteration when T is fixed and DR increases is also caused by the same reason (Vector Method). With the increase of DR, recall for both methods stay high, as fewer components are qualified as relevant ones in the whole universe, and all those components can be retrieved by both methods. The precision for the vector method behaves well when DR is high, since the opportunity of increasing the zero entries of the query vector is less). The precision for the structure method remains low. Since when DR is high, fewer components that are retrieved are qualified as relevant, the value of tn_rel is too small to have the enough number of features in the reference components involved in the relocation process. Therefore less improvement happens in the same relevance feedback session, precision stays low (Structure Method).

When T increases, not too many changes happen for both method when DR is high, but precision of the vector method with lower DR values narrows down. Since when T is high, the average similarity of the retrieved components are high, therefore the average degree of relevancy of retrieved components are relatively larger, so that the fluctuation of the precision is smoother (Vector Method). The performance of Recall of the structure method with low DR values is deteriorating when T increases, due to the fact that there
are fewer components retrieved and tn_rel value of each feature of the reference component is low (Structure Method).

4.5.2 Conclusion of the comparison

After comparison, we can conclude that:

1) The structure method shows more stable performance than the vector method.

2). Substantial improvement of both recall and precision is shown in the vector method while slight improvement of both recall and precision is shown in the structure method when T and DR are high.

3). Better performance is shown by the structure method when both T and DR are low rather than by the vector method. Precision improves with the increase of DR for the vector method.

4) There is no clear trend for both recall and precision when T is fixed and DR increases for the vector method. While for the structure method, recall shows increasing trend and precision shows decreasing trend with the increase of DR.
Chapter 5.

Future Work Directions

The test results of the structure method show that the retrieval performance of the structure method is based, to a large extent, on the quality and quantity of features in the reference components involved in the relocation process. Therefore, the criteria for defining the qualified features that should be involved in the relocation process in order to improve the performance of the retrieval becomes the bottleneck of the structure method. Currently, the criterion used is whether $t = t_{n_{rel}} - t_{n_{nrel}} > 0$. Is the value ‘0’ the choice under which we can achieved the best performance? A threshold $T$ should be defined based on the large quantity of test cases. This is left for future investigation.
Appendix A

Definitions and Notations:

Some definitions and notations used throughout this document are reproduced here for ease of reference.

1. Definition of Mapping:

a. Mapping between software component and reference component set:

Given k reference components $R_1, R_2, ..., R_k$, for each component $C_i = (f_1, ..., f_m)$, define

$$ M: (f_1, f_2, ..., f_m) \rightarrow (R_1, R_2, ..., R_k), \text{ as } M(f_1, f_2, ..., f_m) = (f_{1^*}, f_{2^*}, ..., f_{k^*}). $$

b. Mapping between reference component sets:

$$ M_{rr}: Set_k(R_1, R_2, ..., R_k) \rightarrow Set_m(R'_1, R'_2, ..., R'_m) $$

2. Definition of vector format of software component

Component $C_i$ can be expressed as $(f_{i1}, f_{i2}, ..., f_{ik})$, where $f_{ij} = similarity(C_i, R_j), i = 1, ..., N$

and $j = 1, ..., k$

3. Definition of vector format of query component

Query component $Q$ can be expressed as $(f_1, f_2, ..., f_k)$, where $f_j = similarity(Q, R_j), j = 1, ..., k$

4. Query modification formula

$$ Q_{j+1} = Q_j + \frac{1}{n_1} \sum_{i=1}^{n_1} R_i - \frac{1}{n_2} \sum_{i=1}^{n_2} S_i $$

5. Notation of similarity
\[ \text{similarity}(C, R_n) = \sum_{i=1}^{n} \text{similarity}(C, f_i) \]

6. Notation of similarity mapping

MN: similarity \_ OLD \rightarrow similarity \_ NEW, defined as

\[ MN(ac + bd) = (a + k_1) \cdot (c + k_2) + (b - k_1) \cdot (d - k_2) \quad (1) \]

the right-hand-side of (1) (i.e., the new similarity) is equal to

\[ ac + av_k + c_k + k_1, k_2 + bd + k_1, k_2 - bk_2 - dk_1 \], which is equal to

\[ ac + bd + 2k_1 k_2 + (a-b)k_2 + (c-d)k_1 \quad (2) \].

7. Definition of recall and precision

**Recall** is defined as the ratio \( \frac{\text{number of retrieved and relevant components}}{\text{total number of relevant components in the collection}} \).

**Precision** is defined as the ratio \( \frac{\text{number of retrieved and relevant components}}{\text{number of retrieved components}} \).
APPENDIX B

Proof of the theorem

\[ \text{similarity}(S_n, R_m) = \sum_{i=1}^{n} \text{similarity}(S_n, f_i) \]

**Proof:** In the following, \( E_{ij} \) is the element of the EQ matrix, and \( I_{ij} \) is the element of the Imp matrix as defined in the Fugini et al method. The EQ matrix expresses the degree of keyword compatibility between the \( i \)-th feature of the source description and the stored description. The IMP matrix is used to show the importance between the \( j \)-th feature of the stored description, and the \( i \)-th feature of a source description. The IMP matrix provides the degree of satisfaction that a source description is compatible (or can be replaced with) a stored description.

As:

\[
\text{similarity}(S_n, R_m) = \begin{pmatrix} E_{11} & E_{12} & \ldots & E_{1m} \\ E_{21} & E_{22} & \ldots & E_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ E_{n1} & E_{n2} & \ldots & E_{nm} \end{pmatrix} \cdot \begin{pmatrix} I_{11} & I_{12} & \ldots & I_{1n} \\ I_{21} & I_{22} & \ldots & I_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ I_{m1} & I_{m2} & \ldots & I_{mn} \end{pmatrix} \cdot W = \]

115
(Each element of the vector at the right hand side of the above equality is the summation of the product of \(E_i\) (ith row) and \(I_j\) (jth column).)

\[
\begin{bmatrix}
\sum_{j=1}^{n} E_{1j} \cdot I_{j1} & \sum_{j=1}^{n} E_{1j} \cdot I_{j2} & \ldots & \sum_{j=1}^{n} E_{1j} \cdot I_{jn} \\
\sum_{j=1}^{n} E_{2j} \cdot I_{j1} & \sum_{j=1}^{n} E_{2j} \cdot I_{j2} & \ldots & \sum_{j=1}^{n} E_{2j} \cdot I_{jn} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{j=1}^{n} E_{nj} \cdot I_{j1} & \sum_{j=1}^{n} E_{nj} \cdot I_{j2} & \ldots & \sum_{j=1}^{n} E_{nj} \cdot I_{jn}
\end{bmatrix} \cdot W.
\]

\[
similarity(S_n,f_i) = \begin{pmatrix} E_{11} \\ E_{21} \\ \vdots \\ E_{n1} \end{pmatrix} \cdot \begin{pmatrix} I_{11} & I_{12} & \ldots & I_{1n} \end{pmatrix} \cdot W = \\
\begin{pmatrix} E_{11} \cdot I_{11} & E_{11} \cdot I_{12} & \ldots & E_{11} \cdot I_{1n} \\
E_{21} \cdot I_{11} & E_{21} \cdot I_{12} & \ldots & E_{21} \cdot I_{1n} \\
\vdots & \vdots & \ddots & \vdots \\
E_{n1} \cdot I_{11} & E_{n1} \cdot I_{12} & \ldots & E_{n1} \cdot I_{1n} \end{pmatrix} \cdot W
\]

\[
similarity(S_n,f_m) = \begin{pmatrix} E_{1m} \\ E_{2m} \\ \vdots \\ E_{nm} \end{pmatrix} \cdot \begin{pmatrix} I_{m1} & I_{12} & \ldots & I_{mm} \end{pmatrix} \cdot W = \\
\begin{pmatrix} E_{1m} \cdot I_{m1} & \ldots & E_{1m} \cdot I_{mm} \\
E_{2m} \cdot I_{m1} & \ddots & \vdots \\
\vdots & \ddots & \ddots \\
E_{nm} \cdot I_{m1} & \ldots & E_{nm} \cdot I_{mm} \end{pmatrix} \cdot W
\]

Therefore,
\[
\sum_{i=1}^{m} \text{similarity}(S_n, f_i) = \left( \begin{array}{cccc}
\sum_{j=1}^{m} E_{1j} \cdot I_{j1} & \sum_{j=1}^{m} E_{1j} \cdot I_{j2} & \cdots & \sum_{j=1}^{m} E_{1j} \cdot I_{jm} \\
\sum_{j=1}^{m} E_{2j} \cdot I_{j1} & \sum_{j=1}^{m} E_{2j} \cdot I_{j2} & \cdots & \sum_{j=1}^{m} E_{2j} \cdot I_{jm} \\
\vdots & \vdots & \ddots & \vdots \\
\sum_{j=1}^{m} E_{mj} \cdot I_{j1} & \sum_{j=1}^{m} E_{mj} \cdot I_{j2} & \cdots & \sum_{j=1}^{m} E_{mj} \cdot I_{jm}
\end{array} \right) \cdot W.
\]

Consequently, \( \text{similarity}(S_n, R_m) = \sum_{i=1}^{m} \text{similarity}(S_n, f_i) \).
References


