In Proceedings of the 2011 IEEE/ACM International Conference on Automated Software Engineering, (c) IEEE 2011

Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques

Xiaofan Chen
Department of Computer Science
University of Auckland
Auckland, New Zealand
xche044@aucklanduni.ac.nz

John Grundy
Centre for Computing & Engineering Software Systems
Swinburne University of Technology
Melbourne, Australia
jgrundy@swin.edu.au

Abstract— **Documentation written in natural language and source code are two of the major artifacts of a software system. Tracking a variety of traceability links between software documentation and source code assists software developers in comprehension, efficient development, and effective management of a system. Automated traceability systems to date have been faced with a major open research challenge: how to extract these links with both high precision and high recall. In this paper we introduce an approach that combines three supporting techniques, Regular Expression, Key Phrases, and Clustering, with a Vector Space Model (VSM) to improve the performance of automated traceability between documents and source code. This combination approach takes advantage of strengths of the three techniques to ameliorate limitations of VSM. Four case studies have been used to evaluate our combined technique approach. Experimental results indicate that our approach improves the performance of VSM, increases the precision of retrieved links, and recovers more true links than VSM alone.**

Keywords-component; Traceability, Vector Space Model, Regular Expression, Key Phrases, Clustering

I. INTRODUCTION

Source code alone is not sufficient to capture all information about a software system. Software requirements, architectural decisions, detailed design, tutorials and user documentation, and various types of technical system documentation (e.g. deployment configuration) are important artifacts produced while engineering software systems. Tracing and maintaining interrelationships between these various forms of software documentation and source code enables software engineers to better understand systems, undertake improved maintenance of systems, and ultimately to produce higher quality systems [2, 3, 25]. However, this relies on retrieving high quality candidate links between elements in one artifact (e.g. code constructs) and elements in another (e.g. requirements and detailed design documentation). A set of high quality candidate links represents a link set between these artifacts that contains as many correct links as possible and as few fault links as possible. Moreover, a high quality candidate link set should connect elements of different artifacts at a fine-grained level of detail e.g. part of a design document description and its related source code elements. However, it is very challenging to automatically extract high quality candidate links between the wide variety of artifacts created during the software development life cycle [2, 13, 22, 29].

Many traceability recovery techniques have been invented to retrieve traceability links between artifacts [2, 3, 5-7, 9, 10, 14, 18, 21, 23, 26, 29, 32]. Some need human intervention [9, 10, 18]; others can automatically generate traceability links [2, 3, 5-7, 14, 21, 23, 26, 29, 32]. Unfortunately, no recovery approaches have the capability of recovering all possible links between artifacts automatically and accurately. This is due both to the inherent imprecision when expressing things in natural language and inherent information loss or addition when moving between software artifacts at differing levels of abstraction. Some potentially useful and important links are missed by existing techniques. Similarly, some incorrect or unuseful links are extracted and may confuse developers.

Most existing automated traceability techniques adopt a single approach to trace link retrieval. However, different link retrieval approaches have different strengths and weaknesses. To try and improve the performance of automated traceability link retrieval, we have developed an approach that combines a Vector Space Model (VSM) IR approach, with three supporting techniques: Regular Expression (RE), Key Phrases (KP), and Clustering. These particular techniques have quite different strengths and weaknesses and recover different sets of links due to their vastly different retrieval approaches. Our approach attempts to take advantage of strengths of these techniques to automatically recover links between artefacts at both high precision and high recall. Our particular focus is on retrieving links between class entities and sections in documents written in natural language, e.g. tutorials, handbooks, developer or user's guides, API documentation, architecture documentation, design rationale, emails and so on. The objective of this research is to demonstrate whether our new composite traceability link recovery approach can improve the automatic recovery of traceability links with high precision and recall. We have conducted a detailed experiment with four case studies to evaluate the strengths and weaknesses of our approach. Analysis of experimental results demonstrates that our approach improves the performance of VSM, increases the precision of retrieved links, and recovers more true links than VSM alone.

This paper is organized as follows. Related work is discussed in Section 2. Section 3 describes our traceability link recovery approach and each technique we have applied. A description of the implementation of our tool is described in Section 4, followed by the experimental results in Section

5. Section 6 analyzes these results. Finally, we draw conclusions in Section 7.

II. Related Work

Due to the importance of traceability link recovery, extensive effort in the software engineering research community has been put into improving the precision and recall of recovered traceability links between documents and code through various traceability recovery techniques. These approaches can be classified into two main groups: semi-automatic recovery and automatic recovery.

A. Semi-automatic Techniques

Semi-automatic recovery techniques are those that need human intervention during the traceability link extraction process, such as rule-based, scenario-driven, and value-based approaches. Rule-based approaches [17, 18] use traceability rules to define traceability relations between tracing documents. Since these approaches are dependent on grammatical structures present in the natural language sentences, traceability rules have to be expanded to allow generation of relations that consider all possible grammatical structures. Moreover, building rules is time-consuming.

The scenario-driven technique [9] combines the hypothesized traces and test scenarios that are executed on a running software system to generate traceability relations. It requires test and usage scenarios to be linked to classes in the source code, and it does not support tracing links to program variables and other types – other than classes. In addition, the correctness and completeness of the hypothesized traces largely affects the quality of recovered links.

The value-based approach [10] does not treat every artifact as equally important, so not all trace relationships are equally important in the context of traceability. The value-based approach produces high quality trace relationships among high-value artifacts on a finer-grained level of detail, but the quality of relationships among low-value artifacts is undesirable because they are based on a coarser-grained level of detail. Although this approach can save cost due to its focus on artifacts with high value, the determination of the value of every artifact is complex and time-consuming.

B. Automatic Techniques

Automatic recovery techniques include lightweight and heavyweight techniques. Lightweight techniques do not require pre-computation of the input and can be directly executed at run-time. Bacchelli et al [5, 6] build regular expressions to match class names to words in emails. Their experimental results show that the Regular Expression (RE) approach achieves good accuracy. The drawback is that this approach fails to retrieve links between classes and emails where class names don't explicitly appear but are mentioned implicitly, such as an email that describes tasks that a class should fulfill but does not directly mention its name.

Heavyweight techniques, by contrast, require pre-processing of their input. These techniques include Information Retrieval (IR) and Text Mining (TM). Many traceability recovery techniques to date make use of a variety of Information Retrieval (IR) approaches [2, 3, 7, 14, 21, 23, 26, 29] to automatically recover traceability links. However, the accuracy rate of link recovery by using IR heavily relies on a cut point; only links that have a similarity value greater than or equal to the cut point are shown to users [7, 21]. The same cut point may or may not be suited to different software systems. Using a low cut point retrieves a larger number of accurate (true) links than using a high cut point, but more incorrect (fault) links are captured at the same time.

Antoniol et al. [2] apply two different IR models, Probabilistic Model (PM) and Vector Space Model (VSM), to extract links between code and documentation. The results show that IR provides a practical solution for automated traceability recovery. The two IR models have similar performances when terms in artifacts perform a preliminary morphological stemming. A traceability recovery tool based on PM was developed to explore how the retrieval performance can be improved by learning from user feedback [3]. The results show that significant improvements are achieved both with and without preliminary stemming [3, 21]. Cleland-Huang et al. [7] propose an approach to improve the performance of dynamic requirements traceability by incorporating three different strategies into PM, namely hierarchical modeling, logical clustering of artifacts, and semi-automated pruning of the probabilistic network. The results indicate that the three strategies effectively improve trace retrieval performance.

Settimi et al. [26] investigate the effectiveness of VSM and VSM with a general thesaurus for generating links between requirements, code, and UML design models. The comparison results show that precision and recall are not improved by the use of the general thesaurus. Hayes et al. [14, 15] use VSM but with a context-specific thesaurus that is established based on technical terms in requirement documents to recover links between requirements. The results show that improvements in recall and sometimes in precision are achieved. Marcus and Maletic [23] introduce Latent Semantic Indexing (LSI), an extension of the VSM, to recover links between documentation and source code. The results show that LSI achieves very good performance without the need for stemming as required for PM and VSM. Wang et al. [29] present four enhanced strategies to improve LSI, namely, source code clustering, identifier classifying, similarity thesaurus, and hierarchical structure enhancement. The comparison results indicate that this approach has higher precision than LSI and PM, but has lower recall. Although various strategies have been applied to enhance the performance of IR techniques, no approaches to date can largely decrease fault links at low cut points and significantly increase true links at high cut points [2, 7, 23, 26, 29].

The TM technique organizes related texts in documents to extract domain-specific information from texts [28, 31]. Witte et al [32] employ Information Extraction, a subfield of TM, to capture traceability links through extracting entities (e.g. methods, classes, packages, etc.) from software documents. Its limitation is that it can only extract from documents salient facts about pre-specified types of events, entities, or relationships, though it generates relationships with high accuracy [12, 31]. Types of entities have to be pre-

defined, and grammar rules have to be built for detecting complex named entities.

To varying degrees, none of the traceability recovery techniques developed so far is able to produce sufficiently consistent and high enough quality results to meet developer's needs. Semi-automatic techniques are unable to generate traceability links automatically without human intervention. It is difficult to employ these techniques to retrieve traceability links between artifacts in a system for people who are unfamiliar with the system. Although automatic techniques improve this issue, their limitations impede them from capturing all potential true links and few fault links.

III. OUR APPROACH

In order to recover traceability links at a high-level of precision and recall, we have explored an approach incorporating three supporting techniques, Regular Expression (RE), Key Phrases (KP), and Clustering, into a Vector Space Model (VSM) to recover links between sections in documents and class entities. Our approach is intended to overcome the limitations of VSM by taking advantage of strengths of RE, KP, and Clustering.

We use an IR model, VSM, as the fundamental basis of our approach as VSM can retrieve all potential links with appropriate queries. However, VSM has three main limitations [1, 2, 7, 15, 21, 23, 26, 27, 29, 30]. First, very few true links are retrieved at high cut points. Second, many fault links are captured at low cut points. The third limitation is that VSM misses links in the following two situations: class names that do not follow a common naming convention strategy; and documents that use different words to describe related classes. Combining the first supporting technique, Regular Expression (RE), with VSM allows extraction of more true links at high cut points. As long as class names are retrieved correctly and refined regular expressions are built, RE can retrieve all possible links that are related to these class names and return few fault links as well.

We added the second technique, Key Phrases (KP), to our approach to recover links missed by VSM. We extend the VSM queries to include key phrases of comments in the source code. If code is well commented, KP can extract key phrases from code comments closely related to classes. Clustering, the third technique incorporated, aims to eliminate fault links at low cut points by refining existing retrieved traceability links. As the aim of our approach is to trace useful links between class entities and sections in documents, we take advantage of the inherent hierarchical structure of documents to cluster links retrieved by VSM, RE, and KP. Therefore, our combination approach increases the precision at any cut point and retrieves links with a high recall. The following section describes the four techniques used in detail.

A. The Basic Retrieval Approach

Information Retrieval (IR) is widely used in searching fields such as web search engines and library document search. We decided to employ an IR technique as the foundation of our traceability links retrieval approach as its query-based approach has potential to recover all types of link, if appropriate queries are constructed. The IR engine we employed is Apache Lucene, which is a full-featured text search engine written in Java [4]. We chose this as it is broadly used for IR experimentation and practice. Lucene uses VSM to index text and determine how relevant a section is to a query [4, 20]. As many papers have extensively discussed VSM [1-3, 7, 14, 21, 23, 29], we only briefly describe how queries are built and similarity scores of links are calculated.

A class name (or identifier) composed of two or more words is split into separate words. A query string for VSM is established by using the OR operator to combine the name and the separated words. For example, DragSource is split into the words drag and source, then the query string is "DragSource OR drag source OR drag OR source". The query is case-insensitive.

The output of the indexing process is a *term-by-document* matrix, where *term* represents all words that occur in documents, and *document* indicates all documents in the VSM corpus. Each entry $a_{i,j}$ of this matrix denotes a weight for the frequency of the i^{th} term in the j^{th} document. Each matrix column is considered as a vector that describes a document. Queries are represented in a similar way by a matrix, where each vector indicates a query. The similarity between a document and a query is measured by the cosine of the angle between the corresponding vectors. In other words, a matching document may have one or more query terms and is ranked based on the frequency of term occurrence and number of query terms present in the document [2, 20, 21]. In the end, traceability links between documents and classes are retrieved. Each link has a similarity score to display how much the related document and class is matched.

There are three main drawbacks with using VSM. The method calculating link similarity values results in some true links with a very low similarity score and the majority of retrieved links have low similarity values. Therefore, the lower the cut point that is used, the more possible links are retrieved but also the more fault links are captured as well. This leads to the first limitation that very few true links are captured at high cut points. The second limitation is that many fault links are extracted at low cut points. The third limitation is that links are missed in the following two situations: class names not following a naming convention strategy and documents using different words to describe related classes. We have found that these are both common occurrences in many software documentation artifacts.

B. Regular Expression (RE)

In order for us to augment the number of retrieved links at high cut points, a RE technique is used. A regular expression, which is a pattern of characters that describes a set of strings, is constructed and used to find all of the occurrences of this pattern in an input sequence. Here, we use REs to find class names in documents. It is case sensitive.

Class names can be placed into two groups. One group is class names containing only one word, such as Control,

Main, Graphics etc. Another is class names formed by compound words, such as NamingExceptionEvent, DragSource etc. For the second group, class names are most likely not part of common words that can be found in a dictionary. Therefore, once they appear in documents, most likely they represent class names. For the first group, class names probably belong to common words. Then we need to make sure the same words found in documents indicate class names and not other names.

For the second group, simply matching class names against their occurrence in documents suffices. From inspection of typical documents, we observe that class names can be surrounded by a wide variety of non-word characters but must exclude the hyphen "-". A hyphen attached before or after a class name can be part of another class name. For example, the string "DragSource" matches a class named "DragSource", but also a class name is written as "DragSource-Listener" in documents when a class name is separated over two lines and is connected by a hyphen: "DragSource-" is at the end of a line, "Listener" is at the beginning of the following line. It raises another issue that hyphens may exist inside class names, e.g. "DragSource-Listener". Therefore, we extend the regular expressions developed by Bacchelli et al [5, 6] to the following regular expression code (take the class named "Control" for the example):

(.*)(^a-zA-Z0-9\-)<C-?o-?n-?t-?r-?o-?l>(^a-zA-Z0-9\-)(.*)

In order to identify class names in the first group, we can additionally match different parts of the package name of a class in documents. For example, a package named javax.naming.event has three parts: javax, naming, event. It is not feasible to require the package name to be presented before the class name, because it is very rare that a package name is cited before the class name in documents. If the class name, the last part of the package name, and at least one of other parts of the package name are found, then the single word in documents denote a class name. This method also can apply to identify classes sharing the same name but belonging to two different packages. The regular expression code for matching each part of package names is:

(.*)(^a-zA-Z0-9\-)<each part of package name>(^a-zA-Z0-9\-)(.*)

These two regular expressions can correctly capture all documents directly containing class names and return few unrelated documents. Therefore, links recovered by RE are considered as true links, they are assigned with the highest similarity value. This largely expands the retrieved link sets at high cut points but does not change the fault links recovered by VSM. This approach still fails to retrieve links that are missed by VSM.

Both TM, discussed in Section 2, and RE can fulfill class name entity recognition. We found through experimentation that the results obtained from both approaches are the same. However, TM spends much more time than RE, and combining TM into our traceability recovery system made the whole system much slower than RE. Therefore, we chose to use RE rather than more sophisticated TM techniques in our current tool.

C. *Key Phrases (KP)*

Key Phrases provide a brief summary of a document's content [33]. We wanted to use the KP technique to extract key words (or key phrases) from comments of code to provide a brief summary of each class's description comment and use these to augment our VSM technique's link recovery.

There are two situations where VSM is unable to retrieve correct links. Firstly, when class names do not follow a naming convention strategy VSM struggles to retrieve documents that do not explicitly mention the class name. For example, for a class named "RefAddr", its VSM query is "RefAddr OR ref addr OR ref OR addr", VSM is unable to retrieve documents not containing "RefAddr" as "ref" and "addr" are not common words. Secondly, documents implicitly mentioning a class but not explicitly using the same word as the class name or separated words of the compounded class name are also problematic. For example, a class named "Media", but where documents may use "medium" to indicate this class. We have found that these two issues can be addressed by taking the comments in source code into consideration. Generally, software developers provide comments to describe the purpose of the class or what tasks the class fulfills. Extracting key phrases from comments can help find alternative words to the class name or words indicating what tasks the class fulfills. For example, "medium" indicates the class "Media", "reference address" refers to the purpose of the class "RefAddr". As long as comments in each classes are well documented, KP can extract all possible key phrases that summarize the purpose of each class. We found that adding these extracted key phrases to the VSM queries enables our approach to work in the above two contexts. However, many fault links at low cut points are also recovered.

D. *Clustering*

In general, every document has an inherent hierarchical structure. Documents are usually divided into sections with headings. Each section has a direct parent or some direct children or some siblings. There exist tangled relationships between these sections. For example, in this paper, "Section 3.A The Basic Retrieval Approach" has a direct parent, "Section 3 Our Approach", and three siblings, "Sections 3.B, 3.C, and 3.D". It has no children. Section 3.A, 3.B, 3.C, and 3.D cross-reference each other to some extent. We take advantage of these tangled relationships to reduce the number of fault links retrieved by using Clustering.

Clustering is a division of a set of objects into groups of similar objects: clusters [22]. We modify the K-mean clustering algorithm [22] to meet our needs. There are three main steps in this: initialization, assignment, and removal. We take links between the class "java.awt.dnd.DragSource" and sections in a document as an example to illustrate our clustering algorithm. Table 1 shows an example where 34 sections are related to "DragSource". Each line represents a link and lines colored blue and italicized refer to true links. Before starting the initialization step, all retrieved links are grouped based on classes; namely, links related to the same class are grouped together. Clustering is performed on each

group that represents sections related to the same class. Then the algorithm selects k clusters according to the number of links with similarity values $\geq s$. Each cluster contains one of these related sections. When the group contains links with a similarity value that equals to 1, then the algorithm uses $s = 1$. Otherwise, the algorithm uses $s = 0.3$ to create clusters. From empirical observation we found four reasons to use this latter value when none of the links' similarity value in the group is equal to 1. Firstly, a majority of the fault links have a similarity score ≤ 0.3. Secondly, links with similarity ≥ 0.3 are more likely to be true. Thirdly, if we use $s \leq 0.3$, our approach retrieves many fault links and only slightly more true links. Fourthly, if $s \geq 0.3$, our approach slightly decreases the number of fault links but does not obtain more true links. Empirically, therefore, we found the 0.3 threshold to be the best choice for the target systems used in our experiment. We need to conduct more experiments, however, to validate its suitability for other systems. In Table 1, 15 links have a similarity score = 1. The algorithm thus creates 15 clusters, each one containing one of these sections.

TABLE I. SECTIONS RELATED TO JAVA.AWT.DND.DRAGSOURCE

1.0----dnd1.pdf:2.1 Overview
1.0----dnd1.pdf:2.2.1 DragGestureRecognizer
1.0----dnd1.pdf:2.3 Drag Source
1.0----dnd1.pdf:2.3.1 The DragSource definition
1.0----dnd1.pdf:2.3.2 The DragSourceContext Definition
1.0----dnd1.pdf:2.3.5 The DragSourceDragEvent Definition
1.0----dnd1.pdf:2.3.6 The DragSourceDropEvent Definition
1.0----dnd1.pdf:2.4.3 The DropTargetContext Definition
1.0----dnd1.pdf:2.4.4 The DropTargetListener Definition
1.0----dnd1.pdf:2.4.5 The DropTargetDragEvent and …
1.0----dnd1.pdf:2.5 Data Transfer Phase
1.0----dnd1.pdf:2.5.1 FlavorMap and SystemFlavorMap
1.0----dnd1.pdf:2.5.2 Transferring Data across the JVM …
1.0----dnd1.pdf:3.0.1 What are the implications of the …
1.0----dnd1.pdf:3.0.3 Lifetime of the Transferable(s)?
0.06234840----dnd1.pdf:3.0.4 Implications of ACTION_…
0.05901812----dnd1.pdf:2.3.3 The DragSourceListener Definition
0.05479498---- dnd1.pdf:2.5.3 Transferring lists of files across…
0.05061744----dnd1.pdf:2.3.4 The DragSourceEvent Definition
0.04429026----dnd1.pdf:3.0.2 Inter/Intra VM transfers?
0.04183720----dnd1.pdf:3.0.5 Semantics of ACTION_…
0.03083606----dnd1.pdf:2.4.1 java.awt.Component additions…
0.02992645----dnd1.pdf:2.5.4 Transferring java.rmi.Remote …
0.02787049----dnd1.pdf:3.0 Issues
0.02787049----dnd1.pdf:2.0 API
0.02658593----dnd1.pdf:2.4.2 The DropTarget Definition
0.02162598----dnd1.pdf:1.1 Provision of a platform independent
0.01930924----dnd1.pdf:2.2 Drag Gesture Recognition
0.01330427----dnd1.pdf:1.2 Integration with platform …
0.01330427----dnd1.pdf:Appendix A : DropTargetPeer definition
0.01317056----dnd1.pdf:1.0 Requirements
0.00965462----dnd1.pdf:2.4.6 Autoscrolling support
0.00064342----dnd1.pdf:Appendix B : DragSourceContextPeer definition
0.00064342----dnd1.pdf:Appendix C : DropTargetContextPeer definition

Next, the algorithm assigns the direct parent, all direct children and all siblings of the initial section to the cluster, but only new sections that aren't already in other clusters and are in the retrieved link set. Take the cluster for section 2.3 in Table 1 for example: sections 2.3.1, 2.3.2, 2.3.5, 2.3.6, and 2.5 are not assigned to this cluster as they belong to other clusters, and section 2.4 is not assigned as it is not in the retrieved link set. Lines colored by red (bold) and blue (italics) indicate links included in clusters. Finally, links not in clusters are discarded. Thus, in this case, 6 links out of 34 are discarded in the group for "DragSource". We have found that our clustering approach eliminates many fault links at low cut points.

IV. IMPLEMENTATION

Figure 1 illustrates the traceability recovery process of our approach. First, if a document contains sections, it is partitioned into small sub-documents according to sections or headings (1). For example, if a PDF document contains 10 headings including all sub-headings, it is split into 10 sub-documents; the contents of each are the text between its heading and the following one. These sub-documents are then preprocessed.

Next, source code is analyzed by the code dependency analysis system in order to extract source code identifiers (every class, method, package name), and comments inside code (2). Code dependency analysis is based on Eclipse's JDT Java parser [8]. These extracted class names are passed to the Regular Expression (RE) processor to find sections that directly mention class names (3). Links retrieved by the RE processor are assigned the highest similarity score (= 1), and form the RE link set.

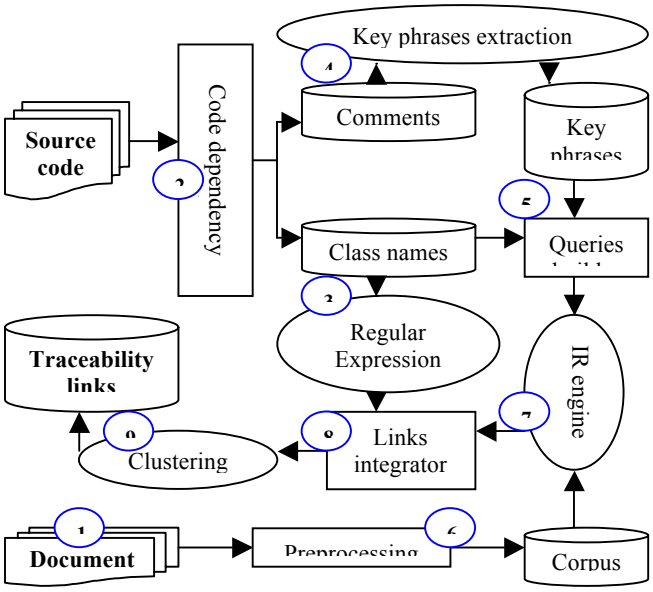


Figure 1. Traceability recovery process of our approach

At the same time, extracted comments inside code are passed to the Key phrases extraction system (4). This is based on KEA, a keyphrase extraction algorithm developed by Witten et al. [19, 33]. This extracts key phrases from comments. These extracted key phrases are combined with extracted class names to form IR queries (5). A query string for IR is established by using OR operators to combine the class name, the separated words if the class name is formed by compound words, and key phrases extracted from comments in the class code.

Before using the Apache Lucene IR engine [4, 20] to capture links between sections and class entities, sections in documents are preprocessed (6). Lucence preprocessing starts by generating tokens from consecutive letters in the text stream according to token boundaries that are defined at non-letter characters. Next, non-textual tokens (i.e. special symbols, numbers etc.) are dropped. A lower case filter

transforms all capital letters into lower case letters, and a stop-words filter removes common words (i.e. articles, adverbs, etc.). Finally, an IR corpus is generated containing all documents and words (or tokens) in the documents. The IR engine retrieves traceability links according to queries, and computes similarity scores ($0 \leq$ similarity score ≤ 1) based on the frequency and distribution of the key words or phrases (7). Recovered links forms the IR link set. The RE link set and the IR link set are then merged together (8). If a link can be found in both sets, then the one in the IR set is removed and we leave the link in the RE set (i.e. with higher rank). Finally, the merged link set passes through the Clustering system to refine the link set to produce the final candidate traceability links (9).

To make the extracted traceability links "useful" for maintainers our final step is to visualize recovered links allowing users to browse and maintain these links in a natural and intuitive way. We use a hierarchical, graphical traceability link visualization that can be expanded and contracted to enable users to interact with large numbers of extracted relationships. A screen dump of a prototype version showing links visualization is shown in Figure 2. Here a software engineer has selected a source file (PrintJob.java) and its related sections retrieved by our link recovery technique in the file (JPS_PDF.pdf) are highlighted.

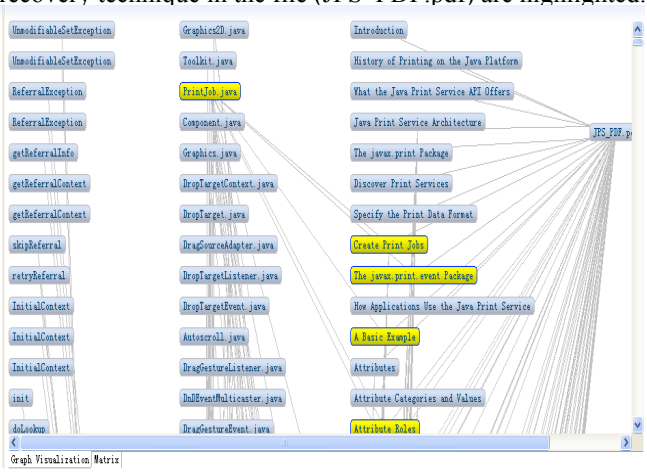


Figure 2. A screen dump from our Eclipse-based prototype tool in use.

V. EVALUATION

A. Case Studies

To validate the effectiveness of our approach, we have set up four case studies based on four unrelated software systems. The first system we used is JDK 1.5, a free software system for Java developers. Table 2 describes the packages in JDK 1.5 and their corresponding PDF documents used in this study, as well as the number of Java classes and the number of sections in them. We divided these PDF files into sections based on their headings. This case study contains 760 true links. We describe how we built the oracle traceability link set for JDK 1.5 in Section 5.C.

The systems used for the other three case studies are ArgoUML, Freenet, and JMeter. Alberto Bacchelli [6] kindly provided the three systems, their email archives, and their oracle traceability link sets. These emails were extracted from active development mailing lists of each project. Table 3 provides details of these three case studies.

TABLE II. JDK1.5 PACKAGES AND DOCUMENTS

JDK 1.5		**#classes/ sections**
Java packages	java.awt, javax.naming, and javax.print packages	249
PDF files	JPS_PDF.pdf: Java™ Print Service API User Guide	68
	dnd1.pdf: Drag and Drop subsystem for the Java Foundation Classes	41
	jndispi.pdf: Java Naming and Directory Interface™ Service Provider Interface(JNDI SPI)	73
	Total sections:	182

TABLE III. CLASS ENTITIES, EMAILS, AND TOTAL TRUE LINKS PER SYSTEM

	Classes	**Emails**	**Total true links**
ArgoUML	423	378	308
Freenet	517	372	516
JMeter	372	348	563

B. Evaluation Metrics

Precision, Recall, and F-measure are common metrics used in the evaluation of IR systems. The three metrics depend on three figures: correct (or true) links retrieved, fault links retrieved, and missing links.

Correct links retrieved are those that are correctly captured by the system. Fault links are those that are wrongly detected by the system. Total links retrieved combine these two kinds of link. Relationships that are not found by the system are called missing links. Total correct links are the sum of correct links retrieved and missing links. Precision can be defined as the ratio of the number of correct retrieved links over the total number of retrieved links. If precision equals 1, it means that all the recovered links are correct, though there could be correct links that were not recovered.

Precision = Correct links retrieved / Total links retrieved

Recall is the ratio of the number of correct retrieved links over the total number of correct links. Recall = 1 indicates that all correct links are recovered, but there may be incorrect recovered links.

Recall = Correct links retrieved / Total correct links

The F-measure combines precision and recall based on their weighted harmonic mean to measure the effectiveness of retrieval. β is an adjustable weight to favor precision over recall. We take β=1 to weight precision and recall equally.

F-measure = (β^2+1) Precision × Recall / $((\beta^2$Recall)+Precision)

Two sets of traceability links between sections in documents and class entities are prepared in order to compute precision, recall, and F-measure. One set is produced by a system under evaluation; the other is an oracle traceability link set carefully prepared manually (Section 5.3

describes how the link set for JDK1.5 is established; the oracle link sets of ArgoUML, Freenet, and JMeter area as provided by Alberto Bacchelli). The latter is critical as it is a crucial factor in determining the number of missing links. Comparison of the two sets is then conducted to determine whether a link is correct, faulty, or missing.

C. Building the Oracle Traceability Link Set

In order to build the oracle traceability link set for JDK1.5, we employed a method of manually verifying trace links by a group as used in [5, 6, 15] to build the oracle traceability link set for our case study, JDK1.5. We recruited 11 analysts: 9 analysts had at least 6 years of Java programming experience, and 2 participants had more than 9 years of Java programming experience. We set up two rules to assist participants in finding and verifying a link. First, if a section directly mentions a class identifier/name, then this section is related to this class. The second rule is that if a section describes tasks that a class should fulfill, then they are related. At the first stage, the classes were divided into 6 sets. 6 participants then manually retrieved links between sections in documents and classes by following the above two rules. After they completed their task, we asked another participant to verify these links. At the second stage, conflict links produced at the first stage were randomly divided into 3 overlapping sets. Three other participants verified these conflict links by carefully studying the text of documents and the comments inside code. After the three finished, we asked a senior participant to verify those links still having conflicts. This participant carefully studied the text of documents and the comments in code. This participant also consulted with another senior participant. Each conflict link was thus analyzed by at least 3 participants. When three reviewers agreed that the conflict link was a fault, we considered this link to be a fault link and discarded it. The final oracle link set comprised 760 true links. Our rigorous manual verification of the true links remedied any potential bias of adding incorrect links to the oracle link set [5, 6, 15].

D. Evaluation Results

To evaluate whether the three supporting techniques, RE, KP, and Clustering, ameliorate limitations of VSM, we compared the performances of four different combinations of techniques: VSM; the combination of VSM and RE; the combination of VSM, RE and KP; and our final approach VSM, RE, KP and Clustering. The following sections describe the results produced by the four different combination techniques. Every approach recovers links with a similarity score ≥ the cut point. For example, the cut point is 0.02, which denotes that all links having a similarity score ≥ 0.02 are extracted by our approach. A cut point < 0.3 is considered as a low cut point in the following discussion. Otherwise we consider it to be a high cut point. In Figure 3, we summarize the precision results of all approaches for the four case studies. Recall results of all approaches are in Figure 4.

1) The Basic Retrieval Approach: First, we used VSM to recover links between documents and class entities to discover VSM's performance at different cut points. It is obvious from precision results in Figure 4 and recall results in Figure 5 that the lower the cut point used, the lower the precision value but the higher the recall value VSM obtains. Although VSM retrieves a majority of true links at low cut points from 0 to 0.1, many fault links are extracted, especially at 0 and 0.02 cut points. VSM gets the highest precision value at 0.9 cut point for JDK1.5 and ArgoUML and at 0.7 cut point for Freenet and JMeter but only recovers very few true links.

2) VSM and Regular Expressions (RE): We then evaluated the combination of VSM and RE to verify whether RE can increase the number of retrieved links at high cut points. Figure 4 shows that adding RE to VSM improves precision at all cut points except for JMeter's 0.5 and 0.7 cut points. In Figure 5, we observe that recall is largely increased at all cut points especially for high cut points. This indicates that adding RE to VSM retrieves more true links than VSM alone.

3) VSM, RE, and Key Phrases (KP): To recover links missed by VSM, we added an additional technique, KP. From Figures 4 and 5, compared with the combination of VSM and RE, we see that after adding KP to VSM and RE, precision at all cut points is increased for JDK1.5 and ArgoUML, but recall has a slight decrease. However, there is no significant improvement in Freenet and JMeter. Compared with VSM, adding KP still increases recall at all cut points except for the slight decrease at cut points from 0.02 to 0.08 for JDK1.5. This shows that this combination retrieved more true links and less fault links than VSM.

4) VSM, RE, KP, and Clustering: Incorporating Clustering into the last combination aims to reduce the number of fault links but not deteriorate recall too much. Figures 4 and 5 show that our approach of integrating the three supporting techniques with VSM fulfills the above two objectives. Precision is largely increased at all cut points especially at low cut points. The majority of fault links are discarded at low cut points. Although our approach retrieves less true links than VSM alone at low cut points for JDK1.5 and ArgoUML, at 0 to 0.02 cut points for Freent, and at 0 to 0.04 cut points for JMeter, recall is only slightly reduced and still reaches a value ≥ 80% for JDK1.5, Freenet and JMeter, and > 62% for ArgoUML. This shows that our approach largely reduces the number of fault links without suffering from low recall at low cut points.

E. Performance of Our Approach

We ran our approach on an iMac with a 2.4 GHz Intel Core Duo processor and 3GB of RAM. Figure 3 shows that our approach took up to 5 minutes to execute on each case. For all cases, this is 4-9 times more than VSM, 2-6 times more than VSM+RE, and up to 10 seconds more than VSM+RE+KP. 80% of time for JDK1.5 and at least 60% of time for other cases are spent on KP extraction. It is because KEA, the key phrases processor in our approach, uses an expensive machine learning algorithm for training and key

phrase extraction [33]. Our approach thus produces a much better result than the other 3 combination techniques but is slower. It is vastly faster than manually extracting links. When building the oracle link set, every participant spent one hour on average to identify related sections of 50 classes.

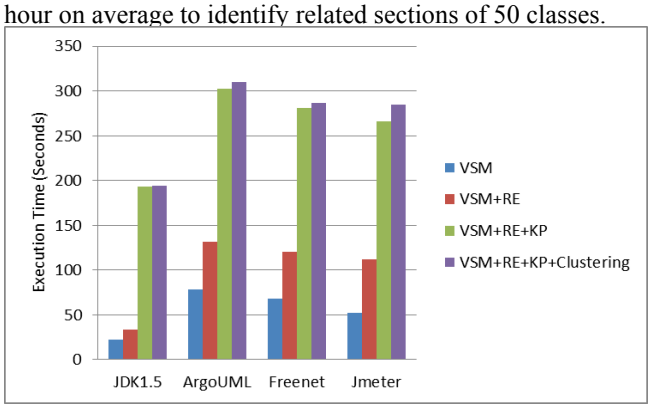


Figure 3. Execution times for different combinations

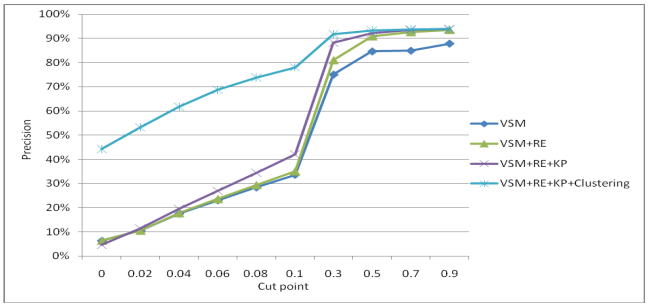
VI. DISCUSSION

According to our experimental results, precision is gradually improved through incrementally adding techniques into the combination approach, and is greatest when incorporating all three techniques with VSM. Adding RE to VSM increases precision at high cut points from 0.3 to 0.9 and recall at all cut points. Analysis of the four case studies shows that documents contain many class names that enable RE to match classes to documents. Further adding KP increases precision at all cut points for JDK1.5 and ArgoUML but slightly decreases recall. Freenet and JMeter are unresponsive to the KP technique. Analysis of source code reveals a low number of comments in their source code. In addition, the key phrases extracted from comments contain many key words unrelated to the purpose of classes. Finally, precision at low cut points from 0 to 0.1 is greatly increased by adding Clustering. Analysis of the four cases shows that documents have inherent hierarchical structure that provides useful hierarchical information for Clustering to refine retrieved links. Our approach is able to obtain good Precision at all cut points. Moreover, recall for our approach is much higher than for VSM at high cut points, but slightly less than for VSM at low cut points for JDK1.5 and ArgoUML, at 0 to 0.02 cut points for Freenet, and at 0 to 0.04 cut points for JMeter.

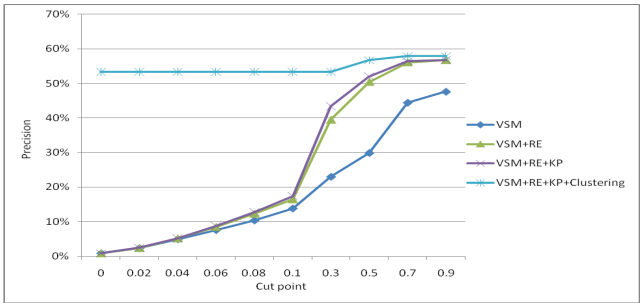
Therefore, we conclude that our approach improves the precision of retrieved links and achieves high recall by utilizing the strengths of RE, KP, and Clustering to mitigate limitations of VSM. VSM has three main drawbacks: it recovers very few links at high cut points, has low Precision at low cut points, and misses links if class names do not follow the naming convention strategy and if documents use different words to describe the related classes.

Combining RE with VSM eliminates the first drawback of VSM. Adding KP to this combination ameliorates the third drawback of VSM. Finally, integrating Clustering ameliorates the drawback of many fault links produced by VSM. Furthermore, the F-measure results of all approaches in Figure 6 show that our approach is the most effective among all approaches we evaluated if precision and recall are considered equally important.

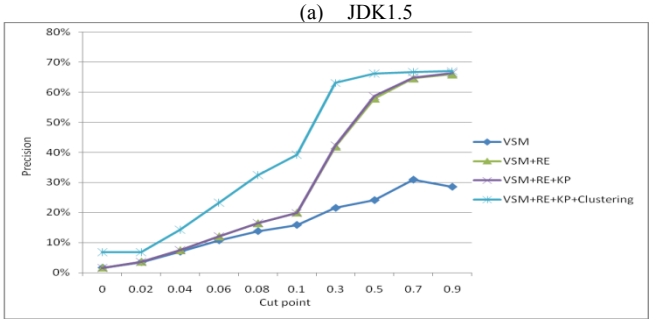
The main limitation of our approach is that some true links are discarded after adding Clustering. This is because the group containing links related to a same class is totally removed when no links in the group have a similarity value larger than the threshold *s* value, this leads to no clusters for this group being created. True links in such groups are cut.



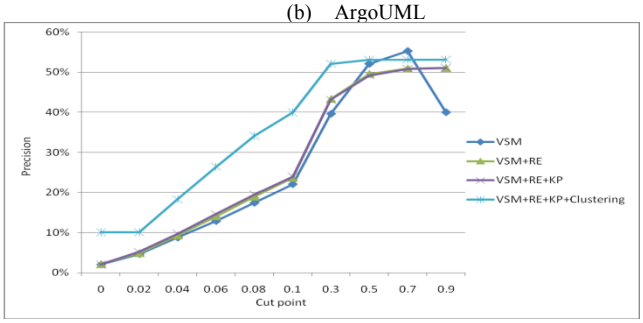
(a) JDK1.5



(b) ArgoUML

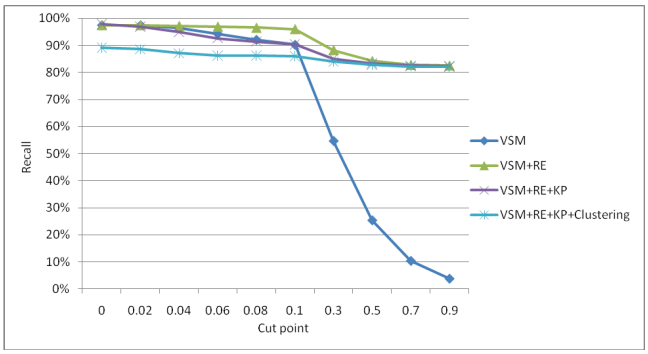


(c) Freenet

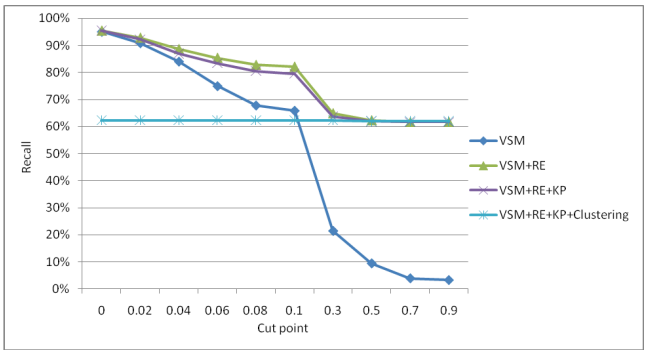


(d) JMeter

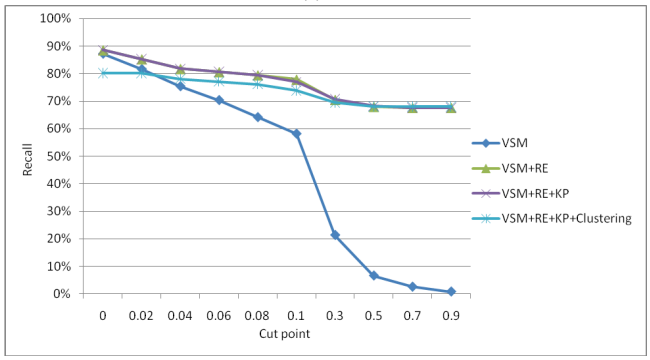
Figure 4. Precision results for (a) JDK1.5, (b) ArgoUML, (c) Freenet, and (d) JMeter



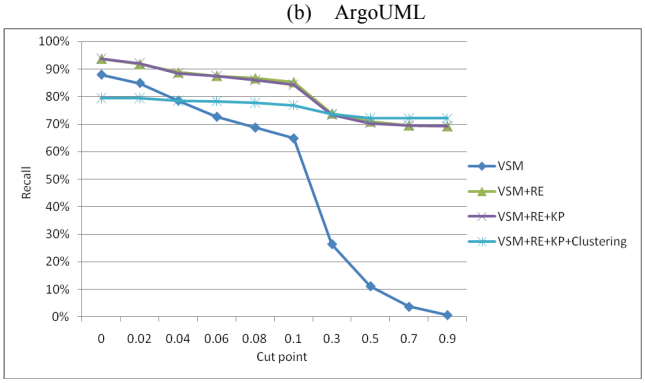
(a) JDK1.5



(b) ArgoUML

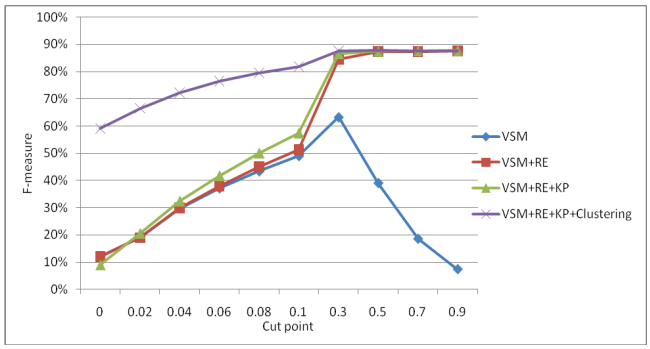


(c) Freenet

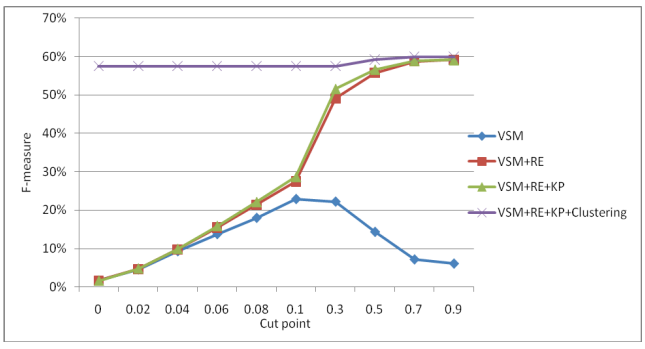


(d) JMeter

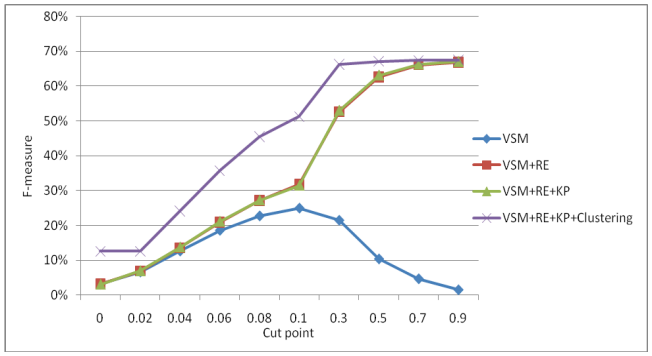
Figure 5. Recall results for (a) JDK1.5, (b) ArgoUML, (c) Freenet, and (d) JMeter



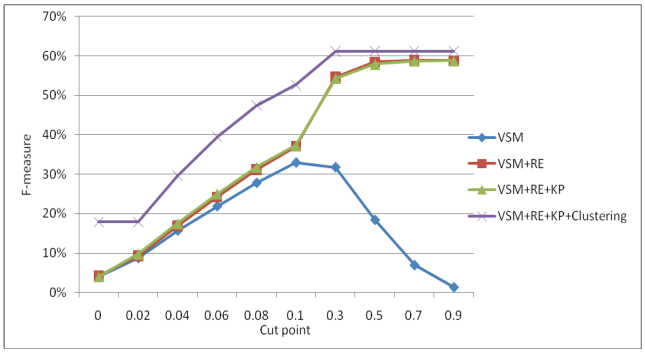
(a) JDK1.5



(b) ArgoUML



(c) Freenet



(d) JMeter

Figure 6. F-measure (β=1) results for (a) JDK1.5, (b) ArgoUML, (c) Freenet, and (d) JMeter

In future work we will experiment with allowing users to configure thresholds and select some or all techniques to apply to the extracted link set. Furthermore, we will explore the impact of other techniques to refine the extracted links such as our visual IDE's user creation and editing of links and both user and automated ranking of relationship quality.

In addition, we will carry out a usability evaluation of our traceability relationship recovery approach and our trace link visualization tool to determine how effective they are in assisting users navigate between source code elements and associated documentation elements.

VII. THREATS TO VALIDITY

First, we relied on human judgment to build the oracle link set and thus this set might not 100% correct. To alleviate this, we applied a very rigorous manual verification strategy to analyze every true link, which were verified by at least 3 analysts. Second, our traceability recovery technique may show different results when applied to other software systems with other types of documents. To alleviate this, we chose 4 unrelated open-source systems. These systems are varied in the sizes of the systems, the types of documents, the structures of documents, and the availability of comments in source code. However, we cannot confirm that our results are similar in closed-source systems.

VIII. SUMMARY

It is a major challenge for traceability recovery techniques to extract relationships between diverse artifacts of a software system at high-levels of precision and recall. Many recovery techniques exist but none so far produces sufficiently consistent and high enough quality of results that software developers require. Our traceability system incorporates three supporting techniques, RE, KP, and Clustering, with VSM to extract links between documents and class entities. The three techniques ameliorate the key limitations of VSM by taking advantage of the respective strengths of each of the three supporting techniques. Our experimental results from four different combination recovery approaches provide a demonstration that our combination recovery approach can eliminate some limitations of VSM. Our approach improves precision at all cut points, reduces fault links at low cut points, and increases the number of true links at high cut points.

ACKNOWLEDGEMENT

The authors gratefully acknowledge Alberto Bacchelli for agreeing to share his exemplar test sets and oracles, and the financial support of the Foundation for Research, Science and Technology and University of Auckland.

REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia. "Information retrieval models for recovering traceability links between code and documentation". *ICSM'00*, 2000, San Jose

[2] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. "Recovering traceability links between code and documentations". *TSE* 28 (10), Oct. 2002, pp. 970-983

[3] G. Antoniol, G. Casazza, and A. Cimitile. "Traceability recovery by modelling programmer behavior". *7th WCRE,* Queensland, Australia, Nov. 2000, pp. 240-247

[4] Apache Lucene – Overview, from http://lucene.apache.org/java/docs/

[5] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes. "Benchmarking lightweight techniques to link e-mails and source code". *In Proceeding of WCRE 2009*, pp. 205-214

[6] A. Bacchelli, M. Lanza, and R. Robbes, R. "Linking E-mails and source code artifacts". *ICSE'10*, May 2010, pp.375-384

[7] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. "Utilizing supporting evidence to improve dynamic requirements traceability". *RE'05*, Paris, Aug. 2005, pp.135-144

[8] Eclipse Java Development Tools (JDT), http://www.eclipse.org/jdt

[9] A. Egyed. "A scenario-driven approach to trace dependency analysis". *TSE*, 2003, 29(2), PP. 116-132

[10] A. Egyed, S. Biffl, M. Heindl, and P. Grunbacher. "A value-based approach for understanding cost-benefit trade-offs during automated software traceability". *TEFSE'05*, 2005, California, USA, pp. 2-7

[11] R. Fjeldstad and W. Hamlen. "Application program maintenance-report to our respondents". *Tutorial on Software Maintenance*, 1983, 13-27. Parikh, G.&Zvegintzov, N.

[12] GATE Information Extraction, extracted from http://gate.ac.uk/ie/

[13] O.G. Gotel and A.C.W. Finkelstein. "An analysis of the requirements traceability problem". *1st RE*, 1994, pp. 94-101

[14] J. H. Hayes, A. Dekhtyar, and J. Osborne. "Improving requirements tracing via information retrieval". *Proc. Int'l Conf. Requirements Eng. (RE)*, pp. 151-161, Sept. 2003

[15] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. "Advancing candidate link generation for requirements tracing: the study of methods". *TSE*, Vol. 32, No. 1, January 2006, pp. 4-19

[16] D. Jin and J. Cordy. "Ontology-based software analysis and reengineering tool integration: the OASIS service-sharing methodology". *21st ICSM 2005.* pp.613-616

[17] W. Jirapanthong and A. Zisman. "Supporting product line development through traceability". *APSEC*, 2005, pp.506-514

[18] W. Jirapanthong and A. Zisman. "XTraQue: traceability for product line systems", *Software and System Modeling* 8 (1), 2009, 1619-1366

[19] KEA: keyphrase extraction algorithm. 2010. Extracted on 1 May 2010 from http://www.nzdl.org/Kea/

[20] M. Konchady. "*Building search applications: Lucene, LingPipe, and Gate*", Musstru Publishing, Oakton, Virginia, 2008

[21] A. D. Lucia, F. Fasano, R. Oliveto, and G. Tortora. "Recovering traceability links in software artifact management systems using information retrieval methods". *TOSEM*, 2007, Vol. 16, No. 4

[22] J. B. MacQueen. "Some methods for classification and analysis of multivariate oberservations". *5th Berkeley Symp. On Math. Stat. and Prob*. 1967, pp. 281-297

[23] A. Marcus and J. I. Maletic. "Recovering documentation-to-source-code traceability links using latent semantic indexing". *25th ICSE*, 2003, pp. 125-135

[24] J. Rilling, P. Charland, and R. Witte. "Traceability in Software Engineering—Past, Present and Future". *CASCON Workshop*, IBM Technical Report: TR-74-211, October 25 2007

[25] R. Seacord, D. Plakosh, and G. Lewis. "*Modernizing legacy systems: software technologies, engineering processes, and business practices*". 2003, Addison-Wesley

[26] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. DePalma. "Supporting software evolution through dynamically retrieving traces to UML artifacts". *7th IWPSE*, 2004, Kyoto, Japan, pp. 49-54

[27] T. Standish. "An essay on software reuse". *TSE* 10 (5), 1984, 494-497

[28] A. Stranieri and J. Zeleznikow. "*Knowledge discovery from legal databases*", 2005 Vol 69, Springer

[29] X. Wang, G. Lai, and C. Liu. "Recovering relationships between documentation and source code based on the characteristics of software engineering". *Electronic Notes in Theoretical Computer Science 243*, 2009, pp. 121-137

[30] R. Watkins and M. Neal. "Why and how of requirements tracing". *5th ASM*, 1994, California, pp.104-106

[31] S. M. Weiss, N. Indurkhya, T. Zhang, and F.J. Damerau. "*Text mining: predictive methods for analyzing unstructured information.*" Springer New York, 2005

[32] R. Witte, Q. Li, F.F. Informatic, Y. Zhang, and J. Rilling. "Text mining and software engineering: an integrated source code and document analysis approach". *IET Software* 2 (1), 1, 2008, pp. 1-19

[33] I. H. Witten, G. W. Paynter, E. Frank, C. Gutwin, and C. G. Nevill-Manning. "Kea: practical automatic keyphrase extraction". *4th ACM DL*, 1999, Berkeley, pp. 254-255