New Ideas and Emerging Results Track: a Combination Approach for Enhancing Automated Traceability

Xiaofan Chen
Department of Computer Science
University of Auckland
Auckland, New Zealand
(0064 9) 373-7599 88260
xche044@aucklanduni.ac.nz

John Hosking
Department of Computer Science
University of Auckland
Auckland, New Zealand
(0064 9) 373-7599 88297
john@cs.auckland.ac.nz

John Grundy
Centre for Complex Software Systems & Services
Swinburne University of Technology
Melbourne, Australia
(0061 3) 9214-8731
jgrundy@swin.edu.au

ABSTRACT

Tracking a variety of traceability links between artifacts assists software developers in comprehension, efficient development, and effective management of a system. Traceability systems to date based on various Information Retrieval (IR) techniques 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 describe an experimental approach that combines Regular Expression, Key Phrases, and Clustering with IR techniques to enhance the performance of IR for traceability link recovery between documents and source code. Our preliminary experimental results show that our combination technique improves the performance of IR, increases the precision of retrieved links, and recovers more true links than IR alone.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement.

Keywords

Traceability; Regular Expression; Key Phrases; Clustering.

1. INTRODUCTION

In practice, artifacts produced during the software development life cycle (SDLC), such as source code, designs, requirements and documentation, end up being disconnected from each other. They are often separated into different documents, file formats and repositories, created and maintained by different individuals, and evolve at different rates [1, 2, 10]. These disconnected artifacts hinder engineers from undertaking comprehension, effective development, efficient management, and improved maintenance of a sytem. Implementing effective traceability during the SDLC can ameliorate this issue by allowing maintainers to navigate between and browse more effectively related artifacts [2, 4, 5, 7]. However, it is very challenging to extract high accuracy relationships between the wide variety of artifacts created during the SDLC [2, 5, 8, 11].

Many traceability recovery techniques [1, 2, 4, 6, 7, 9-11], based on Information Retrieval (IR) techniques, have been invented to retrieve traceability links between artifacts. Unfortunately, no IR approaches to date have the capability of recovering links between artifacts with high precision and recall. IR techniques generate traceability links by computing a similarity score based upon the frequency and distribution of terms in textual format documents. The accuracy rate of link recovery 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 [4, 7]. This means that some potentially useful and important links are missed. Similarly, some incorrect or unuseful links are extracted and may confuse developers. To enhance the performance of traceability link retrieval, we have developed an approach that combines IR with three other techniques, Regular Expression (RE), Key Phrases (KP), and Clustering, to improve recovery of traceability links. These different techniques have their own strengths and weaknesses and recover different relationships due to their differing approaches. Our approach attempts to take advantage of strengths of the three techniques to enhance the performance of IR.

Our particular focus is on retrieving links between class entities and sections in documents written in natural language. This paper aims to demonstrate whether and how our traceability link recovery approach can improve the automatic recovery of traceability links with high precision and recall. We have conducted an experiment to evaluate the strengths and weaknesses of our recovery approach. Analysis of the experimental results illustrates that our recovery approach both increases the precision of retrieved links and recovers more true links than IR alone.

2. RELATED WORK

Many of traceability recovery techniques to date make use of Information Retrieval (IR) approaches [1, 2, 4, 6, 7, 9-11] to automatically recover traceability links. 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, and 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 [1]. The results show that significant improvements are achieved both with and without preliminary stemming [1, 7]. Cleland-Huang et al. [4] 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. [10] investigated the effectiveness of VSM and VSM with a general thesaurus for generating links between requirements, code, and UML models. The comparison results show that precision and recall are not improved by the use of the general thesaurus. Hayes et al. [6] 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 [9] 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. [11] 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 can largely decrease incorrect (fault) links at low cut points and significantly increase correct (true) links at high cut points [2, 4, 9-11].

3. OUR PROPOSED APPROACH

We have been exploring a new approach combining Regular Expression (RE), Key Phrases (KP), and Clustering techniques with Vector Space Model (VSM) IR to recover links between sections in documents and class entities. Our approach is intended to overcome the limitations of IR by taking advantage of strengths of RE, KP, and clustering. Adding KP enables IR to generate all potential links. RE increases the number of true links at high cut points. The majority of fault links at low cut points are discarded by adopting Clustering. The four techniques are described in the following sections.

3.1 Information Retrieval

Our basic retrieval technique uses VSM to recover links between class entities and sections in documents. VSM queries include class names and their constituent words if a class name is formed by compound words. VSM extracts from a collection a subset of sections that is deemed relevant to a given query and assigns a similarity score ($0 \leq$ similarity score ≤ 1) to each retrieved section based on frequency and distribution of key words in the query. This can result in some accurate relationships having a very low similarity score [1, 2, 4, 6, 7, 9-11]. The lower the cut point that is used, the more possible relationships are retrieved but the more fault relationships are captured as well. In other words, at a high cut point, IR captures few links with few positive links. Another limitation is that we have found that IR can miss 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.

3.2 Regular Expression

In order for us to augment the number of retrieved links at high cut points, the RE technique is used to find all of the occurrences of class names in documents. This technique 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, the 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 have 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 [4, 5] 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 the class in documents. For example, a package named javax.naming.event has three parts: javax, naming, event. It is not feasible to require the last part of 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 same words in documents denote the class name. This method can also be used 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 capture documents directly containing class names. As links recovered by RE are considered to be 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.

3.3 Key Phrases

Key Phrases provide a brief summary of a document's content [12]. We have used 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, a class named "RefAddr", its 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 found these two issues can be addressed by taking comments of code into consideration. Generally, software developers provide comments to describe what the purpose of the class is or what tasks the class fulfills. Extracting key phrases from comments can help us to 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". 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.

3.4 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 3.1" has a direct parent, "Section 3", and three siblings, "Section 3.2, 3.3, and 3.4". It has no children. Section 3.1, 3.2, 3.3, and 3.4 cross-reference each other to some extent. We utilize these tangled relationships to reduce fault links by using the Clustering technique.

Clustering is a division of a set of objects into groups of similar objects: clusters [8]. We modify the K-mean clustering algorithm [8] to meet our needs. There are three main steps in this: initialization, assignment, and removal. 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 is equal 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 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 system used in our experiment. We need to conduct more experiments, however, to validate its suitability for other systems.

Next, the algorithm assigns the direct parent, all direct children and all siblings of the initial section to the cluster, but only new sections aren't in other clusters but are in the retrieved link set. Finally, links not in clusters are discarded. We have found that our clustering approach eliminates many fault links at low cut points.

4. IMPLEMENTATION

Figure 1 illustrates the traceability recovery process of our approach. First, documents are partitioned into small sub-documents according to sections or headings (1). Next, source code is analyzed by the code dependency analysis system to extract source code identifiers (every class, method, package name) and comments inside code (2). These extracted class names are passed to the Regular Expression processor to find sections that directly mention the class name (3). Links retrieved by the RE processor are assigned the highest similarity score (= 1), and form the RE link set.

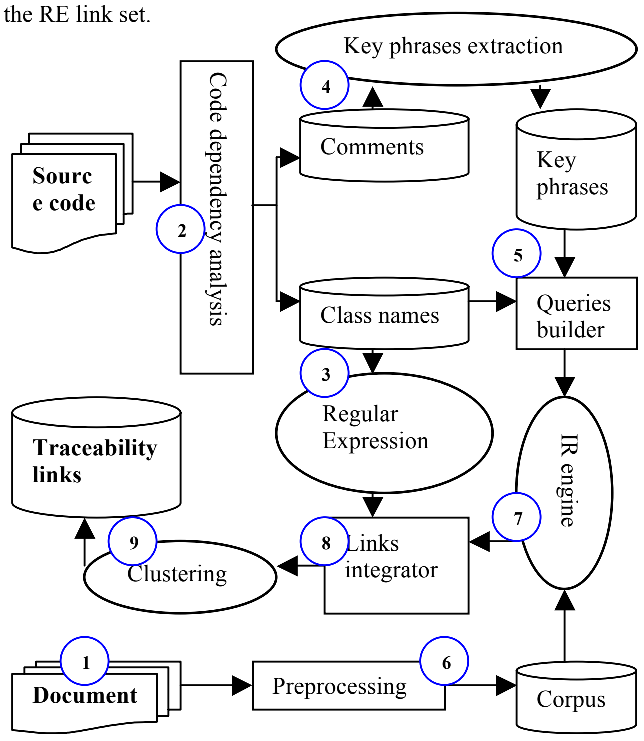

Figure 1. Traceability recovery process of our approach

At the same time, extracted comments inside code are passed to the Key phrases extraction system (4). This extracts key phrases from comments. These extracted key phrases are combined with extracted class names to form VSM queries (5). After the sub-documents are preprocessed (6), the IR engine retrieves traceability links according to queries, and computes similarity scores 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).

5. EXPERIMENTAL RESULTS

We have set up a case study, the Java SE Development Kit Version 5 (JDK 1.5) and a set of related documents, to evaluate whether our approach improves the precision and recall of retrieved links. We compared our approach with VSM. In Figure

2, we summarize the precision and recall results of the two approaches. Precisions at all cut points are significantly improved by our approach; especially at low cut points from 0 to 0.1. Our approach is able to obtain good Precision at all cut points. We observe that Recall for our approach is much higher than for VSM at high cut points from 0.3 to 0.9, but slightly lower at low cut points from 0 to 0.1. Nevertheless, our approach achieves reasonably high Recall values, above 82% at all cut points. The F-measure results of all approaches in Figure 3 show that our approach is more effective than VSM if Precision and Recall are considered equally important.

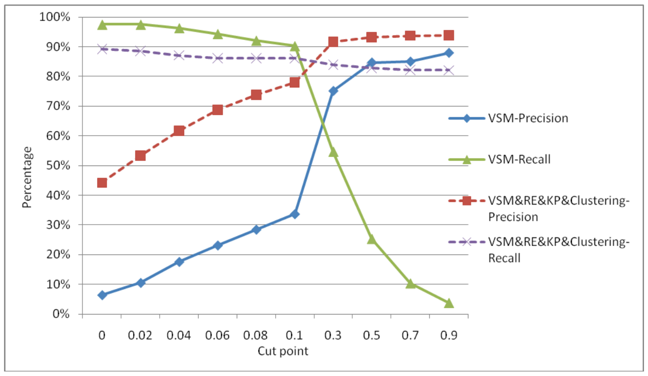

Figure 2. Precision/Recall of VSM and our approach

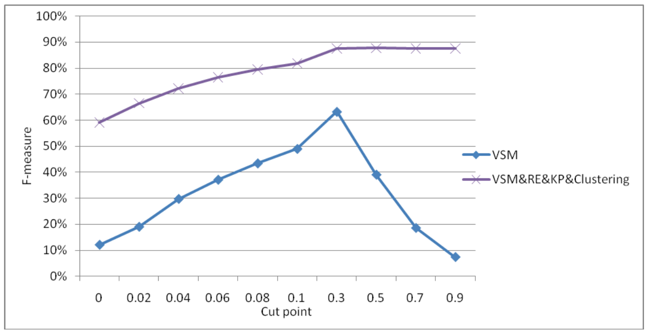

Figure 3. F-measure of VSM and our approach

Our preliminary experimental results demonstrate that our approach improves precision at all cut points and recall at high cut points. In other words, our approach largely increases true links at high cut points from 0.3 to 0.9, and significantly decreases fault links at all cut points. 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 the 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. Therefore, true links in such groups are cut.

6. IMPACT AND FUTURE DIRECTIONS

It is a major challenge for traceability recovery techniques to extract relationships between artifacts of a system at high-levels of both precision and recall. Many recovery techniques based on IR exist but none so far produces sufficiently consistent and high enough quality of results that developers require. Our approach combines RE, KP, and Clustering techniques with VSM to extract links between sections in documents and class entities. Our approach enhances VSM's performance by taking advantage of strengths of RE, KP, and Clustering to overcome VSM's shortages. The experimental results provide a preliminary demonstration that our approach eliminates VSM's some limitations, improves precision at all cut points, recovers links missed by VSM, and increases recall at high cut points.

In addition to trialling our system on other corpuses, we will experiment with allowing users to configure thresholds, select some or all techniques to apply to the extracted link set. We will also experiment with automated tuning of thresholds from user ranking of extracted relationships. Furthermore, we will explore using other techniques to refine the extracted relationships such as user creation or editing of links and ranking of relationship quality. In addition, we will carry out a usability evaluation of our approach and associated visualization tool to determine how effective they are in assisting users navigate between source code elements and associated documentation elements.

7. REFERENCES

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

[2] Antoniol, G., Canfora, G., Casazza, G., Lucia, A. D., and Merlo, E. 2002. Recovering traceability links between code and documentations. *TSE*, Vol. 28, No. 10, Oct., pp. 970-983

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

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

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

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

[7] Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. 2007. Recovering traceability links in software artifact management systems using information retrieval methods. *TOSEM*, Vol. 16, No. 4, Article 13

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

[9] Marcus, A. and Maletic, J. I. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. *25th ICSE'03*, pp. 125-135

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

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

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