# An Empirical Study on Recovering Requirement-to-Code Links

[Anonymized for Double-Blind Review]

*Abstract*—**Requirements traceability provides support for critical software engineering activities such as change impact analysis and requirements validation. Unfortunately many organizations have ineffective traceability practices in place, largely because of poor communication and time pressure problems. Therefore researchers have proposed various approaches to automatically recover requirement-to-code links. Typically, these approaches are based on Information Retrieval techniques, and use various features such as synonyms, verb-object phrases, and structural information. Although many links are thus recovered, the effectiveness of individual features is not fully evaluated, and it is rather difficult to combine different features to produce better results. In this paper, we implement a tool, called R2C, that combines various features to recover requirement-to-code links. With the support of R2C, we conduct an empirical study to understand the effectiveness of these features in recovering requirement-to-code links. Our results show that verb-object phrase is the most effective feature in recovering such links. A preliminary case study indicates that our tuning combines different features to produce better results than IR-based technique using a single feature.**

***Keywords—Requirement-to-Code Links, Traceability Recovery, Verb-Object Phrase, Information Retrieval***

***Category: other (software maintanance, software requirement)***

## I. Introduction

Software traceability is recognized as an important quality of a well-engineered software system [1]. The traceability information, including links between requirements and code, is critical in the management of software development [**?**], so many companies carefully define such traceability information before development. In practice, some companies may not define the traceability carefully, and even if such traceability initially is defined, it soon becomes obsolete, since both requirements and implementations are changing due to various factors [2]. As a result, programmers often have to maintain the links between requirements and code manually, which is time-consuming and error prone [3].

To reduce the heavy manual effort for maintaining such links, automatically recovering traceability has long been a hot research topic in the software engineering community [4]. Researchers have proposed many approaches, and most of them are based on information retrieval (referred to as IR-based approaches in this paper). Typically, IR-based approaches extract terms to represent requirements and code, and compare extracted terms to build the links between requirements and code. Although IR-based approaches successfully recover many links, their precisions and recalls are often below expectation.

We manually inspected many requirements and their corresponding code snippets and find that the following three features are widely used. First, although requirements and code use different terms, these terms are often synonyms. Second, requirements often contain many irrelevant words, whereas verb-object phrases convey the essential meanings. Third, code that linked to the same requirement always have relationships, such as function calls, inheritance or realization relationships). About the effectiveness of these different features, many research questions in this research direction are still open. For example, which is the most effective feature to recover the links? How to integrate different features for the best results?

In this paper, we conduct an empirical study to address the above research questions. This paper makes the following major contributions:

- We implement a tool, called R2C, that recovers links between requirements and code. With the support of R2C, we conduct an evaluation on four real projects. Our results show that R2C is advanced, since it achieves even better results than a state-of-the-art tool.
- Empirically, we show that verb-object phrases are the most important feature to recover the links between requirements and code, and combining various features can lead to better results than individual features.

The rest of this paper is structured as follows. Section II introduces semantic features in existing literatures. Section III presents our empirical study. Section IV introduces related work. Section V concludes.

## II. Features

In this section, we present the three key features in literatures. Different approaches implement similarity formulae with subtle differences. It is infeasible to implement all the formulae, but we implement reasonable formulae in R2C.

### A. The Three Key Features

We find that most approaches use the following features:

- **Synonyms.** IR-based approaches extract terms from requirements and code. Although these terms are not identical, they are often synonyms, when a requirement and a piece of piece are linked correctly.
- **Verb-object phrases.** In requirements, most sentences have verb-object phrases, which convey the essential meanings of sentences. We can catch the main information of the requirements with the

support of natural language processing (NLP) techniques.

- **Structural Information.** There are lots of structural information potentially contained in source code. In traceability links, source code related to the same requirement always have relationships.

## B. Synonym

Although synonyms are important, it is nontrivial to determine synonyms of terms. Some recent approaches [5], [6] define their own glossary and ontology, and their limitation lies in the huge manual effort to build such glossary or ontology. To reduce the human effort, R2C uses WordNet [7] to locate synonyms of a given term. In WordNet, each term can have several senses, and each sense is stored as a semantic tree. The leaves of a semantic tree are words with similar meanings. There are many researches in calculating the similarity between two senses of terms, a common method is based on the *synset*, *classword* and *sense explanation* in WordNet [8]. Given two terms ($S_1$ and $S_2$), their approach [8] calculates their similarity as $sim(S_1, S_2)$. However, a term may have different senses, and it is difficult to determine its sense in a context. We find that in many pairs of terms, their correct senses have the maximum values of $sim(S_1, S_2)$. As a result, we define the similarity between two terms as follows:

$$WSim(W_1, W_2) = max\{sim(S_i, S_j)\} \tag{1}$$

where $S_i$ represents a sense of word $W_1$, and $S_j$ represents a sense of word $W_2$.

## C. Verb-object Phrases

In traditional IR techniques, a document is regarded as a bag of unordered words. Typically, these techniques focus on only nouns, since they are designed to understand the key idea of a document. In such cases, nouns are often more important than the other words.

However, the assumption does not fit the context of recovering links between requirements and code. In software projects, nouns are often limited, so many nouns in arbitrary requirements and code snippets are identical. As a result, it leads to many errors, when traditional approaches try to recover links based on only nouns.

We find that in many incorrect requirement-to-code links, although their nouns are identical, their verbs are different, since they describe actions. For example, we investigated the requirements and code of a hospital information management system, and we find that both documents and code contain some nouns (*e.g.*, drug) many times. These words are often not helpful in recovering the requirement-to-code links. In the contrast, from the perspective of actions, requirements and code are distinct, since they are related to different actions (*e.g.*, buying drugs). As a result, it may achieve better results, if we extract verb-object phrases from requirements and code comments.

R2C uses Stanford Parser [9] to analyze syntactic structure of sentences. Based on the built syntax trees and Part-of-speech tags, R2C extracts the verb-object phrases from both requirements and code comments. Extracted verbs and nouns may be in morphological forms. R2C reduces these words to its root word with the stemmer of Stanford Parser. For example, it reduces $book$ and $books$ to the root word, $book$.

As code comments are different from code elements, R2C treats them in different ways:

**Requirements & Code comments.** Both requirements and code comments are in natural language, R2C extracts terms from them with the same following steps:

- Part-of-speech (POS) tagging. R2C first builds POS tags for each sentence of a given code comments. These POS tags include verbs, nouns, adjectives, adverbs, and others.
- Parsing. R2C analyzes syntactic structure of sentences and the dependency between terms.
- Extracting verb-object phrases. R2C extracts a verb-object phrase from each simple sentence or each clause of a complicated sentence. Extracted verb-object phrases are in the form of $\langle verb, noun\rangle$.
- Stemming verb-object of phrases. R2C reduces verbs and objects to their root words.

**Code elements.** Code elements are in programming languages. R2C extract verb-object phrases from code elements with the following steps.

- Extracting identifiers. R2C extracts identifiers such as class names, method names and variable names from a given piece of code.
- Extracting words. R2C splits names of identifiers into sets of words. For example, $getUserInfo$ is split into $\{get, user, info\}$, and $system.initialize$ is split into $\{initialize, system\}$.
- Recovering acronyms. R2C extends minimum edit distance algorithm [10] to compare the distances from words in code comments and an acronym, and resolves the acronym as the word with minimum distance. For example, $info$ is resolved as $information$.
- Extracting and stemming verb-object phrases. R2C combines words into simple sentences. After that, it extracts and stems verb-object phrases as it extracts such phrases from code comments.

We find that verb-object phrases between requirements and code may not be identical. For two given verb-object phrases ($\langle V_1, N_1\rangle$ and $\langle V_2, N_2\rangle$), R2C defines their similarity as follows:

$$\begin{aligned} PSim(P_1, P_2) =& \alpha \times WSim(V_1, V_2) \\ &+ (1-\alpha) \times WSim(N_1, N_2) \end{aligned} \tag{2}$$

where $WSim(W_i, W_i)$ is defined in Equation 1; $\alpha$ and $(1-\alpha)$ are the weights of verbs and nouns, respectively.

### D. Computing Text Similarity

Following other state-of-the-art tools, R2C treats requirements and code as texts and compare their similarities based on IR techniques.

To compute the similarity between requirements and code, R2C adopts Vector Space Model (VSM), a widely used model in IR. In VSM, a document is represented as an $n$ dimension vector $< w_1, w_2, \cdots, w_n >$, where $n$ represents the number of distinct terms such as words or phrases, and $w_i (1 \leq w_i \leq n)$ represents the weight of a unique term. R2C calculates the weight for each term based on Frequency-Inverse Document Frequency (TF-IDF) metric. In TF-IDF, the weight $w_i$ of a term in a document increases with its occurrence frequency in this document and decreases with its occurrence frequency in all documents. It is defined as follows:

$$w_i = tf_i \times log \frac{|D|}{|\{j : t_i \in d_j\}|} \tag{3}$$

where $tf_i$ represents the occurrence frequency of term $t_i$ in a document, $|D|$ represents the number of documents, and $|\{j : t_i \in d_j\}|$ represents the number of documents that contain the term $t_i$. It is noteworthy that we calculate TF-IDF values of terms for requirements and code separately.

The TF-IDF metric is designed for terms in natural language documents. Code is in programming languages, which are quite different from natural languages. First, class names play more important roles in expressing the functionality of code than variable names. Second, methods with more lines of code are more important than those methods with only several lines of code. Based on the above findings, we use the parameter $\eta$ to tune the weights $w_i$ of terms as follows:

$$\eta = \gamma \times log(LOC) \tag{4}$$

where $\gamma$ represents the importance of these terms, and LOC represents lines of code. If terms are not extracted from code, we set $\eta$ as 1. In addition, as discussed before, terms may not be identical but synonyms. Considering this, we tune the weight $w_i$ as follows:

$$w_i = \sum_{\{j:(P_i,P_j)\in S\}} \big(\eta \times tf_i \times idf_i \times PSim(P_i, P_j)\big) \tag{5}$$

where $\{j : (P_i, P_j) \in S\}$ represents iterating all synonymous terms $P_j$ of a term $P_i$.

After R2C builds VSM based on TF-IDF, it defines the similarity between requirements and code as the cosine of the angle between the corresponding vectors.

$$Sim(r, c) = \frac{\vec{V_r} \times \vec{V_c}}{||\vec{V_r}|| \times ||\vec{V_c}||} \tag{6}$$

where $\vec{V_r}$ and $\vec{V_c}$ are the vectors that denote requirements and code, and $||\vec{V}||$ represents the Euclidean norm of vector $V$.

### E. Structural Information

Consider an example in a small book management system. There is a requirement $AuthorizeUser$ and a source code class $User.java$. With IR-based approach mentioned before, it is easy to recover a link between them. However, there are some other correct links missed. The requirement also relevant to $Reader.java$ which extends $User.java$. Injecting structural information, R2C is able to recover this missed link because of the inheritance relationship between $User.java$ and $Reader.java$. After recovering a set of initial links, R2C next uses structural information to update similarity values between requirements and code.

```
  Input: $G(C, E)$, $s$, $Links$ and $Sim(s, c)$
  Output: $Sim(s, c)$
1 for all $(s_i, c_j) \in Links$ do
2     for $c_k \in C$ do
3         if $(c_j, c_k) \in E$ then
4             $Sim(s_i, c_k) = Sim(s_i, c_k) + \delta \times Sim(s_i, c_j)$;
5         end
6     end
7 end
8 return $Sim(s, c)$;
```

**Algorithm 1:** Updating with structural information

In Algorithm 1, $G(C, E)$ is the indirect graph of relationships between source code, in which $C = \{c_1, c_2, \cdots, c_n\}$ means the set of code and $E = \{(c_i, c_j)\}$ means the set of relationships. The relationships include call relations and inheritance relations. Moreover, let $S$ be the set of requirement documents and $Links$ be the set of initial links recovered in the previous step. $Sim(s, c)$ is the similarity calculated with Equation 6. If a requirement is linked to a code, it is probably linked to some other related code, so we increase their similarity value by adding an extra $\delta \times Sim(s_i, c_j)$. Although adding the extra values improves the similarity calculation, it is difficult to determine $\delta$. Considering the size can sensibly differ from one system to another, we set a adaptive $\delta$ as follows [11]:

$$\delta = medium\,\{(max_i - min_i)/2\} \tag{7}$$

where $max_i$ and $min_i$ are the maximum and minimum similarity values between all requirement documents and source code, and $medium$ denotes the variability.

## III. Empirical Study

In this study, we focus on two following research questions:

- To what degree does our approach combine various features to produce better results (Section III-B)?
- Which is the most effective feature to recover requirement-to-code links (Section III-C)?

To answer the first research question, we compare R2C with an IR-based tool (*baseline*) [12] and two improved tool *phrasing* [13] and *O-CSTI* [11]. The results show that R2C is an advanced tool, which increases the reliability of the results in the follow-up research question. To answer the second research question, we combine different features. Our results show that verb-object phrases are the most effective feature to recover requirement-to-code links.

### A. Setup

Table I shows the subject projects in our study. The requirements and code comments of all these projects are in English, but some nouns in eTour and EasyClinic are in Italian. In eTour, iBooks and SMS, most classes, methods have code comments and some important code lines have

TABLE I. PROJECTS FOR CASE STUDY

| Project | Requirements | Code | Links(Golden Set) |
|---|---|---|---|
| eTour | 58 | 116 | 366 |
| iBooks | 19 | 61 | 104 |
| SMS | 64 | 102 | 1071 |
| EasyClinic | 30 | 47 | 93 |

code comments as well. In EasyClinic, only classes have code comments.

In our study, we need the ground truth of links between requirements and code. SMS provides the links between its requirements and code, so we use these links as the golden standard. The other three projects do not provide such links, so we build them by experts. Comparing the manual results with our results, we calculate the recalls, precisions and F-measures of R2C.

## B. Metrics

A recovered link falls into one of the four categories, *i.e.*, a link that is identified as a true link (TP), a link that is identified as a false link (FP), a true link that is missed (TN), and a false link that is missed (FN). Based on these categories, we define recall, precision and F-measure as follows:

$$Recall = \frac{TP}{TP + FN} \tag{8}$$

$$Precision = \frac{TP}{TP + FP} \tag{9}$$

$$F\text{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{10}$$

## C. Overall Results

Table II shows the overall result. In the eTour project, the golden standard has 366 links between 58 requirements and 116 methods. The baseline approach recovered 186 (50.82%) valid links from 402 (46.27%) identified links. Phrasing and O-CSTI are better, respectively recovered 176 (48.09%) from 349 (50.43%) and 188 (51.37%) from 371 (50.67%). The R2C performed the best, recovered 197 (53.83%) valid links from 312 (63.14%) identified links. As F-measures are concerned, R2C outperforms the baseline approach from 0.4844 to 0.6170. In the iBooks project, the golden standard has 104 links between 19 requirements and 61 methods. R2C recovered 71 (68.27%) of them, with the precision 77.17%. Compared with the baseline approach, Phrasing and O-CSTI, both recall and precision in R2C are significantly higher, which leads to a better F-measure. The SMS project is largest project in our study, with 1071 links between 64 requirements and 102 methods. R2C recovered 638 (59.57%) valid links, while the baseline approach, Phrasing and O-CSTI respectively recovered 597 (55.74%), 602 (56.21%) and 591 (55.18%) valid links. In total, the other three approaches recovered 1462, 1317 and 1380 links, whereas R2C identified only 1049 links. R2C improves the precision as well, leading to a better F-measure with 0.5925. In the EasyClinic project, the golden standard has 93 links between

TABLE II. OVERALL RESULT

| Project | Links | Approach | Recall | Precision | F-measure |
|---|---|---|---|---|---|
| eTour | 366 | Baseline | 0.5082 | 0.4627 | 0.4844 |
| | | Phrasing | 0.4809 | 0.5043 | 0.4923 |
| | | O-CSTI | 0.5137 | 0.5067 | 0.5102 |
| | | R2C | 0.5546 | 0.6952 | 0.6170 |
| iBooks | 104 | Baseline | 0.4712 | 0.4495 | 0.4601 |
| | | Phrasing | 0.4519 | 0.6184 | 0.5222 |
| | | O-CSTI | 0.4231 | 0.5366 | 0.4731 |
| | | R2C | 0.6827 | 0.7717 | 0.7245 |
| SMS | 1071 | Baseline | 0.5574 | 0.4083 | 0.4714 |
| | | Phrasing | 0.5621 | 0.4571 | 0.5042 |
| | | O-CSTI | 0.5518 | 0.4283 | 0.4823 |
| | | R2C | 0.5957 | 0.6261 | 0.6105 |
| EasyClinic | 93 | Baseline | 0.4409 | 0.4227 | 0.4316 |
| | | Phrasing | 0.4516 | 0.4242 | 0.4375 |
| | | O-CSTI | 0.4624 | 0.4216 | 0.4410 |
| | | R2C | 0.4946 | 0.4946 | 0.4946 |

30 requirements and 47 methods. In both Recall and Precision, R2C is marginally higher (less than 7%) than the other three approaches. Table II shows the overall result when F-measure achieve the maximum.

As industrial applications of automated traceability are only considered successful when they achieve high recall levels [14], we also evaluate the precision of all four projects at a fixed recall level close to 0.9, and the result is shown in Figure 2.

In summary, our results show that R2C is more effective than the baseline approach. The results show that R2C is an advanced tool, and increases the reliability when we evaluate the most effective feature.

## D. The Most Effective Feature

To evaluate the effectiveness of individual features, we rerun our evaluation with different settings. In particular, based on the baseline approach, R2C-a denotes the setting when we use only synonym feature; R2C-b denotes the setting when we use only verb-object phrase feature; and R2C-c denotes the setting when we use only structural information feature. Then we also combines these features to understand their impacts on each other.

We select the iBooks project in this study, and Figure 3 shows the results. As shown in Figure 3, the combination of arbitrary two features improve the results of a single feature, and the combination of all the three features improve the results of any two features. The result highlights our tuning algorithm, since it combines features to produce better results. As far as a single feature is concerned, our results show that the verb-object phrase is the most effective feature to recover requirement-to-code links.

## E. Discussion

Our results show that R2C is more effective in recovering requirement-to-code links than the other approaches, since it

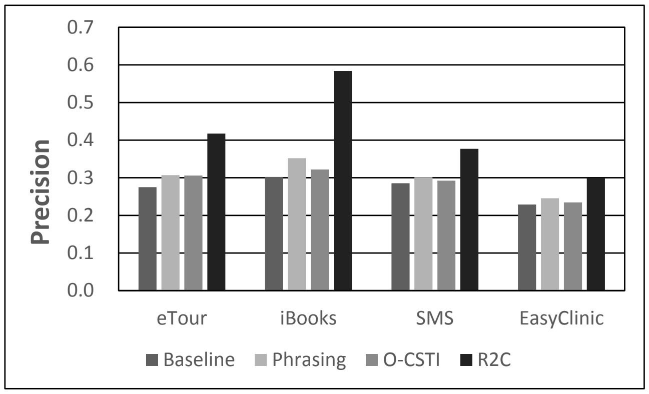

Fig. 1. Evaluate result at a fixed recall level close to 0.9

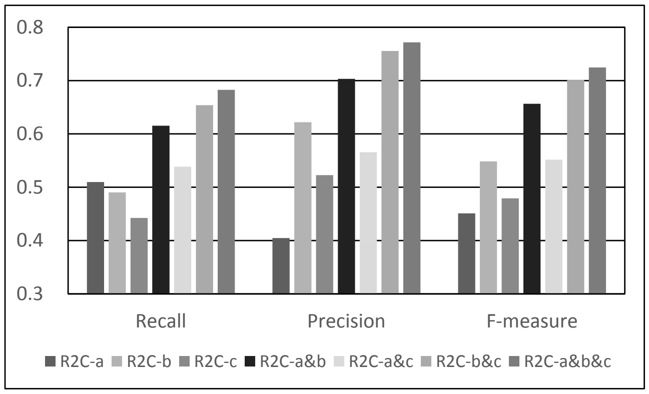

Fig. 2. The results in different settings

achieves higher F-measure 0.07 - 0.20, or higher precision 5% - 23% at a fixed recall. We find that the higher F-measure comes from more accurate extracted terms and structural information. All these four approaches are based on VSM, the baseline approach and O-CSTI extract keywords by only removing stop words, whereas R2C and Phrasing extract phrases to accurately convey the major meanings of documents without losing much information. With the initial links R2C and O-CSTI use structural information of source code to improve the result, while the baseline approach and Phrasing did not. Overall R2C integrate many techniques (*e.g.*, source code identifier processing, semantic analysis, synonym identification and source code structural analysis) to achieve the best result.

However, R2C still fail to recover some links. We find four issues for further improvements. First, as we use only a verb-object phrase to represent a sentence, it may lose information in some cases. Second, we use WordNet to find synonyms of terms. As WordNet does not include all terms, we fail to identify synonyms for some terms. Third, some programmers do not write code comments or name their identifiers meaningfully, which reduces the effectiveness of our approach. For example, we notice that sometimes programmers write their private affairs in code comments, and such comments are extracted as terms by our approach. Last, we only use function calls, inheritance or realization relationships from source code structural analysis, there may be some other useful structural information we ignored.

In our study of overall result, we find the improvement in the EasyClinic project is not so distinct as in other projects. We checked the project, and find two factors. One is that in this project, names of identifiers are in Italian, so we fail to extract proper verbs and nouns from code. The other factor is that in this project, programmers write comments for only classes, so extracted terms are insufficient to recover links.

Why verb-object phrases are so important in recovering requirement-to-code links? We manually check the documents of iBooks project and find the reasons. There is a sentence "*Create an user, and initial the information of borrowing books*" in UC5, and "*User login, and create a request of borrowing books*" in UC14. Without verb-object phrases extraction, this sentence in UC5 is represented as the vector $\{$*create, user, initial, information, borrow, book*$\}$ and the sentence in UC14 is represented as $\{$*user, login, create, request, borrow, book*$\}$. As VSM is unordered, these two vectors are so similar that both UC5 and UC14 may be linked to $User.java$ that contains a method $create()$. In fact, only UC5 should be linked to $User.java$ and UC14 has no relation with this code file. When we use verb-object phrases as the terms of vector instead of single words, the key information of these sentences are represented as follows: $\{$(*create, user*), (*initial, information*)$\}$ for UC5 and $\{$(*login, user*), (*create, request*), (*borrow, book*)$\}$ for UC14. Using verb-object phrases as the terms of VSM, the similarity between UC5 and $User.java$ still return a high value, but that between UC14 and $User.java$ return a low value with phrases mismatching. Then we can obviously identify the valid link and irrelevant link by comparing their similarity values with the threshold.

## IV. Related Work

Most approaches in literature are based on information retrieval (IR) algorithms, such as vector space model (VSM) [15] and probabilistic network model (PNM) [12] [16], which calculate the calculate of links based on the frequency and distribution of terms. For example, Hayes et al. [15] conjunct a thesaurus with the VSM to establish the links between requirements and code. LSI [17] and Latent Dirichlet Allocation (LDA) [18] [19] were applied to understand the semantics context of terms in the requirements and code from the viewpoint of probability, which is not real understand the meaning of terms. To real understand the semantics context of terms from natural language, ontology and glossaries [5] [6] were merged into existing approaches.

There are also some researchers combine textual and structural analysis of software artifacts for traceability link recovery [11]. Based on existing traceability links recovered by traditional IR-based approaches or manual search, these tools recovered more missing links by using the structural information of both documentations and source code. To improve the precision of result, researchers try some self-adapts approaches to learn from human feedback. A limited number of tools integrate user feedback. For example, ADAMS [17], POIROT [20], and RETRO [15] collect relevance feedback on the links which have been created automatically using IR techniques. Users can increase or decrease term weighting used to compute the similarity according to whether a term occurs in a rejected or confirmed link [15] [21], and they also use eye-trackers to explore how project analysts verify links between requirements and code.

However, this kind of approaches is still not good enough, studies show that the feedback is incorrect approximately 25% of the time, which may negatively impact the quality of the links [22], and the higher the quality of the starting matrix, the worse the decision the analysts make [23].

As an improvement of the existing approaches, we proposed a integrated approach to recover links Requirement-to-Code traceability links (R2C). R2C applied some new techniques (i.e. semantic analysis, and structural analysis etc.) on three widely important features, then achieved more information from requirements and code. And these information are the key for the performance improvement in recovering links.

## V. Conclusion and Future Work

This paper presents an approach R2C to recover traceability links between requirements and code. With the support of R2C, we conduct an empirical study on three features. Results show that R2C combines various features to produce better results, and the most effective feature is the verb-object phrase in recovering links.

It is worthy mentioning that several issues need further studies. First, R2C analyzes requirements only in for English, we plan to improve and apply it in other languages. Second, there are still much space for improving the precisions and recalls of our approach. Future work will be devoted to using global optimization techniques based on feedback, to further improve our approach. Finally, besides requirement-to-code links, links between other software artifacts with our techniques will be considered in future work.

## Acknowledgment

This research is supported by 973 Program in China (Grant No. 2015CB352203) , National Natural Science Foundation of China (Grant No. 61472242), and Key Lab of Information Network Security, Ministry of Public Security (Grant No. C14609).

## References

[1] Orlena CZ Gotel and Anthony CW Finkelstein. An analysis of the requirements traceability problem. In *Proc. 1st RE*, pages 94–101, 1994.

[2] Barry W Boehm. Software risk management: principles and practices. *IEEE Software*, 8(1):32–41, 1991.

[3] Jane Huffman Hayes, Alex Dekhtyar, and James Osborne. Improving requirements tracing via information retrieval. In *Proc. 11th RE*, pages 138–147, 2003.

[4] Jane Cleland-Huang, Orlena CZ Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Proc. FOSE*, pages 55–69, 2014.

[5] Jin Guo, Jane Cleland-Huang, and Brian Berenbach. Foundations for an expert system in domain-specific traceability. In *Proc. 21st RE*, pages 42–51, 2013.

[6] Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proc. 35th ICSE*, pages 522–531, 2013.

[7] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[8] Abdallah Qusef, Gabriele Bavota, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. Recovering test-to-code traceability using slicing and textual analysis. *Journal of Systems and Software*, 88:147–168, 2014.

[9] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. Generating typed dependency parses from phrase structure parses. In *Proc. LREC*, pages 449–454, 2006.

[10] Andres Marzal and Enrique Vidal. Computation of normalized edit distance and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):926–932, 1993.

[11] A. Panichella, C. Mcmillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia. When and how using structural information to improve ir-based traceability recovery. In *Proc. 17th CSMR*, pages 199–208, 2013.

[12] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[13] Xuchang Zou, Raffaella Settimi, and Jane Cleland-Huang. Phrasing in dynamic requirements trace retrieva. In *Proc. 30th COMPSAC*, pages 265–272, 2006.

[14] Jane Cleland-Huang, Brian Berenbach, Stephen Clark, Raffaella Settimi, and Eli Romanova. Best practices for automated traceability. *Computer*, (6):27–35, 2007.

[15] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[16] Jane Cleland-Huang, Raffaella Settimi, Oussama BenKhadra, Eugenia Berezhanskaya, and Selvia Christina. Goal-centric traceability for managing non-functional requirements. In *Proc. 27th ICSE*, pages 362–371, 2005.

[17] Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Enhancing an artefact management system with traceability recovery features. In *Proc. 20th ICSM*, pages 306–315, 2004.

[18] Alex Dekhtyar, Jane Huffman Hayes, Senthil Sundaram, A Holbrooke, and Olga Dekhtyar. Technique integration for requirements assessment. In *Proc. 15th RE*, pages 141–150, 2007.

[19] Hazeline U Asuncion, Arthur U Asuncion, and Richard N Taylor. Software traceability with topic modeling. In *Proc. 32nd ICSE*, pages 95–104, 2010.

[20] Jun Lin, Chan Chou Lin, Jane Cleland-Huang, Raffaella Settimi, Joseph Amaya, Grace Bedford, Brian Berenbach, Oussama Ben Khadra, Chuan Duan, and Xuchang Zou. Poirot: A distributed tool supporting enterprise-wide automated traceability. In *Proc. 14th RE*, pages 363–364, 2006.

[21] Yonghee Shin and Jane Cleland-Huang. A comparative evaluation of two user feedback techniques for requirements trace retrieval. In *Proc. 27th SAC*, pages 1069–1074, 2012.

[22] Wei-Keat Kong, Jane Huffman Hayes, Alex Dekhtyar, and Olga Dekhtyar. Process improvement for traceability: A study of human fallibility. In *Proc. 20th RE*, pages 31–40, 2012.

[23] David Cuddeback, Alex Dekhtyar, Jane Huffman Hayes, Jeff Holden, and Wei-Keat Kong. Towards overcoming human analyst fallibility in the requirements tracing process (nier track). In *Proc. 33rd ICSE*, pages 860–863, 2011.