# Information Retrieval Methods for Automated Traceability Recovery

**Andrea De Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk**

## 1 Introduction

Today's software systems are extremely large and include a multitude of artifacts, in addition to the source code. A software system includes artifacts such as: source code, design documents, requirement documents, test cases, bug reports, communications between stakeholders, etc. These are created and maintained over long periods of time by different people. Establishing and maintaining explicit connections between software artifacts is recognized to be a difficult yet important problem. This problem is being addressed from multiple angles. Development processes, such as, model driven or test driven development, address this issue partially. New integrated development environment, such as IBM's Jazz[1] also aim at simplifying this task. Defect tracking systems, such as, Bugzilla,[2] also provide support for this problem. The few cases, usually in mission critical software or certain companies, where explicit traceability between artifacts exists, are the exception rather than the norm. In such cases, this is achieved with very high costs that are prohibitive in commercial settings. In consequence, due to the absence of integrated solutions and cost effective commonly accepted practices, the reality is that most of the existing software systems lack explicit representations of traceability links between artifacts. Legacy systems suffer even more of this problem. Even in cases where efforts were made to establish traceability links among artifacts, they are often obsolete, as different artifacts evolve at different speeds and there are no widespread solutions to maintain existing traceability links. The need for tools and techniques to recover traceability links between artifacts in legacy systems is particularly important for a variety of software evolution tasks. These include general maintenance tasks, impact analysis, program comprehension, and more encompassing tasks such as reverse engineering for redevelopment and systematic reuse.

1 http://www-01.ibm.com/software/rational/jazz/

2 http://www.bugzilla.org/

R. Oliveto (✉)
University of Molise, Pesche (IS), Italy
e-mail: rocco.oliveto@unimol.it

A major challenge in the recovery of traceability links between software artifacts is the fact that these artifacts are in different formats and at different abstraction levels. More than that, sometimes the semantics of such links is interpreted differently by various people. For example, the `main()` function of a C++ program can be considered as relevant to all test cases or to none of them, as it will be executed in all scenarios. An added challenge is the fact that there is no defined data format for software engineering data and artifacts, so database and data analysis centered approaches are impractical. However, there is one type of data present in all software artifacts: textual data. Extracting and analyzing this data is essential to the development of traceability link recovery tools and techniques. In most artifacts the textual parts are descriptive in nature, that is, they describe the informal semantics of the artifacts. The assumption is that if the textual content of two artifacts refer to similar concepts, then the two artifacts are conceptually related and a traceability link between them could be established.

One solution adopted by researchers and practitioners to extract and analyze the textual data embedded in software artifacts is the use of Information Retrieval (IR) techniques (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993). IR-based methods recover traceability links on the basis of the similarity between the text contained in the software artifacts. The higher the textual similarity between two artifacts, the higher the likelihood that a link exists between them. A distinct advantage of using IR techniques is that they do not rely on a predefined vocabulary or grammar. This allows the method to be applied without large amounts of preprocessing or manipulation of the input, which drastically reduces the costs of link recovery.

This chapter introduces a generic process for the use of IR techniques for the recovery of traceability links between software artifacts. It also describes in details the most common IR techniques used in this process and the main technical challenges with such applications and their evaluations.

## 2 Using IR Methods for Traceability Recovery

The foundation for applying IR-based methods to traceability link recovery is based on the similarity between the words in the text, which are contained in various software artifacts. The conjecture is that if two artifacts have high textual similarity then they are likely to refer to the same or similar concepts and they are good candidates to be linked with each other. The underlying principle behind this is that many artifacts, such as software documentation, or even source code, contain ample textual descriptions (Antoniol et al., 2002; Dekhtyar et al., 2004) and most programmers use meaningful words from the problem domain to name source code entities, such as identifiers and comments (Antoniol et al., 2002, 2007; Haiduc and Marcus, 2008).

The process for traceability link recovery using IR methods (a.k.a. trace retrieval) consists of several key steps:

1. document parsing, extraction, and pre-processing;
2. corpus indexing with an IR method;

3. ranked list generation;
4. analysis of candidate links.

Taken as a whole, the process is organized in a pipeline architecture, where the output from each step constitutes the input for the next step. In the first step, the software artifacts are extracted at the given granularity level (e.g., class, method, or paragraphs), then they are pre-processed and represented as a set of documents in the resulting corpus. In the second step, the traceability recovery technique uses an IR method (e.g., Latent Semantic Indexing (Deerwester et al., 1990)) to index diverse software artifacts and represent them in a homogeneous document space by extracting information about the occurrences of terms (or words) within them. This information is used to define similarity measures between various documents (i.e., software artifacts). In the third step the IR-based traceability recovery method compares a set of source artifacts (represented as documents) against another set of target artifacts and uses the defined similarity measure to rank all possible pairs by their similarities (*candidate traceability links*). Once these candidate links are generated, they are provided as a result to software engineers for examination. The software engineer reviews the candidate links, determines those that are actual links (*confirmed links*), and discards the *false positives*. In order to do this, the software engineer examines the text of the software artifacts having a candidate link, determines the purpose of these artifacts (e.g., the meanings of the requirements or the functionality of source code), compares the meanings, and makes the decision based on whether she determines that the meanings of these artifacts are adequately related. The process of candidate link evaluation is based on human judgment and thus has all the advantages and disadvantages associated with such activities. The results (confirmed links and false positives) from the candidate link evaluation step may also be used to provide feedback to the IR tool to improve the tracing accuracy (De Lucia et al., 2006b; Di Penta et al., 2002; Hayes et al., 2006). The next subsections describe in details the first three steps of an IR-based traceability recovery process, while the approaches exploited to analyze the candidate links are presented in a separate section (Section 4).

## 2.1 Document Parsing, Extraction and Pre-processing

The majority of IR-based traceability recovery approaches have been applied to software artifacts, such as requirements,[3] source code,[4] external documentation,[5]

[3] See e.g., (Antoniol et al., 2000a; 2000b, 2002; Capobianco et al., 2009a, 2009b; Cleland-Huang et al., 2005, De Lucia et al., 2004, 2006a, 2006b, 2007; Di Penta et al., 2002; Hayes et al., 2003, 2006; Lormans and Van Deursen, 2005, 2006; Lormans et al., 2006, 2008; Marcus and Maletic, 2003; Marcus et al., 2005; Oliveto et al., 2010; Settimi et al., 2004; Zou et al. 2007).

[4] See e.g., (Antoniol et al., 1999, 2000a, 2000b, 2002; De Lucia et al., 2004, 2006a, 2006b, 2007; Capobianco et al., 2009a, 2009b; Di Penta et al., 2002; Marcus and Maletic, 2003; Marcus et al., 2005; Oliveto et al., 2010; Settimi et al., 2004).

[5] See e.g., (Antoniol et al., 1999, 2000a, 2002; Marcus and Maletic, 2003; Marcus et al., 2005).

design documentation,[6] test cases[7], defect or bug reports (Yadla et al., 2005), and emails (Bacchelli et al., 2010).

IR-based traceability recovery approaches extract and represent information from textual software artifacts using different granularities depending on the type of software artifact. Techniques operating on source code artifacts parse these artifacts using a developer-defined granularity (that is, methods, classes, function, or files). While several granularities are applicable to source code artifacts, the majority of recovery methods parse and represent the artifacts at a class level granularity (see e.g., (Antoniol et al., 2002; De Lucia et al., 2007; Marcus and Maletic, 2003)). This makes sense in Object-Oriented software systems, as clases are the primary decomposition unit supported by the programming languages. Traceability recovery methods using other types of artifacts (e.g., requirements, external documentation, design documents, bug reports) represent these artifacts using a user-defined granularity level, which varies from application to application and depends on the physical and logical representation of these artifacts. In such cases, a decision is required on how to partition these artifacts into atomic documents.

Traceability recovery techniques apply different pre-processing strategies on textual documents represented in the corpus, before indexing them with a specific IR method. The frequently used preprocessing steps are:

- *text normalization*: prunes out white spaces and most non-textual tokens from the text (i.e., operators, special symbols, some numerals, etc.);
- *identifier splitting*: splits into separate words terms composed of two or more words. IR techniques may miss occurrences of concepts if identifiers are not split. Similarly, incorrect splitting can cause a decrease of the accuracy of program search techniques (Enslen et al., 2009). To split multi-word identifiers, most existing automatic software analysis tools that use natural language information rely on coding conventions (Antoniol et al., 2002). When simple coding conventions, such as camel casing and non-alphabetic characters (e.g., "_" and numbers), are used to separate words and abbreviations, automatically splitting multi-word identifiers into their constituent words is straightforward. However, there are cases where existing coding conventions break down (e.g., SIMPLETYPENAME). In these cases more sophisticated approaches have to be used (see e.g., (Enslen et al., 2009; Lawrie et al., 2010; Madani et al., 2010)).
- *stop word removal*: an artifact generally contains common words (i.e., articles, adverbs, etc.) that are not useful to capture the semantics of the artifact content. A stop word function and/or a stop word list are applied to discard such words. The stop word function prunes out all the words having a length less than a fixed

[6] See e.g., (Capobianco et al., 2009; De Lucia et al., 2004, 2006a, 2006b, 2007; 2009b; Lormans and Van Deursen, 2005; 2006; Lormans et al., 2006, 2008; Settimi et al., 2004).

[7] See e.g., (Capobianco et al., 2009a, 2009b; De Lucia et al., 2004, 2006a, 2006b, 2007; Lormans and Van Deursen, 2005, 2006; Lormans et al., 2006, 2008).

threshold, while the stop word list is used to remove all the words contained in a given word list. Generally, good results are achieved using both the stop word function and the stop word list (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993). Stop word lists are language specific, for example, English has different stop words than Italian.

A more complicated document pre-processing is represented by morphological analysis, like stemming. Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form. The stem need not be identical to the morphological root of the word. It is usually sufficient that related words map to the same stem, even if this stem itself is not in a valid root. A stemmer for English, for example, should identify the string "cats" (and possibly "catlike", "catty", etc.) as based on the root "cat", and "stemmer", "stemming", "stemmed" as based on "stem". A stemming algorithm reduces the words "fishing", "fished", "fish", and "fisher" to the root word, "fish". There are several existing stemming algorithms, one of the most popular stemmers for the English language is the Porter stemmer (Porter, 1980). Not all stemmers work the same. Some stemmers are more conservative than others and may generate more false positives or false negatives. However, IR-based traceability link recovery techniques are not very sensitive to the subtle differences between such algorithms. It is important that the same stemmer is used when processing all artifacts.

Some of the pre-processing strategies depend upon which IR model is used to index the artifact corpus. For example, stemming is regularly an optional step while using LSI, but constantly required when using a VSM (Antoniol et al., 2002; Hayes et al., 2006; Marcus and Maletic, 2003). An alternative approach, based on searching for $n$-grams rather than stems, may be used instead. In (Hollink et al., 2004) the authors investigate the effectiveness of language-dependent (e.g., stemming) and language-independent (e.g., character $n$-gramming) approaches to cross-lingual text retrieval. They show that morphological normalization improves retrieval effectiveness, especially for languages that have a more complex morphology than English. The authors also showed that $n$-gram-base can be a viable option in the absence of linguistic resources to support a deep morphological normalization (Hollink et al., 2004). In the context of traceability link recovery the use of 2-grams to compare the content of software artifacts (phrasing) helps improving the overall recovery accuracy, especially in the top part of the ranked list (Zou et al., 2006, 2008, 2010).

The terms extracted from the documents are stored in a $m \times n$ matrix (called *term-by-document matrix* (Baeza-Yates and Ribeiro-Neto, 1999)), where $m$ is the number of all unique terms that occur within the documents, and $n$ is the number of documents in the repository. A generic entry $w_{i,j}$ of this matrix denotes a measure of the weight (i.e., relevance) of the $i$th term in the $j$th document (Baeza-Yates and Ribeiro-Neto, 1999). Various methods for weighting terms have been developed in the IR field. However, three main factors come into play in the final term weighting formulation:

1. *Term Frequency* (or *tf*): words that repeat multiple times in a document are considered salient. Term weights based on *tf* have been used in the vector space model since the 1960s.
2. *Document Frequency*: words that appear in many documents are considered common and are not very indicative of document content. A weighting method based on this, called inverse document frequency (or *idf*) weighting, was proposed by Sparck-Jones in the early 1970s (Sparck Jones, 1972).
3. *Document Length*: when collections have documents of varying lengths, longer documents tend to score higher since they contain more words and word repetitions. This effect is usually compensated by normalizing for document lengths in the term weighting method. In the context of traceability recovery, interesting results have been achieved using the pivot normalization term weighting approach that allows to specify the normalization factor depending on the specific collection of artifacts (Settimi et al., 2004).

All these factors can be taken into account while applying both a local and a global weighting to increase/decrease the importance of terms within or among documents. Specifically, a generic entry $a_{i,j}$ of the term-by-document matrix can be calculated as follows:

$$a_{i,j} = L(i,j) \cdot G(i) \tag{1}$$

where $L(i, j)$ is the local weight of the $i$th term in the $j$th document and $G(i)$ is the global weight of the $i$th term in the whole document collection. In general, the local weight increases with the frequency of the $i$th term in the $j$th document, while the global weight decreases as much as the $i$th term is spread across the documents of the document space. For example, each term can be weighted using the *tf-idf* indexing mechanism (Baeza-Yates and Ribeiro-Neto, 1999):

$$a_{i,j} = tf_{i,j} \cdot idf_i$$

where $tf_{i,j}$ and $idf_i$ are the term frequency and the inverse document frequency of the term $i$, respectively. The term frequency is computed as

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where $n_{i,j}$ represents the occurrences of term $i$ in the document $j$. The inverse document frequency is computed as

$$idf_i = log\left(\frac{n}{doc_i}\right)$$

where $doc_i$ is the number of documents where the term $i$ appears.

To better understand the tf-idf weighting schema consider an artifact containing 50 words wherein the word "doctor" appears 5 times while the word "system" appears 10 times. Following the previously defined formulas, the term frequency (*tf*) for "doctor" is $(5/50) = 0.1$, while for "system" is $(10/50) = 0.2$. Since the number of occurrences of "system" are higher than those of "doctor" the local weight of the former word is higher. Now, assume we have 100 artifacts and "doctor" appears only in 10 of these, while "system" appears in 90 artifacts. Then, the inverse document frequency for "doctor" is calculated as $log(100/10) = 1$ while for "system" is $log(100/90) = 0.05$. The $tf - idf$ score is the product of these quantities, i.e., $0.2 \cdot 0.05 = 0.01$ for "system" and $0.1 \cdot 1 = 0.1$ for "doctor". As we can see, the schema gives a higher weight to "doctor" as it is a more discriminating word compared to "system".

A more sophisticated weighting schema has been proposed by Dumais (Dumais, 1991). In this schema the local weight is represented by the term frequency scaled by a logarithmic factor, while the entropy of the term within the document collection is used for the global weight:

$$L(i,j) = log(tf_{ij} + 1) \qquad G(i) = \sum_{j=1}^{n} \frac{p_{ij} log(p_{ij})}{log(n)} \tag{2}$$

where $tf_{ij}$ is the frequency of the $i$th term in the $j$th document and $p_{ij}$ is defined as:

$$p_{ij} = \frac{tf_{ij}}{\sum_{k=1}^{n} tf_{ik}} \tag{3}$$

An advantage of using the entropy of a term to define its global weight is the fact that it takes into account the distribution of the term within the document space.

The weight of the term could also take into account the importance of the term for the specific domain. In particular, artifacts could contain critical terms and phrases that should be weighted more heavily than others, as they can be regarded as more meaningful in identifying traceability links. These terms can be extracted from the project glossary (Zou et al., 2006, 2008, 2010) or external dictionaries (Hayes et al., 2003). The importance of the terms can be derived also from the analysis of their grammatical nature (Capobianco et al., 2009a). Such an approach is based on the observation that the language used in software documents can be classified as sectorial language,[8] where the terms that provide more indication on the semantics of a document are the nouns, while the verbs tend to play a connection role and have a generic semantics (Jurafsky and Martin, 2000; Keenan, 1975). Thus, the artifact content can be pre-processed to filter out all the terms that are not nouns.

[8] The language used by people who work in a particular area or who have a common interest (Jurafsky and Martin, 2000; Keenan, 1975).

## *2.2 Corpus Indexing and Ranked List Generation*

Based on the *term-by-document matrix* representation, different IR methods can be used to rank pairs of source and target artifacts based on their similarities. A survey of available research papers reveals that probabilistic models (Abadi et al., 2008; Antoniol et al., 1999; Cleland-Huang et al., 2005), VSM (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993; Salton et al., 1975), and LSI (Deerwester et al., 1990) are the three most frequently used IR methods for traceability recovery. In particular, only in few cases different methods have been used to recover traceability links between different types of artifacts (Asuncion et al., 2010; Capobianco et al., 2009b). In (Asuncion et al., 2010) a topic modeling technique, namely Latent Dirichlet Allocation (LDA) (Blei et al., 2003) is used for traceability link recovery between text-based artifacts (such as requirements and design documents). The authors monitor the operations (e.g., opening a requirements specification or visiting a Wiki page) performed by the software engineers during software development identifying a list of potentially related artifacts. Such relationships are then used to extract a set of topics that can be subsequently used to infer other relationships between code and documentation. In (Capobianco et al., 2009b) the proposed traceability recovery method models the information contained in a software artifact by particular interpolation curves of plots mapping terms and their frequency on the artifact. Then, the similarity between artifacts is computed by calculating the distance of the corresponding interpolation curves. It is worth noting that a recent empirical study highlighted that none of these techniques sensibly outperforms the others (Oliveto et al., 2010).

The most used IR methods for traceability link recovery, i.e., the probabilistic models and the vector space-based models (VSM and LSI), are described in the following subsections. In the probabilistic model, a source artifact is ranked according to the probability of being relevant to a particular target artifact. In vector space-based models, artifacts are represented by vectors of terms. Thus, source artifacts are ranked against target artifacts by computing a distance function between the corresponding vectors.

### 2.2.1 Probabilistic Models

Three different probabilistic models have been proposed to recover links between software artifacts (Abadi et al., 2008; Antoniol et al., 1999; Cleland-Huang et al., 2005). The approaches proposed in (Antoniol et al., 1999; Cleland-Huang et al., 2005) are based on conditioned probability. In particular, this model computes the ranking scores as the probability that a document $D_i$ (target artifact) is related to the query $Q$ (source artifact):

$$sim(D_i, Q) = Pr(D_i|Q) \tag{4}$$

Applying Bayes' rule (Bain and Engelhardt, 1992), the conditioned probability above can be transformed in:

$$Pr(D_i|Q) = \frac{Pr(Q|D_i)Pr(D_i)}{Pr(Q)} \tag{5}$$

For a given query component, $Pr(Q)$ is a constant and it is possible to further simplify the model by assuming that all documents have the same probability. Therefore, for a given query $Q$, all documents $D_i$ are ranked by the conditioned probabilities $Pr(Q|D_i)$.

These conditioned probabilities are computed by estimating a stochastic language model (De Mori, 1998) for each document $D_i$. Indeed, due to the hypothesis that the query and the documents insist on the same vocabulary $V$, a query $Q$ can be represented by a sequence of $m$ words $w_1; w_2; \cdots; w_m$ (the words composing the query) of the vocabulary $V$ and the conditioned probability:

$$Pr(Q|D_i) = Pr(w_1; w_2; \cdots; w_m|D_i) \tag{6}$$

can be estimated on a statistical basis by exploiting a stochastic language model for the document $D_i$. This model collects statistics about the frequency of the occurrences of sequences of words of $V$ in $D_i$ that allow to estimate $Pr(w_1; w_2; \cdots; w_m|D_i)$ for any sequence of words $w_1; w_2; \cdots; w_m$ of $V$. However, the probability above can be written as:

$$Pr(w_1; w_2; \cdots; w_m|D_i) = Pr(w_1|D_i) \prod_{k=2}^{m} Pr(w_k|w_1; \cdots; w_{k-1}, D_i) \tag{7}$$

and when $m$ increases the conditioned probabilities involved in the above product quickly become difficult to estimate for any possible sequence of $m$ words in the vocabulary. A simplification can be introduced by conditioning the dependence of each word to the last $n-1$ words (with $n < m$):

$$Pr(w_1; w_2; \cdots; w_m|D_i) \approx \; \approx Pr(w_1; \cdots; w_{n-1}|D_i) \prod_{k=n}^{m} Pr(w_k|w_{k-n+1}; \cdots; w_{k-1}, D_i) \tag{8}$$

This $n$-gram approximation, which formally assumes a time-invariant Markov process (Cover and Thomas, 1991), greatly reduces the statistics to be collected in order to compute $Pr(Q|D_i)$. Clearly, this also introduces an imprecision. However, $n$-gram models are still difficult to estimate because, if $|V|$ is the size of the vocabulary, all possible $|V|^n$ sequences of words in the vocabulary have to be considered. Indeed, the estimation can be very demanding even for a 2-gram (bigram) model.[9] Moreover, the occurrence of any sequence of words in a document $D_i$ is a rare event, as it generally occurs only a few times and most of the sequences will never occur due to the sparseness of data. Therefore, in this approach, it is possible to considered a unigram approximation ($n = 1$) that corresponds to consider all words $w_k$ to

[9] In a bigram model, $Pr(w_1; w2; \cdots; w_m|D_i \approx Pr(w_1|D_i \prod_{k=2}^{m} Pr(w_k|w_{k-1}D_i)$.

be independent. Therefore, each document $D_i$ is represented by a language model where unigram probabilities are estimated for all words in the vocabulary and:

$$sim(D_i, Q) = Pr(Q|D_i) = Pr(w_1; w_2; \cdots; w_m|D_i \approx \prod_{k=1}^{m} Pr(w_k|D_i) \tag{9}$$

Unigram estimation is based on the term frequency of each word in a document. However, using the simple term frequency would turn the product $\prod_{k=1}^{m} Pr(w_k|D_i)$ to zero, whenever any word $w_k$ is not present in the document $D_i$. This problem, known as the zero-frequency problem (Witten and Bell, 1991), can be avoided using different approaches (see (De Mori, 1998)). A possible approach consists of smoothing the unigram probability distribution by computing the probabilities as follows (Antoniol et al., 2002):

$$Pr(w_k|D_i) \tag{10}$$

where $N$ is the total number of words in the document $D_i$ and $c_k$ is the number of occurrences of words $w_k$ in the document Di. The interpolation term is:

$$\lambda = \frac{n}{(N * |V|)}\beta \tag{11}$$

where $n$ is the number of different words of the vocabulary $V$ occurring in the document $D_i$. The value of the parameter is computed according to Ney and Essen (1991) as follows:

$$\beta = \frac{n(1)}{(n(1) + 2 * n(2))} \tag{12}$$

where $n(j)$ is the number of words occurring $j$ times in the document $D_i$.

In the context of traceability link recovery, the probabilistic model based on conditioned probability has been used to recover links among requirements (Gibiec et al., 2010; Cleland-Huang et al., 2010), requirements and UML diagrams (Cleland-Huang et al., 2005), requirements and source code (Abadi et al., 2008; Antoniol et al., 2000a, 2000b, 2000c, 2002; Di Penta et al., 2002), and manual pages and source code (Antoniol et al., 1999, 2000a). In these studies different enhancing strategies have been also proposed in order to improve recovery accuracy. The first strategy exploits partial knowledge of a subset of traceability links (Antoniol et al., 2000b). In particular, a set of previously identified links can be supplied to the probabilistic network to improve the recovery accuracy.

Knowledge about the structure of the artifacts can also be exploited to improve the performances of a probabilistic recovery method. In particular, different enhancing strategies, namely hierarchical modeling, logical clustering of artifacts, and semi-automated pruning of the probabilistic network, have been proposed to incorporate supporting information into a probabilistic retrieval algorithm and improve the retrieval accuracy (Cleland-Huang et al., 2005). The first enhancing strategy (that is, hierarchical modeling) is based on the observation that artifacts are generally arranged in a hierarchical format. This hierarchical structure could be exploited to better identify the context of each artifacts, since, in general, the words used to name and describe the higher level artifacts capture the meaning (i.e., context) of their lower-level components. The hierarchical links are then exploited when computing the probabilities that a target artifact is relevant for a given source artifact. The second strategy is based on the conjecture that links tend to occur in clusters. Thus, if a link exists between a source artifact and a target artifact, and if that target artifact is a part of a logical cluster of artifacts, then there would be a higher probability that additional links should exist between the same source artifact and other target artifacts in the cluster. It is worth noting that this approach differs from the previous ones because the enhancement is based on sibling artifacts rather than on ancestral information. The last enhancing strategy, i.e., pruning of the probabilistic network, aims at attenuating the synonym problem. In particular, a set of constraints can be automatically added to the probabilistic network exploiting previous identified links (used as training set). The proposed enhancing strategies are able to improve the recovery accuracy of a canonical probabilistic recovery method. An analysis of the improvements provided by each enhancement strategy indicates significant overlap between the hierarchical and clustering techniques. Instead, the pruning technique seems to improve different aspects of the probabilistic model. In particular, while the first two approaches tend to generally improve the precision, the third strategies tends to specifically attenuate a particular problem, i.e., the synonym problem (Cleland-Huang et al., 2005). Such a result suggests that the pruning technique could be combined with hierarchical modeling or logical clustering of artifacts.

Recently, another enhancement strategy has been proposed to improve the recovery accuracy of probabilistic retrieval methods. Such a strategy, called Query Term Coverage, increases the relevance ranking of links between artifacts that have more than one unique word in common (Zou et al., 2007, 2010). This approach tends to reduce the number of incorrectly retrieved links between unrelated pairs of artifacts that contain only a single matching word, which co-occurs multiple times.

Another probabilistic model has been proposed by Abadi et al. (2008). The proposed approach, referred as JS model, is driven by a probabilistic approach and hypothesis testing techniques. As well as other probabilistic models, it represents each document through a probability distribution. This means that an artifact is represented by a random variable where the probability of its states is given by the empirical distribution of the terms occurring in the artifact (i.e., columns of the term-by-document matrix). It is worth noting that the empirical distribution of a term is based on the weight assigned to the term for the specific artifact (Abadi et al., 2008). In the JS method the similarity between two artifacts is given by a "distance" of their probability distributions measured by using the Jensen-Shannon (JS) Divergence (Cover and Thomas, 1991).

### 2.2.2 Vector Space-Based Models

In the VSM, an artifact is represented by a vector of terms (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993; Salton et al., 1975). The definition of a term is not inherent in the model, but terms in the context of traceability recovery are typically words. Note that, for example, identifiers are often made of words that are not from a natural language, yet they are considered valid in his context. If words are chosen as terms, then every word in the vocabulary becomes an independent dimension in a very high dimensional vector space. Since any artifact contains a limited set of terms (the vocabulary can be millions of terms), most artifact vectors are very sparse and they generally operate in a positive quadrant of the vector space, i.e., no term is assigned a negative value.

To assign a numeric score to a document (target artifact) for a query (source artifact), the model measures the similarity between the query vector and the document vector. The similarity between two vectors is once again not inherent in the model. Typically, the angle between two vectors is used as a measure of divergence between the vectors, and the cosine of the angle is used as the numeric similarity.[10] If $\overrightarrow{D}$ is the document vector and $\overrightarrow{Q}$ is the query vector, then the similarity of document to query (or score of for) can be calculated as follows (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993):

$$sim(D,Q) = \frac{\overrightarrow{D} \cdot \overrightarrow{Q}}{\| \overrightarrow{D} \| \cdot \| \overrightarrow{Q} \|} = \frac{\sum_{t_i \in D,Q} w_{t_iD} \cdot w_{t_iQ}}{\sqrt{\sum_{t_i \in D} w_{t_iD}^2} \cdot \sqrt{\sum_{t_i \in Q} w_{t_iQ}^2}} \tag{13}$$

where $w_{t_iQ}$ is the value of the $i$th component in the query vector $\overrightarrow{Q}$, and $w_{t_iD}$ is the $i$th component in the document vector $\overrightarrow{D}$. Since any word not present in either the query or the document has a $w_{t_iQ}$ or $w_{t_iD}$ equals to 0, respectively, it is possible to sum only over the terms common in the query and the document. As an alternative, the inner-product (or dot-product) between two vectors is often used as a similarity measure. If all the vectors are forced to be unit length, then the cosine of the angle between two vectors is the same as their dot-product (Baeza-Yates and Ribeiro-Neto, 1999; Harman, 1993).

In the context of traceability recovery, the VSM has been used to recover traceability links among requirements (De Lucia et al., 2006b; Hayes et al., 2003, 2006), requirements and source code (Abadi et al., 2008, Antoniol et al., 2000a, 2002, De Lucia et al., 2006b; Marcus and Maletic, 2003; Marcus et al., 2005), manual pages and source code (Antoniol et al., 2000a, 2002; Marcus and Maletic, 2003, Marcus et al., 2005), UML diagrams and source code (De Lucia et al., 2006b, Settimi et al., 2004), test cases and source code (De Lucia et al., 2006b), and defect reports and source code (Yadla et al., 2005).

A common criticism of VSM is that it does not take into account relations between terms (Deerwester et al., 1990). For instance, having an “automobile” in one document and a “car” in another document does not contribute to the similarity measure between these two documents. LSI (Deerwester et al., 1990) was developed to overcome the synonymy and polysemy problems, which occur with the

[10] The cosine has a property indicating 1.0 for identical vectors and 0.0 for orthogonal vectors.

VSM model. In LSI the dependencies between terms and documents, in addition to the associations between terms and documents, are explicitly taken into account. LSI assumes that there is an underlying or "latent structure" in word usage that is partially obscured by variability in word choice, and uses statistical techniques to estimate this latent structure. For example, both "car" and "automobile" are likely to co-occur in different documents with related terms, such as "motor", "wheel", etc. LSI exploits information about co-occurrence of terms (i.e., latent structure) to automatically discover synonymy between different terms.

LSI defines a term-by-document matrix $A$ as well as VSM. Then it applies the Singular Value Decomposition (SVD) (Cullum and Willoughby, 1998) to decompose the term-by-document matrix into the product of three other matrices:

$$A = T_0 \cdot S_0 \cdot D_0 \tag{14}$$

where $T_0$ is the $m \times r$ matrix of the terms containing the left singular vectors (rows of the matrix), $D_0$ is the $r \times n$ matrix of the documents containing the right singular vectors (columns of the matrix), $S_0$ is an $r \times r$ diagonal matrix of singular values, and $r$ is the rank of $A$. $T_0$ and $D_0$ have orthogonal columns, such that:

$$T_0^T \cdot T_0 = D_0^T \cdot D_0 = I_r \tag{15}$$

SVD can be viewed as a technique for deriving a set of uncorrelated indexing factors or concepts (Deerwester et al., 1990), whose number is given by the rank $r$ of the matrix $A$ and whose relevance is given by the singular values in the matrix $S_0$. Concepts "represent extracted common meaning components of many different words and documents" (Deerwester et al., 1990). In other words, concepts are a way to cluster related terms with respect to documents and related documents with respect to terms. Each term and document is represented by a vector in the $r$-space of concepts, using elements of the left or right singular vectors. The product $S_0 \cdot D_0$ ($T_0 \cdot S_0$, respectively) is a matrix whose columns (rows, respectively) are the document vectors (term vectors, respectively) in the $r$-space of the concepts. The cosine of the angle between two vectors in this space represents the similarity of the two documents (terms, respectively) with respect to the concepts they share. In this way, SVD captures the underlying structure in the association of terms and documents. Terms that occur in similar documents, for example, will be near each other in the $r$-space of concepts, even if they never co-occur in the same document. This also means that some documents that do not share any word, but share similar words may nonetheless be near in the $r$-space.

SVD allows a simple strategy for optimal approximate fit using smaller matrices (Deerwester et al., 1990). If the singular values in $S_0$ are ordered by size, the first $k$ largest values may be kept and the remaining smaller ones set to zero. Since zeros were introduced into $S_0$, the representation can be simplified by deleting the zero rows and columns of $S_0$ to obtain a new diagonal matrix $S$, and deleting the corresponding columns of $T_0$ and rows of $D_0$ to obtain $T$ and $D$ respectively. The result is a reduced model:

$$A \approx A_k = T \cdot S \cdot D \tag{16}$$

where the matrix $A_k$ is only approximately equal to $A$ and is of rank $k < r$. The truncated SVD captures most of the important underlying structure in the association of terms and documents, yet at the same time it removes the noise or variability in word usage that plagues word-based retrieval methods. Intuitively, since the number of dimensions $k$ is much smaller than the number of unique terms $m$, minor differences in terminology will be ignored.

The choice of $k$ is critical: ideally, it is desirable to have a value of $k$ that is large enough to fit all the real structure in the data, but small enough not to fit the sampling error or unimportant details. The proper way to make such a choice is an open issue in the factor analysis literature (Deerwester et al., 1990; Dumais, 1991). In the application of LSI to information retrieval, good performances have been achieved using about 100 concepts on a document space of about 1,000 documents and a vocabulary of about 6,000 terms (Deerwester et al., 1990). With much larger repositories (between 20,000 and 220,000 documents and between 40,000 and 80,000 terms), good results have been achieved using between 235 and 250 concepts (Dumais, 1991; Poshyvanyk et al., 2007; Revelle et al., 2010). It is worth noting that software repositories are not comparable with document collection generally used in the text retrieval field. This implies that in the context of traceability recovery the size of the LSI subspace does not significantly influence the recovery accuracy (De Lucia et al., 2007) and values between 100 and 250 provides acceptable results (De Lucia et al., 2007; Hayes et al., 2006; Marcus and Maletic, 2003). Nonetheless, the choice of $k$ in this application is still an open issue.

In the context of traceability recovery, LSI has been used to recover traceability links between requirements (De Lucia et al., 2006b; Hayes et al., 2006), requirements and source code (Abadi et al., 2008; De Lucia et al., 2004; De Lucia et al., 2006a, 2006b, 2007; Marcus and Maletic, 2003; Marcus et al., 2005), manual pages and source code (Antoniol et al., 2000a, 2002; Marcus and Maletic, 2003; Marcus et al., 2005), UML diagrams and source code (De Lucia et al., 2004; 2006a, 2006b, 2007, Settimi et al., 2004), test cases and source code (De Lucia et al., 2004; De Lucia et al., 2006a, 2006b, 2007; Lormans and Van Deursen, 2005, 2006; Lormans et al., 2006, 2008).

## 3 Measuring the Performance of IR-Based Traceability Recovery Methods

The performances–in terms of retrieval accuracy–of an IR-based traceability recovery method is generally evaluated through a set of retrieved links over a set of relevant links. The set of retrieved links is obtained by cutting the ranked list provided by an IR method at a given point and considering only the top links in the ranked list. The set of relevant links is generally derived by a traceability matrix provided by original developers of the system at the end of the process (this matrix is intended to contain the correct links). Clearly, the set of retrieved traceability links, in general, does not match exactly the set of relevant links between the artifacts in the repository. In fact, any IR methods will fail to retrieve some of the relevant links while, on the other hand, it will also retrieve links that are not relevant (false positives). This is one of the key reasons why IR-based traceability recovery methods are semi-automatic and require some degree of interaction between software developers and a traceability link recovery tool.

By and large, the retrieval performance of IR methods is measured using two metrics, namely recall and precision (Baeza-Yates and Ribeiro-Neto, 1999). Recall is the ratio between the number of links that are successfully retrieved and the number of links that are relevant (Baeza-Yates and Ribeiro-Neto, 1999):

$$recall = \frac{|\{\text{relevant\_links}\} \cap \{\text{retrieved\_links}\}|}{|\{\text{relevant\_links}\}|} \tag{17}$$

It is easy to note that it is trivial to achieve 100% of recall by returning all links in response to any query. Therefore, recall alone, is not enough, but one needs to measure the number of non-relevant links as well. Precision is the fraction of the links retrieved that are relevant to the source artifact (Baeza-Yates and Ribeiro-Neto, 1999):

$$precision = \frac{|\{\text{relevant\_links}\} \cap \{\text{retrieved\_links}\}|}{|\{\text{retrieved\_links}\}|} \tag{18}$$

Differently from the recall, precision takes all retrieved links into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called *precision at n* or *P@n*. Note that the meaning and usage of "precision" in the field of IR differs from the definition of accuracy and precision within other branches of science and technology. In particular, in the fields of science, engineering, industry and statistics, accuracy is a degree of conformity of a measured or calculated quantity to its actual (true) value, while precision, also called reproducibility or repeatability, is the degree to which further measurements or calculations show the same or similar results (Baeza-Yates and Ribeiro-Neto, 1999).

Both measures have values in the interval of [0, 1]. If the recall value is 1, it means that all relevant links have been recovered, though there could be recovered links that are not relevant. If the precision is 1, it implies that all recovered links are relevant, though there could be relevant links that were not recovered. In general, retrieving a lower number of links results in higher precision, while a higher number of retrieved links increases the recall.

The recall and precision are orthogonal metrics and measure two different concepts. Often, an aggregate measure, namely F-measure, is used to obtain a balance between them. The F-measure or balanced F-score is the weighted harmonic mean of precision and recall:

$$F = 2 \cdot \frac{(\text{precision} \cdot \text{recall})}{(\text{precision} + \text{recall})} \tag{19}$$

This is also known as the $F_1$ measure, because recall and precision are evenly weighted. The general formula for non-negative real $\alpha$ is:

$$F_\alpha = (1 + \alpha) \cdot \frac{(\text{precision} \cdot \text{recall})}{(\alpha \cdot \text{precision} + \text{recall})} \tag{20}$$

Two other commonly used F-measures are the $F_2$ measure, which weights recall twice as much as precision, and the $F_{0.5}$ measure, which weights precision twice as much as recall.

Another metric used to measure the performances of an IR-based traceability recovery method is represented by average precision that is defined as the mean of the precision scores obtained after each correct link is retrieved, using zero as the precision for correct links that are not retrieved (Cleland-Huang et al., 2010; Gibiec et al., 2010).

The main role of IR tools consists of reducing the document space, while recovering all the relevant links between artifacts. Without tool support, one must analyze all artifacts in order to identify the dependencies between them. With a reduced document space the number of artifacts to analyze is generally much smaller. This means that high recall values (possibly 100%) should be pursued. Of course, in this case higher precision values reduce the effort required to discard false positives (documents that are retrieved, but not relevant to a given query).

To achieve an indication of the benefits of using an IR approach in a traceability link recovery process, it is possible to use the Recovery Effort Index (REI), defined as the ratio between the number of documents retrieved and the total number of documents available (Antoniol et al., 2002):

$$REI_i = \frac{|\text{retrieved\_artifacts}|}{|\text{target\_artifacts}|}\%$$

This metric can be used to estimate the percentage of the effort required to manually analyze the results achieved by an IR tool (and discard false positive), when the recall is 100%, with respect to a completely manual analysis. For a given software system, the quantity 1 - REI can be used to estimate the effort savings due to the use of an IR method to recover traceability links, with respect a completely manual analysis (Antoniol et al., 2002). The lower the REI, the higher the benefits of the IR approach.

## 4 Analysis of Candidate Links

The similarity measures obtained by applying a particular IR method are used to build a ranked list of candidate links. This list contains all possible pairs of source and target artifacts ranked according to textual similarities among source and target artifacts. This means that the list of candidate links inevitably contains links that are not correct and have to be discarded by the software engineer. Two different approaches have been proposed to analyze the ranked list. The first approach is based on the analysis of the full ranked list of candidate links (Capobianco et al., 2009b; Cleland-Huang et al., 2005; De Lucia et al., 2006b, 2007; Hayes et al., 2003; Yadla et al., 2005). However, the list of candidate links contains a higher density of correct traceability links in the upper part of the list and a much lower density of such links in the bottom part of the list (De Lucia et al., 2007, 2009a). This means that in the lower part of the ranked list the effort required to discard false positives becomes much higher than the effort to validate correct links.

The above considerations suggest the use of some method to cut the ranked list (e.g., a threshold on the similarity value), thus presenting the software engineer only the subset of top links in the ranked list (ranked list filtering) (Abadi et al., 2008; Antoniol et al., 2002; De Lucia et al., 2007; Hayes et al., 2006; Lormans et al., 2008; Marcus and Maletic, 2003; Settimi et al., 2004; Zou et al., 2007). Different strategies have been proposed to filter ranked lists. Broadly, these methods can be classified into *cut-point* and *threshold* based strategies. The first category of methods cut the ranked list regardless of the values of the similarity measure (cut point based strategy):

1. *Constant cut point*: this method consists of imposing a threshold on the number of recovered links (Antoniol et al., 2002; Marcus and Maletic, 2003). In this way, the top $\mu$ links of the ranked list are selected.
2. *Variable cut point*: this is an extension of the previous method that consists of specifying the percentage of the links of the ranked list that have to be retrieved (cut percentage). In this way the cut point depends on the size of the ranked list.

The second category (threshold based strategy) use a threshold $\varepsilon$ on a similarity measure and only the pairs of artifacts having a similarity measure greater than or equal to $\varepsilon$ will be retrieved:

1. *Constant threshold*: this is the standard method used in the literature. A widely adopted threshold is $\varepsilon = 0.70$, that for the vector space model (and LSI) approximately corresponds to a 45° angle between the corresponding vectors (Marcus and Maletic, 2003).
2. *Scale threshold*: a threshold $\varepsilon$ is computed as the percentage of the best similarity value between two artifacts, i.e., $\varepsilon = c \cdot \mathit{MaxSimilarity}$, where $0 \leq c \leq 1$ (Antoniol et al., 2002). In this case, the higher the value of the parameter $c$, the smaller the set of links returned by a query.
3. *Variable threshold*: this is an extension of the constant threshold approach. The constant threshold is projected from the interval [0, 1] into the interval [min similarity, max similarity], where min similarity and max similarity are the minimum and maximum similarity values in the ranked list (De Lucia et al., 2004, 2006a, 2007).

It is worth noting that the lower the similarity threshold used, the higher the number of correct links as well as the number of false positives retrieved and the relative effort to discard them. Such a limitation of IR-based traceability recovery methods have encouraged researchers to identify an "optimal" threshold enabling the retrieval of as many correct links as possible, while keeping low the effort required to analyze and discard false positives. However, this ideal threshold is not easy to identify, as it can change together with the type of artifacts and projects (De Lucia et al., 2007). For this reason, an incremental traceability recovery process has been proposed (De Lucia et al., 2007) where the similarity threshold is incrementally decreased to provide the software engineer with the control on the number of correct links validated and false positives discarded at each iteration. In this way, starting with a high threshold, the links suggested by the tool can be analyzed and classified step-by-step and the process can be stopped when the effort of discarding false positives is becoming much higher than the effort of identifying new correct links. Clearly, the traceability recovery tool maintains knowledge about the classification actions performed by the software engineer, thus showing at each iteration only the new traceability links retrieved.

User studies highlighted that the threshold used to stop the recovery process plays an important role (De Lucia et al., 2009a). In particular, when the software engineer stops the traceability recovery process using a low threshold, she increases the number of correct links compared to software engineers stopping the process with a higher threshold. This situation suggests that a higher number of correct links could be traced by providing the software engineer with the full ranked list of possible links ordered by decreasing similarity values (full ranked list contrasted to the incremental approach). A recent study compared the two approaches through a controlled experiment (De Lucia et al., 2008). The achieved results demonstrated that the number of correct links retrieved with two different processes is comparable and that the incremental process significantly reduces tracing errors. Moreover, the number of links analyzed showing the full ranked list is significantly larger than the number of links analyzed adopting the incremental process. All these results suggest that the incremental process reduces the effort required to identify traceability links using an IR-based traceability recovery tool (De Lucia et al., 2008). This means that analyzing the lower part of the ranked list only results in a small improvement of the number of correct links retrieved. Indeed, while in the upper part of the ranked list the density of correct links is quite good, in the bottom part of the list such a density decreases in a way that the prioritization made by the IR method does not help anymore.

The use of the link coverage analysis has been proposed to enrich the set of correct links identified with one of the approaches described above (De Lucia et al., 2009b). In particular, after preliminary traceability link recovery sessions performed using canonical approaches, the software engineer can exploit the information provided by a link coverage analysis to identify source artifacts poorly traced on the set of target artifacts. In particular, for a generic artifact $a$, it is possible to define a traceability coverage index as follow:

$$traceabilityCoverage_a = \frac{|links_a(targets)|}{|targets|}$$

where *targets* represents the set of target artifacts and $links_a(targets)$ represents the set of links traced between the artifact *a* and the artifacts in the set *target*. Then, all the source artifacts are ranked (in an increasing order) according to their traceability coverage index.

The ranked list of source artifacts computed according to the link coverage analysis can be exploited to guide traceability recovery sessions focusing attention only on source artifacts with a low traceability coverage index. The conjecture is that the probability of identifying new correct links is higher when focusing on poorly traced artifacts. It is worth noting that after the identification of poorly traced artifacts the software engineer can recover the links using one of the approaches described above.

Once a traceability recovery tool produces a ranked list of candidate links, developers need to examine all these suggestions starting with the documents having highest similarity values. For every candidate link, a decision is required as to whether the link is a correct traceability link between two artifacts. If it is not the correct link, the user discards that and proceeds to the next suggestion in the ranked list. The process is continued until all the links in the ranked list are examined. The size of the ranked list is determined according to one of the strategies (threshold, cut point or combination) described above. The classification performed by the software engineer can be provided to the tool as a feedback for improving tracing performances (Antoniol et al., 2000c; Di Penta et al., 2002; De Lucia et al., 2006b; Hayes et al., 2003, 2006). In general, the user is asked to judge the relevance of the top few links retrieved by the system. If the user judges a retrieved link as correct, different strategies can be used to alter the source artifact in order to "move" it towards relevant artifacts and away from irrelevant artifacts, in the expectation of retrieving more relevant links and less irrelevant links in next iterations. It is worth noting that the first adjustment is designed to potentially increase the recall, while the second adjustment can potentially increase the precision.

## 5 Trace Retrieval in Action: Recovering Traceability Links in the iTrust System

This section describes the application of different IR methods to recover links between software artifacts of the iTrust system. iTrust is a medical application that provides patients with a means to keep up with their medical history and records as well as communicate with their doctors, including selecting which doctors to be their primary caregiver, seeing and sharing satisfaction results, and other tasks. iTrust is also an interface for medical staff from various locations. It allows the staff to keep track of their patients through messaging capabilities, scheduling of office visits, diagnoses, prescribing medication, ordering and viewing lab results, among other functions. The iTrust artifacts include use cases, source code, test cases, and trace matrices. The source code of the iTrust project is available[11] on sourceforge and is accessible through the project's webpage.

The goals of this empirical study are:

- providing empirical evidence of the accuracy of IR methods when used to recover links between software artifacts;
- analyzing the accuracy improvement achieved by performing a morphological analysis on the software artifacts;
- comparing the accuracy of different IR methods.

In the context of the study we employed three widely used IR methods for traceability recovery, namely JS (a probabistic model), VSM (a vector space based model), and LSI (a space reduction based model). These methods have been applied to recover links between 33 use cases and 47 JSP pages. We focus on this type of artifacts since the trace matrix of the latest version of the system only include mapping between these types of artifacts.[12] The matrix is used as an oracle to evaluate the accuracy of the employed IR-based recovery methods.

For each method, the term-by-document matrix is extracted following the steps of the process described in Section 2.1, i.e., term normalization, identifier splitting, and term filtering and weighting. As for the splitting of composite identifiers, a camel case splitting heuristic was used. For the term filtering process, a canonical English stop word list[13] augmented with HTML keywords is used in combination with a stop list function aimed at pruning out all the terms with a length less than 3. Finally, a *tf-idf* weighting schema is applied on the term-by-document matrix.

To evaluate the accuracy of each recovery method for each traceability recovery activity we automatically collected the number of correct links and false positives by using a tool that takes as an input the ranked list of candidate links produced by the IR method (e.g., VSM) and classifies each link as correct link or false positive until all the correct links in the original traceability matrix have been recovered.

A first analysis and comparison of the different IR methods is performed by recall and precision. Figures 1, 2, and 3 show the precision/recall curves achieved by using the employed IR methods with and without the use of stemming. As we can see, while the precision of all the methods are acceptable for lower values of recall (especially for the JS method), when the goal is to recover all the correct links (100% of recall) the list of candidate links contains a huge number of false positive resulting in a very low precision (lower than 10%). As mentioned before, this is one of the main limitation of IR-based traceability recovery methods. Unfortunately, such a limitation cannot be completely mitigated by using enhancing strategies, such

[11] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php

[12] http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=tracing

[13] http://www.ranks.nl/resources/stopwords.html

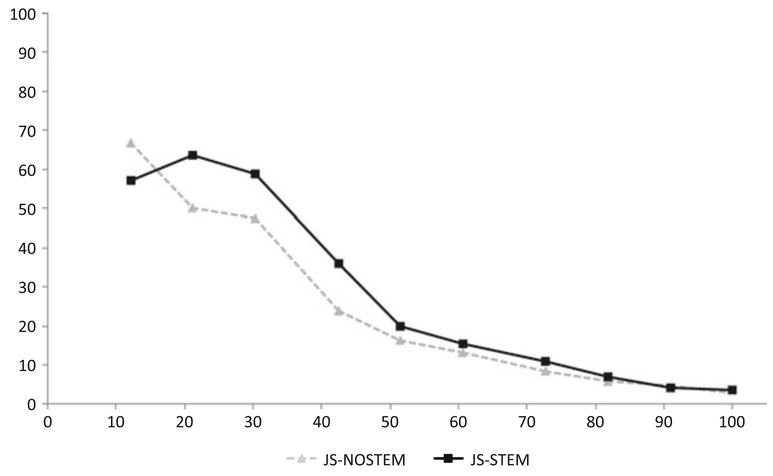

**Fig. 1** Using JS to recover links between JSP pages and use cases of iTrust. Precision – vertical axis. Recall – horizontal axis

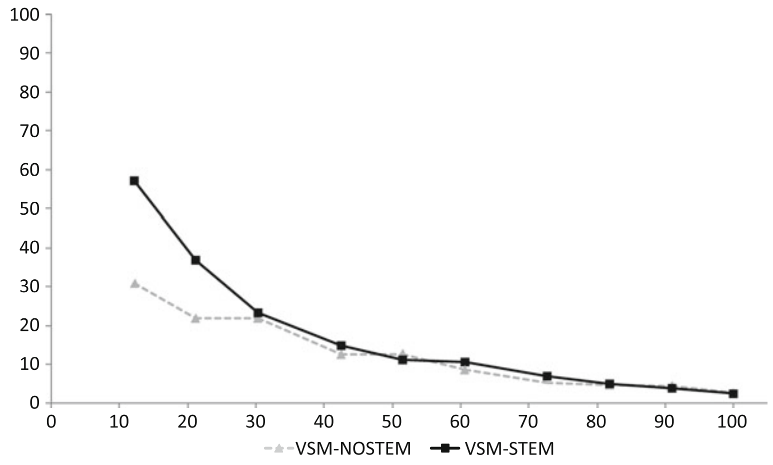

**Fig. 2** Using VSM to recover links between JSP pages and use cases of iTrust. Precision – vertical axis. Recall – horizontal axis

as stemming and user feedback analysis (De Lucia et al., 2006b). In our experimentation, it looks like there is an upper bound to the performance improvements that cannot be overcome, even if an advanced artifact pre-processing, like stemming, is used. This means that IR-based traceability recovery tools should be used to trace as many as possible correct links keeping low the effort to discard false positives. Then, focused traceability recovery sessions should be performed to identify links between untraced artifacts (De Lucia et al., 2009b). Alternatively, manual tracing activities could be conducted to enrich the set of traced links.

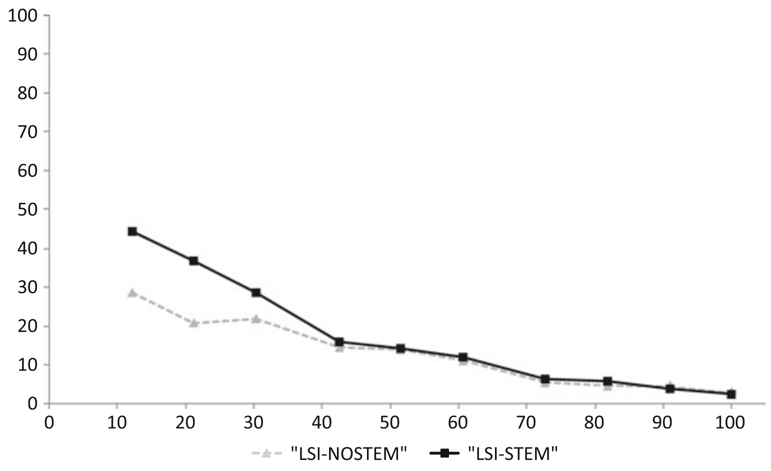

**Fig. 3** Using LSI to recover links between JSP pages and use cases of iTrust. Precision – vertical axis. Recall – horizontal axis

Regarding the comparison between the employed IR methods, Fig. 4 shows the average precision achieved by all the methods with and without the use of stemming. As we can seen, the precision of the JS method is better than the precision of both VSM and LSI by about 10% on average. A more detailed comparison of the employed method can be obtained through recall/precision curves. Figures 5 and 6 show the comparison of JS, VSM, and LSI with and without the use of stemming, respectively, through precision/recall curves. As we can see the three

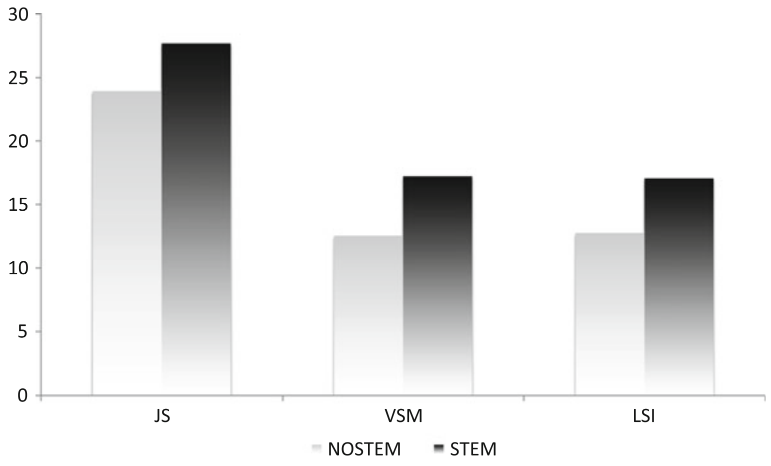

**Fig. 4** Comparison of JS, VSM, and LSI using average precision

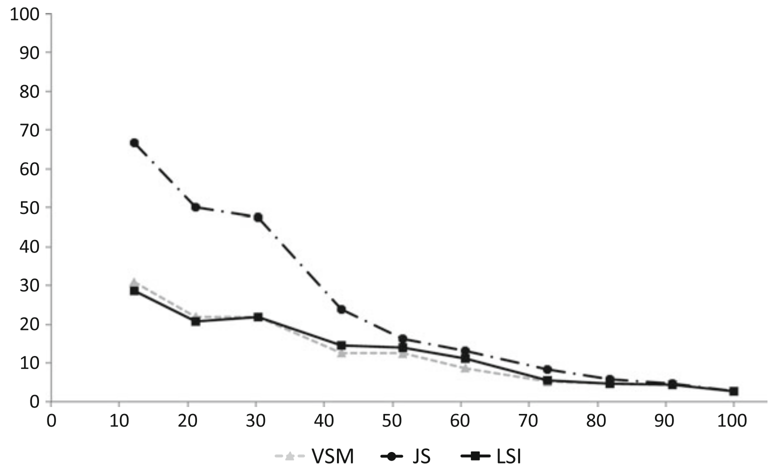

**Fig. 5** Comparison of JS, VSM, and LSI without stemming tracing use cases onto JSP pages of iTrust. Precision – vertical axis. Recall – horizontal axis

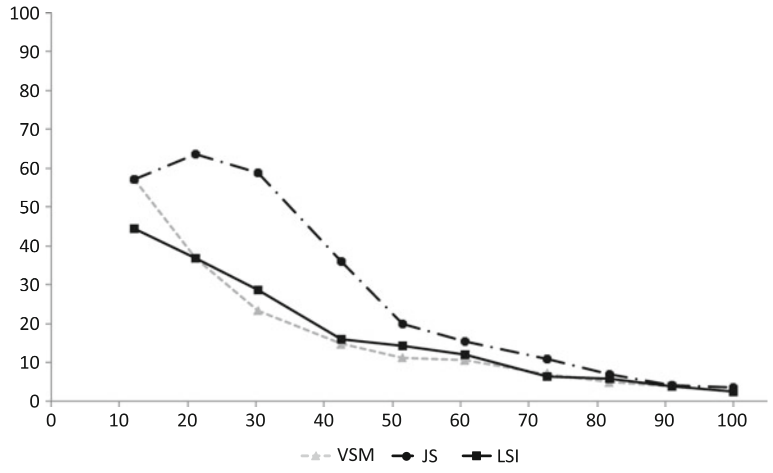

**Fig. 6** Comparison of JS, VSM, and LSI with stemming tracing use cases onto JSP pages of iTrust. Precision – vertical axis. Recall – horizontal axis

IR methods exhibit almost the same behavior in the two scenarios, i.e., with and without stemming. The JS method seems to be the more accurate method as compared to VSM and LSI (this confirm the differences in terms of average precision) on this particular data set. It is impossible to generalize this result to other data sets. Interestingly, when recall is higher than 50% the three methods provide almost the same precision.

## 6 Conclusion

Traceability links between software artifacts represent an important source of information, if available, for different stakeholders, e.g., project managers, analysts, designers, maintainers, and end users, and provides important insights during different phases of software development. Traceability information can also be used when certifying a safety-critical product to show that all requirements were implemented and covered by specific tests.

Unfortunately, establishing and maintaining traceability links between software artifacts is a time consuming, error prone, and person-power intensive task (Ramesh and Jarke, 2001). Consequently, despite the advantages that can be gained, explicit traceability is rarely established unless there is a regulatory reason for doing so. Extensive effort in the software engineering community (both research and commercial) has been brought forth to improve the explicit connection of software artifacts. Promising results have been achieved using Information Retrieval (IR) techniques (Baeza-Yates and Ribeiro-Neto, 1999; Deerwester et al., 1990) to recover links between different types of artifacts (see e.g., (Antoniol et al., 2002; De Lucia et al., 2007; Hayes et al., 2006; Marcus et al., 2005)). IR-based methods propose a list of candidate traceability links on the basis of the similarity between the text contained in the software artifacts. The conjecture is that two artifacts having high textual similarity share similar concepts, thus they are good candidates to be traced on each other.

This chapter presented a general process of using IR-based methods for traceability link recovery and presented some of them in a greater detail: probabilistic, vector space, and Latent Semantic Indexing models. Common approaches to measuring the performance of IR-based traceability methods as well as the latest advances in techniques for analysis of candidate links were also analyzed and presented. An example on using three IR models to retrieve traceability links between artifacts of the iTrust system was also presented.

IR based techniques for traceability link recovery among artifacts proved to be quite successful so far, but the current research and applications also revealed many areas where improvement is needed and expected. Virtually, all IR techniques include a series of parameters that influence their performance, as presented in this chapter. The optimal values for these parameters are usually derived based on the data model at hand. The IR techniques most commonly used in traceability link recovery operate with parameters established in natural text retrieval applications. As mentioned before, the textual data in software artifacts is not the same as in natural text documents. More than that, there is little evidence that the data in one software systems has the same characteristics as in another. Future work will have to address a generic model that can be used for parameter tuning in IR based traceability link recovery application. There is one precondition to this future work: the creation and dissemination of benchmarks for software artifact traceability link recovery. These are not only necessary for parameter tuning, but they are also essential for the investigation of other IR techniques to be used in such applications. Based on current research, it is unclear which IR technique is best suited for this task and whether that depends on the specifics of the software or not. Benchmarks will help us answer this question. They will also help in the development of new techniques that address traceability link recovery. Such techniques should improve on the state of the art results, as established using the future benchmarks.

**Acknowledgments** We would like to thank the anonymous reviewers for their detailed, constructive, and thoughtful comments that helped us to improve the presentation of the results in this chapter.

## References

Abadi, A., Nisenson, M., Simionovici, Y.: A traceability technique for specifications. In: Proceedings of 16th IEEE International Conference on Program Comprehension, pp. 103–112. IEEE CS Press, Amsterdam, The Netherlands (2008)

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A.: Information retrieval models for recovering traceability links between code and documentation. In: Proceedings of 16th IEEE International Conference on SoftwareMaintenance, pp. 40–51. IEEE CS Press, San Jose, CA (2000a)

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Tracing object-oriented code into functional requirements. In: Proceedings of 8th IEEE International Workshop on Program Comprehension, pp. 79–87. IEEE CS Press, Limerick, Ireland (2000b)

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E.: Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. **28**(10), 970–983 (2002)

Antoniol, G., Canfora, G., De Lucia, A., Merlo, E.: Recovering code to documentation links in OO systems. In: Proceedings of 6th Working Conference on Reverse Engineering, pp. 136–144. IEEE CS Press, Atlanta, GA (1999)

Antoniol, G., Casazza, G., Cimitile, A.: Traceability recovery by modelling programmer behaviour. In: Proceedings of 7th Working Conference on Reverse Engineering, vol. 240–247. IEEE CS Press, Brisbane, QLD (2000c)

Antoniol, G., Guéhéneuc, Y.-G., Merlo, E., Tonella, P.: Mining the Lexicon used by programmers during sofware evolution. In: Proceedings of the 23rd IEEE International Conference on Software Maintenance, pp. 14–23. IEEE Press, Paris, France (2007)

Asuncion, Hazeline U., Asuncion, A., Taylor, Richard N.: Software traceability with topic modeling. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, pp. 95–104. ACM Press, Cape Town, South Africa (2010)

Bacchelli, A., Lanza, M., Robbes, R.: Linking e-mails and source code artifacts. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, vol. 1, pp. 375–384. ICSE, Cape Town, South Africa (2010)

Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley, Reading, MA (1999)

Bain, L., Engelhardt, M.: Introduction to Probability and Mathematical Statistics. Duxbury Press, Pacific Grove, CA (1992)

Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent dirichlet allocation. J. Mach. Learn. Res. **3**, 993–1022 (2003)

Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., Panichella, S.: On the role of the nouns in IR-based traceability recovery. In: Proceedings of 17th IEEE International Conference on Program Comprehension. Vancouver, British Columbia, Canada (2009a)

Capobianco, G., De Lucia, A., Oliveto, R., Panichella, A., Panichella, S.: Traceability recovery using numerical analysis. In: Proceedings of 16th Working Conference on Reverse Engineering. IEEE CS Press, Lille, France (2009b)

Cleland-Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A machine learning approach for tracing regulatory codes to product specific requirements. In: Proceedings of the 32nd

ACM/IEEE International Conference on Software Engineering, pp. 155–164. ICSE, Cape Town, South Africa (2010)

Cleland-Huang, J., Settimi, R., Duan, C., Zou, X.: Utilizing supporting evidence to improve dynamic requirements traceability. In: Proceedings of 13th IEEE International Requirements Engineering Conference, pp. 135–144. IEEE CS Press, Paris, France (2005)

Cover, T.M., Thomas, J.A.: Elements of Information Theory. Wiley-Interscience, New York, NY (1991)

Cullum, J.K., Willoughby, R.A.: Lanczos Algorithms for Large Symmetric Eigenvalue Computations, vol. 1, chapter Real rectangular matrices. Birkhauser, Boston, MA (1998)

De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Enhancing an Artifact management system with traceability recovery features. In: Proceedings of 20th IEEE International Conference on Software Maintenance, pp. 306–315. IEEE CS Press, Chicago, IL (2004)

De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Can information retrieval effectively support traceability link recovery? In: Proceedings of 14th IEEE International Conference on Program Comprehension, pp. 307–316. IEEE CS Press, Athens, Greece (2006a)

De Lucia, A., Fasano, F., Oliveto, R., Tortora, G.: Recovering traceability link in software Artifacts management systems using information retrieval methods. ACM Trans. Softw. Eng. Methodol. **16**(4), Article 13 (2007)

De Lucia, A., Oliveto, R., Sgueglia, P.: Incremental approach and user feedbacks: A Silver Bullet for traceability recovery. In: Proceedings of 22nd IEEE International Conference on Software Maintenance, pp. 299–309. Sheraton Society Hill, Philadelphia, PA. IEEE CS Press (2006b)

De Lucia, A., Oliveto, R., Tortora, G.: IR-based traceability recovery processes: An empirical comparison of “One-Shot” and incremental processes. In: Proceedings of 23rd International Conference Automated Software Engineering, pp. 39–48. ACM Press, L’Aquila, Italy (2008)

De Lucia, A., Oliveto, R., Tortora, G.: Assessing IR-based traceability recovery tools through controlled experiments. Empirical Softw. Eng. **14**(1), 57–93 (2009a)

De Lucia, A., Oliveto, R., Tortora, G.: The role of the coverage analysis in traceability recovery process: A controlled experiment. In: Proceedings of 25th International Conference on Software Maintenance. IEEE Press, Edmonton, Canada (2009b)

De Mori, R.: Spoken Dialogues with Computers. Academic, London (1998)

Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.: Indexing by latent semantic analysis. J. Amer. Soc. Informat. Sci. **41**(6), 391–407 (1990)

Dekhtyar, A., Hayes, J.H., Menzies, T.: Text is software too. In: Proceedings of Mining of Software Repositories Workshop, pp. 22–26. Edinburgh, Scotland (2004)

Di Penta, M., Gradara, S., Antoniol, G.: Traceability recovery in RAD software systems. In: Proceedings of 10th International Workshop in Program Comprehension, pp. 207–216. IEEE CS Press, Paris, France (2002)

Dumais, S.T.: Improving the retrieval of information from external sources. Behav. Res. Meth. Instrum. Comput. **23**, 229–236 (1991)

Enslen, E., Hill, E., Pollock, L.L., Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis. In: Proceedings of the 6th International Working Conference on Mining Software Repositories, pp. 71–80. Vancouver, British Columbia, Canada (2009)

Gibiec, M., Czauderna, A., Cleland-Huang, J.: Towards mining replacement queries for hard-to-retrieve traces. In: Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering, pp. 245–254. ACM Press, Antwerp, Belgium (2010)

Haiduc, S., Marcus, A.: On the use of domain terms in source code. In: Proceedings of 16th IEEE International Conference on Program Comprehension, pp. 113–122. IEEE CS Press, Amsterdam, The Netherlands (2008)

Harman, D.K.: Overview of the first Text REtrieval Conference (TREC-1). In: Proceedings of the First Text REtrieval Conference (TREC-1), pp. 1–20. NIST Special Publication, Gaithersburg, MD (1993)

Hayes, J.H., Dekhtyar, A., Osborne, J.: Improving requirements tracing via information retrieval. In: Proceedings of 11th IEEE International Requirements Engineering Conference, pp. 138–147. IEEE CS Press, Monterey, CA (2003)

Hayes, J.H., Dekhtyar, A., Sundaram, S.K.: Advancing candidate link generation for requirements tracing: The study of methods. IEEE Trans. Softw. Eng. **32**(1), 4–19 (2006)

Hollink, V., Kamps, J., Monz, C., de Rijke, M.: Monolingual document retrieval for European languages. Inform. Retriev. **7**(1–2), 33–52 (2004)

Jurafsky, D., Martin, J.: Speech and Language Processing. Prentice Hall, Englewood Cliffs, NJ (2000)

Keenan, E.L.: Formal Semantics of Natural Language. Cambridge University Press, Cambridge (1975)

Lawrie, D.J., Binkley, D., Morrell, C.: Normalizing source code vocabulary. In: Proceedings of the 17th Working Conference on Reverse Engineering, pp. 3–12. IEEE CS Press, Beverly, MA (2010)

Lormans, M., Deursen, A., Gross, H.-G.: An industrial case study in reconstructing requirements views. Empirical Softw. Eng. **13**(6), 727–760 (2008)

Lormans, M., Gross, H., van Deursen, A., van Solingen, R., Stehouwer, A.: Monitoring requirements coverage using reconstructed views: An industrial case study. In: Proceedings of 13th Working Conference on Reverse Engineering, pp. 275–284. IEEE CS Press, Benevento, Italy (2006)

Lormans, M., Van Deursen, A.: Reconstructing requirements coverage views from design and test using traceability recovery via LSI. In: Proceedings of 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 37–42. ACM Press, Long Beach, CA (2005)

Lormans, M., van Deursen, A.: Can LSI help reconstructing requirements traceability in design and test? In: Proceedings of 10th European Conference on Software Maintenance and Reengineering, pp. 45–54. IEEE CS Press, Bari, Italy (2006)

Madani, N., Guerrouj, L., Di Penta, M., Guéhéneuc, Y.-G., Antoniol, G.: Recognizing words from source code identifiers using speech recognition techniques. In: Proceedings of the 14th European Conference on Software Maintenance and Reengineering. CSMR, Madrid, Spain (2010)

Marcus, A., Maletic, J.I.: Recovering documentation-to-source-code traceability links using latent semantic indexing. In: Proceedings of 25th International Conference on Software Engineering, pp. 125–135. IEEE CS Press, Portland, Oregon (2003)

Marcus, A., Maletic, J.I., Sergeyev, A.: Recovery of traceability links between software documentation and source code. Int. J. Softw. Eng. Knowl. Eng. **15**(5), 811–836 (2005)

Ney, H., Essen, U.: On smoothing techniques for bigrambases natural language modelling. In: Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, pp. 825–828. IEEE CS Press, Toronto, ON (1991)

Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: On the equivalence of information retrieval methods for automated traceability link recovery. In: Proceedings of the 18th IEEE International Conference on Program Comprehension, pp. 68–71. Braga, Portugal (2010)

Porter, M.F.: An algorithm for suffix stripping. Program **14**(3):130–137 (1980)

Poshyvanyk, D., Gael-Gueheneuc, Y., Marcus, A., Antoniol, G., Rajlich, V.: Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Trans. Softw. Eng., **33**(6), 420–432 (2007)

Ramesh, B., Jarke, M.: Toward reference models for requirements traceability. IEEE Trans. Softw. Eng. **27**:58–93 (2001)

Revelle, M., Dit, B., Poshyvanyk, D.: Using data fusion and web mining to support feature location in software. In: Proceedings of the 18th IEEE International Conference on Program Comprehension, pp. 14–23. Braga, Portugal (2010)

Salton, G., Wong, A., Yang, C.S.: A vector space model for information retrieval. Commun. ACM **18**(11), 613–620 (1975)

Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., De Palma, C.: Supporting software evolution through dynamically retrieving traces to UML Artifacts. In: Proceedings of 7th IEEE International Workshop on Principles of Software Evolution, pp. 49–54. IEEE CS Press, Kyoto, Japan (2004)

Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. J. Document. **28**, 11–21 (1972)

Witten, I.H., Bell, T.C.: The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. IEEE Trans. Inform. Theory **37**(4), 1085–1094 (1991)

Yadla, S., Huffman Hayes, J., Dekhtyar, A.: Tracing requirements to defect reports: an application of information retrieval techniques. Innov. Syst. Softw. Eng.: A NASA J. **1**(2), 116–124 (2005)

Zou, X., Settimi, R., Cleland-Huang, J.: Phrasing in dynamic requirements trace retrieval. In: Proceedings of the 30th Annual International Computer Software and Application Conference, pp. 265–272. Chicago, IL (2006)

Zou, X., Settimi, R., Cleland-Huang, J.: Term-based enhancement factors for improving automated requirement trace retrieval. In: Proceedings of International Symposium on Grand Challenges in Traceability, pp. 40–45. ACM Press, Lexington, Kentuky (2007)

Zou, X., Settimi, R., Cleland-Huang, J.: Evaluating the use of project glossaries in automated trace retrieval. In: Proceedings of the International Conference on Software Engineering Research and Practice, pp. 157–163. Las Vegas, NV (2008)

Zou, X., Settimi, R., Cleland-Huang, J.: Improving automated requirements trace retrieval: A study of term-based enhancement methods. Empir. Softw. Eng. **15**(2), 119–146 (2010)