Enhancing Software Traceability By Automatically Expanding Corpora With Relevant Documentation

Tathagata Dasgupta, Mark Grechanik
University of Illinois at Chicago
Chicago, IL 60607–7053
{tdasgu2,drmark}@uic.edu

Evan Moritz, Bogdan Dit, Denys Poshyvanyk
College of William and Mary
Williamsburg, VA 23185
{eamoritz,bdit,denys}@cs.wm.edu

***Abstract*—Software traceability is the ability to describe and follow the life of a requirement in both a forward and backward direction by defining relationships to related development artifacts. A plethora of different traceability recovery approaches use information retrieval techniques, which depend on the quality of the textual information in requirements and software artifacts. Not only is it important that stakeholders use meaningful names in these artifacts, but also it is crucial that the same names are used to specify the same concepts in different artifacts. Unfortunately, the latter is difficult to enforce and as a result, software traceability approaches are not as efficient and effective as they could be – to the point where it is questionable whether the anticipated economic and quality benefits were indeed achieved.**

We propose a novel and automatic approach for expanding corpora with relevant documentation that is obtained using external function call documentation and sets of relevant words, which we implemented in TraceLab. We experimented with three Java applications and we show that using our approach the precision of recovering traceability links was increased by up to 31% in the best case and by approximately 9% on average.

***Keywords*—software traceability; machine learning; API call;**

I. Introduction

Software traceability is the ability to describe and follow the life of a requirement in both a forward and backward direction by defining relationships to related development artifacts [29]. Recovering *traceability links (TLs)* (or *traces*) between requirements and software artifacts automatically and with high precision has a significant economic impact, since TLs improve the quality of different software maintenance tasks by enabling stakeholders to reveal errors and ambiguities in requirements and to ensure that all requirements are implemented and tested. Software traceability is especially useful when it connects concepts from the problem domain (where requirements are expressed using vague objectives or wish lists) to the solution domain (i.e., the domain in which engineers use their ingenuity to solve problems [33, pages 87,109]), e.g., the source code is a solution domain. When creating TLs, stakeholders map high-level intent reflected in the problem domain to low-level implementation details in the solution domain thereby solving an instance of the concept assignment problem [6]. It is highly desirable that programmers who implement specific requirements work with other stakeholders to record traces between requirements and their implementations using traceability matrices. While this is an ideal method, which likely leads to higher precision, it contributes to the high cost of software [25]. In addition, TLs are often recorded after software is implemented to avoid interruption of software development, which makes the task of recovering TLs approximate and error-prone.

Automating software traceability is a big and important problem. Delegating the task of computing traces between different artifacts to an automated tool saves cost and improves various software maintenance efforts. Even with manual traceability, some traces are erroneous, and validating these traces also requires a large investment. It is reported that less than 47% of Fortune 500 companies made their business requirements traceable and well integrated into testing, making software traceability as one of the biggest problems of software engineering [15].

A plethora of different *information retrieval (IR)* approaches for recovering TLs automatically depend on the quality of textual information in requirements and software artifacts. Not only is it important that stakeholders use meaningful names in these requirements and artifacts, but it is also equally important that the same names are used to specify the same concepts in different requirements documents and artifacts. Unfortunately, the latter is difficult to enforce, and as a result, software traceability approaches are not as efficient and effective as they could be – to the point where it was questionable whether the anticipated economic and quality benefits were indeed achieved from traceability approaches.

A fundamental problem of using IR approaches for software traceability is the mismatch in words that are used in requirements and software artifacts to describe high-level concepts. Words are fundamental blocks for existing traceability approaches to compute similarities between artifacts, and subsequently TLs, by matching words in these artifacts (e.g., words in requirements documents, comments in the source code, or the names of program variables and types). If no match is found, then potentially correct TLs are never recovered. This situation is aggravated by the fact that many applications are developed by large teams where different artifacts are created and maintained by different stakeholders; matches between words that designate the same concepts are not guaranteed.

Moreover, programmers routinely use *Application Programming Interface (API)* calls from third-party vendors to encode specific implementations of high-level concepts. For example, encountering an instance of the class `DESKeySpec` in the source code will undoubtedly lead a stakeholder to infer that a requirement for encrypting sensitive data is implemented

in the fragment of the code that uses this instance, even though the class name does not match any words in the requirements documentation. Of course, the stakeholder uses the information that comes from the descriptions of the API calls to obtain a high-level concept and match it to requirements. This is where our intuition lies: *using relevant external documentation to expand the corpora of different artifacts may lead to uncovering more TLs with a higher degree of precision*.

We observed that in industry many computer professionals intuitively attempted to implement this approach in ad-hoc ways. Some tried to merge corpora that they obtained from different applications that belong to the same domain; others mixed different related corpora to increase the probability of retrieving correct TLs using different IR approaches [5]. Even though stories that describe successful results are disseminated about the effectiveness of these approaches, it is not clear how scientifically valid these results are. In addition, expanding corpora manually is an unsystematic and laborious effort whose precision depends on how similar merged corpora are perceived by stakeholders.

We propose an automatic approach for *ENhancing TRAceability usiNg API Calls and rElevant woRds (ENTRANCER)*. The input to ENTRANCER is program source code, requirements documents, and other artifacts, which we collectively call the *base corpora*, and the external information sources from which we enhance the corpora. These sources consist of documentation for third-party API calls (e.g., the Java Development Kit (JDK)[1]) and external lexical databases with sets of cognitive synonyms (e.g., Wordnet[2] synsets), which we call the *enhancing corpora*. The core idea of ENTRANCER is to automatically expand words in the base corpora with semantically related words from the enhancing corpora to recover TLs or traces among requirements documents and other artifacts with a high degree of precision. To the best of our knowledge, this is the first comprehensive study of an automated approach that expands the base multilingual corpora, i.e., Italian and English.

We evaluated ENTRANCER on three open-source Java applications using the TraceLab experimental framework and obtained results that suggest our approach is effective. We showed that ENTRANCER can increase the precision of the recovering TLs by up to 31% in the best case. All data files and subject applications are available from the project website http://www.cs.uic.edu/~drmark/entrancer.htm.

II. Our Approach

In this section, we explain the theory of relevance behind our approach, formulate our hypotheses, and describe the experimental framework TraceLab.

A. Software Traceability Is A Similarity Measure

Suppose that a user is given two photographs and asked to establish traces between different features on these images. Naturally, a user looks for similarities among different features, and depending on the resolution of the image, some features may be blurry or collapsed into just a few pixels. By increasing the resolution and adding more pixels to features, the user can find similarities among these features much faster.

Finding similar features on two images is analogous to finding TLs by measuring similarities between components in the source code and requirements. Many automated traceability approaches imitate stakeholders by measuring similarities between elements in the problem and solution domains (e.g., how closely words in a requirement paragraph match words in comments and identifier names of the source code). When similarity between artifacts is computed using an IR approach, and this similarity is above a user-defined threshold, a TL is recorded between these artifacts. A straightforward approach for measuring similarities between requirements and software artifacts is to count matches among words from requirements to source code identifiers (e.g., names of variables, types, etc.). Computing traces with high precision depends on the quality of information in artifacts, specifically, on programmers choosing meaningful names that reflect correctly the concepts or abstractions that they implement, but this compliance is generally difficult to enforce [2][39].

We define software traceability as the similarity between artifacts by using Mizzaro's well-established conceptual framework for relevance [40, 41]. In Mizzaro's framework, similar artifacts are relevant to one another if they share some common concepts. Once these concepts are known, a corpus of artifacts can be clustered by how they are relevant to these concepts. Subsequently all artifacts in each cluster will be more relevant to one another when compared to artifacts that belong to different clusters. This is the essence of the cluster hypothesis that specifies that artifacts that cluster together tend to be relevant to the same concept [48][39], and consequently traceability links are established between these artifacts.

B. Our Hypotheses

A requirement is relevant to some software artifact if this artifact implements the same abstraction that is specified in this requirement [39]. For example, if a requirement specifies that cryptographic services should be used to protect information, and a module in an application uses encryption, then these requirement and software artifacts are relevant to a certain degree. Currently, most IR approaches use artifacts as "bags of words" with no semantics, and the relevancy or similarity of these artifacts to one another can be determined by matches between these words. This is the essence of *syntagmatic associations*, where artifacts are considered similar (i.e., traced to each other) when terms (i.e., words) in these documents occur together [45]. The problem with this approach is that computed traceability links are relatively imprecise when compared with ENTRANCER (as we show in Section IV).

Syntagmatic associations are used in a variety of techniques for computing TLs, such as *Vector Space Model (VSM)*, where artifacts are represented as vectors of words and a similarity measure is computed as the cosine between these vectors [46]. One main problem with VSM is that different programmers can use the same words to describe different requirements (i.e., the *synonymy* problem) and they can use different words to describe the same requirements (i.e., the *polysemy* problem). This problem is a variation of the vocabulary problem, which states that "no single word can be chosen to describe a

[1]http://docs.oracle.com/javase/6/docs/api/index-files/index-1.html
[2]http://wordnet.princeton.edu

programming concept in the best way" [26]. This problem is general to IR, but somewhat mitigated by the fact that different programmers who participate in the projects use coherent vocabularies to write code and documentation, thus increasing the chance that two words in different applications may describe the same requirement.

Our **first hypothesis** is that *it is possible to increase the precision of IR approaches that are based on syntagmatic associations by expanding the vocabulary of artifacts using related words*. Consider the requirement that information should be protected from unauthorized use. Suppose that a software artifact contains the variable "`encrypt`", which is syntactically different from the word "`protect`", which makes it difficult to establish the match. However, since these words co-occur in many documents, we can enhance the vocabularies of both requirements document and software artifacts by appending both words to the corpora obtained from these requirements documents and software artifacts. As a result, a match will be found and a TL will be established.

In Mizzaro's framework, a key characteristic of relevance is how information is represented in artifacts. We concentrate on *semantic anchors*, which are elements of artifacts that precisely define the artifacts' semantic characteristics [39]. Semantic anchors may take many forms (e.g., they can refer to elements of semantic ontologies that are precisely defined and agreed upon by different stakeholders). This is the essence of *paradigmatic associations* where artifacts are considered similar if they contain terms with high semantic similarities [45]. While paradigmatic associations are considered more reliable than syntagmatic ones, both can be useful when computing TLs between requirements and software artifacts.

Our **second hypothesis** is that *by using semantic anchors it is possible to compute similarities between requirements and software artifacts with a higher degree of accuracy*. Our idea is based on the observation that in the solution domain engineers go into implementation details to realize requirements (i.e., engineers look for reusable abstractions that are often implemented using third-party API calls). Since programs contain API calls with precisely defined semantics, these API calls can serve as semantic anchors to compute the degree of similarity between requirements and artifacts by matching the semantics of these applications that are expressed with these API calls. Programmers routinely use third-party API calls (e.g., the *Java Development Kit (JDK)*) to implement various requirements [10, 21, 30, 31, 47]. API calls from well-known and widely used libraries have precisely defined semantics–unlike names of program variables, types, and words that programmers use in comments. Since these documentations are written by different people who have different vocabularies, appending words from these documentations to the corpora makes a richer vocabulary. In this paper, we use the JDK API calls as semantic anchors to compute similarities among applications by expanding these API calls with words from their documentations, thereby partially addressing the vocabulary problem.

Finally, our **third hypothesis** is that *by using the hybrid of syntagmatic and paradigmatic vocabulary expansion we can increase the precision of computing TLs*. Our rationale for this hypothesis is that combining both approaches expands the vocabulary and increases the precision of IR approaches for traceability.

C. Experimental Testbed – TraceLab

TraceLab [12, 35, 14] is a software infrastructure designed to address the issue related to the reproducibility of experiments (i.e., lack of implementation, implementation details, datasets, etc.) in software engineering (SE) research.

TraceLab provides (i) a set of predefined components (i.e., tools that are commonly used in SE techniques and approaches, such as data importers, preprocessors, IR approaches, state of the art TL recovery techniques, etc.), as well as (ii) a development kit which includes the guidelines and support for creating custom components. These components can be assembled to create experiments, which can be executed alongside other SE techniques, on the same datasets, and their results can be compared to determine which technique produces the best results using standard metrics (e.g., precision, recall, etc.), as well as statistical tests. In addition, the newly created experiments, which are fully reproducible, are shared with the community in order to facilitate the creation of new techniques (based on the existing one) and the comparison of new techniques against existing ones.

TraceLab was funded by the National Science Foundation and was developed at DePaul University in collaboration with Kent State University, University of Kentucky, and the College of William and Mary. Since its introduction, TraceLab has already been successfully used in several projects [43, 23, 24, 16].

III. Experimental Design

In this section, we describe the experimental design for evaluating ENTRANCER.

A. Research Questions

In this paper, we claim that we can improve the precision of computing software traceability using similarity-based methods by expanding the corpus obtained from application's artifacts with relevant documentation. We seek to answer the following Research Questions (RQs).

- RQ_1 Does using expansion of the corpus with documentation from JDK API calls result in a higher precision of recovering TLs?
- RQ_2 Does using expansion of the corpus with a combined documentation from JDK API calls and Wordnet result in a higher precision of recovering TLs?
- RQ_3 Does including words from comments result in a higher precision of recovering TLs when expanding the corpus with a combined documentation from the JDK API calls and Wordnet?
- RQ_4 How does the size of the corpus affect the precision of recovering TLs?
- RQ_5 Is ENTRANCER equally effective using different IR approaches for recovering software TLs?

With these RQs, we decompose our experimental results to evaluate the effectiveness of ENTRANCER for different components of software traceability. In all cases, we start with the initial corpus that contains words from requirements documents on one side and words from the source code of a subject application on the other side.

With RQ_1, we investigate a claim that replacing JDK method occurrences in initial source code corpus with words that appear in the method description in Java API Specification, increases correct TL recovery. Our rationale is the following: API calls allow programmers to express abstraction of high-level concepts from requirement documentation thus reducing the chance of IR methods matching relevant words in the source code corpus. Hence, if every method invocations is replaced with its description, it introduces more relevant words in the corpus which in turn should lead to more recovered correct TLs. The rationale for RQ_2 is similar. We use the Italian WordNet called MultiWordnet [3], to include more relevant words by including synsets of dictionary words existing in the corpus, since the subject applications are written by Italian programmers who wrote comments and identifier names in Italian.

Unlike rest of the source code, comments are written in natural language by the programmers and as such should contain relevant dictionary words and phrases expressing high-level intents. So in RQ_3, we inspect how significant is the impact of source code comments on traceability.

To answer RQ_4, we will consider a correlation between the size of the corpus and the precision of computing TLs when applying different IR methods that we describe in Section III-D1. Since having more words in corpus increases the probability that more correct matches may be obtained between these words and words in requirements documents, the rationale for RQ_4 is to establish if the size of the corpus alone may be indicative of the future quality of obtained TLs.

Finally, our claim is that we designed and implemented a methodology as part of ENTRANCER for achieving a higher precision for software traceability by expanding the corpora of software applications with relevant documentation. The rationale for RQ_5 is to evaluate results to determine if ENTRANCER is effective when compared across different IR-based traceability approaches.

B. Subject Software Applications

The subject Java applications that we used to evaluate different traceability approaches are publicly available for researchers from the TraceLab web site [4]. These applications have been used in other studies, making it easy to reproduce our results and compare them with other approaches.

Albergate, the first evaluated application, is a software system designed to implement all the operations required to administrate and manage a small/medium size hotel (room reservation, bill calculation, etc.). It was developed from scratch by a team of final year students at the University of Verona (Italy) on the basis of 16 functional requirements expressed (as well as all the other system documentation) in the Italian language. Albergate exploits a relational database and consists of 13 requirements documents, 95 classes and about 20 KLOC [4]. Albergate has been used in different traceability studies [38]. **eTour** is an electronic tourist guide developed by students. It has 58 use cases and a total of 174 Java classes with 366 recorded traceability links. Finally, **SMOS** is an application that is used to monitor high school students (e.g., absence, grades). It has 67 use cases and a total of 100 Java classes with 1,044 recorded TLs.

[3]http://multiwordnet.fbk.eu

[4]http://www.coest.org/index.php/for-researches1

C. Preprocessing Source Code Using Identifier Splitting

High quality identifier splitting is very important for achieving good precision for software traceability approaches. Low quality identifier splitting results in words that are not in a dictionary or they incorrectly represent the semantic meaning that they should otherwise convey.

The source code of the three subject applications, was written by Italian developers, and they contain a combination of both Italian and English words in identifier construction. This prevents us from using any identifier splitting method that relies primarily on heuristics derived from mining source code repository. Accordingly, we built a fast, regular expression based identifier splitting that uses camel case and Java identifier naming convention to split each identifier into their separate English/Italian words.

D. Methodology

Our hypotheses are partially based on our idea that it is better to compute similarity-based traceability by utilizing relevant words from Wordnet and API calls as semantic anchors that come from the JDK and that programmers use to implement various requirements. To evaluate ENTRANCER, we carry out experiments to explore its effectiveness and to answer RQs.

1) Independent Variables: In our experimental design, we consider two types of independent variables: IR approaches and corpus treatment methods.

IR Approaches. TraceLab already implements a set of IR approaches, which will be described briefly as independent variables for our experimental design. In the context of ENTRANCER, these techniques take as input a corpus of documents (i.e., the target artifacts) and a set of queries (i.e., the source artifacts) and compute the textual similarities between the source and target artifacts.

One of earliest techniques is the **Vector Space Model (VSM)** [46], which works as follows. First, it represents the set of documents (i.e., the corpus) as a term-by-document (TD) matrix, where each element of the matrix represents the number of occurrences of the term in the document. Second, the TD matrix is normalized and weighted using a traditional weighting scheme, such as the term frequency-inverse document frequency (tf-idf), which gives more weight to terms that are relevant to the document, and less weight to common terms that frequently appear in the documents. Third, the similarities between the source artifacts, which are also represented as a vector of terms, and the target artifacts are computed using the cosine similarity between these two vectors. Although VSM produces good results when the vocabulary between the source and target artifacts matches, it does not handle the polysemy and synonymy problem.

To overcome this issue, a more advanced IR technique called **Latent Semantic Indexing (LSI)** [19] was introduced. LSI also represents the corpus as a TD matrix, but it uses Singular Value Decomposition (SVD) to decompose the weighted

TD matrix into three different matrices using a dimensionality reduction factor k, specified by the user. The reduced space of the decomposed matrices is an approximation of the original TD matrix and captures the most important concepts present in the structure of the original matrix, and at the same time ignoring any minor differences in terminology, thus addressing the polysemy and synonymy problem. Similarly to VSM, LSI also uses the cosine similarity to determine the similarities between source and target artifacts.

Jensen-Shannon (JS) [1] is a recent IR technique that represents each artifact of the corpus as a probability distribution of the terms occurring in the artifact. The probability distribution is based on the weight assigned to each term for that particular artifact. The similarities between two software artifacts (i.e., two probability distributions), are measured using an entropy-based metric, called the Jensen-Shannon Divergence. Similarly to VSM, JS does not take into account the relation between terms, thus it encounters the same problems, namely polysemy and synonymy.

Latent Dirichlet Allocation (LDA) [8] is a generative probabilistic technique, which models each artifact as a mixture of topics. In other words, each artifact is represented as a probability distribution over a set of topics, and each topic is represented as a probability distribution over the set of terms in the corpus. In order to generate a model, the user must specify the number of topics, and other parameters that affect the "smoothness" of the distribution of the topics in the documents, as well as the "smoothness" of the distribution of the words in the topic. The textual similarity between two software artifacts represented as topic distributions is computed using the Hellinger distance [7].

The Relational Topic Model (RTM) [9] is a hierarchical probabilistic model that generalizes LDA, by also considering the links between the modeled artifacts. RTM takes as input the corpus of documents (same as LDA) and a set of predefined links between software artifacts, if they exist. Regardless of the predefined links, the output produced by RTM includes (i) the topic distribution for each artifact (same as LDA), as well as (ii) a set of links between the artifacts based on the similarities of their topic distribution (i.e., artifacts with similar topics will be connected via a link).

Given a set of observations produced by different IR methods, Principal Component Analysis (PCA) can be used to determine various orthogonal dimensions (called principal components) present in the data. These principal components also quantify the variability found in the data. For example, when PCA was applied on IR techniques in the context of traceability link recovery (TLR), Gethers et al. [27] identified that VSM and JS produced equivalent results, and RTM produced orthogonal (complementary) results. Therefore, in order to capture more variability in the data, these orthogonal techniques were combined, resulting in the hybrid techniques VSM-RTM and JS-RTM [27]. These hybrid techniques were generated using an affine transformation [34] between the similarities produced by these orthogonal techniques, where the weight of the IR technique is (1) given equal weight (denoted as IR_1+IR_2), and (2) assigned a weight proportional to the variance obtained during the PCA analysis, denoted as $IR_1 + IR_2(PCA)$. The study showed that combining orthogonal IR methods produced improved results over standalone IR methods for recovering TLs [27].

Corpus Treatment Methods. One of our goals is to investigate how different corpus treatment methods affect the precision of IR approaches for computing TLs. We translated the JDK API documentation into Italian using Google Translate, since the applications and requirements documents were written by Italian programmers. We select five different corpus treatment methods.

- The method **S**trawman (**S**) is a baseline method for our experiments where the complete source code corpus is treated as bag of words.
- Next, for the method **J**DK API call expansion, (**J**), we replaced JDK API calls with their corresponding description in the JDK documentation. This is followed by identifier splitting. Comments from the source code are discarded, so that we can evaluate how expanding JDK API calls with words from the relevant API call documents affects the precision of recovery of TLs thus reducing the influence of other confounding variables.
- The method **J+W** is a one-step extension of the method **J**, where **W**ordnet synsets of dictionary words found in the corpus are injected.
- **J+S** is the extension of the methods **J** and **S**, where the JDK API calls are expanded along with source comments present in the corpus.
- Finally, the method **J+S+W** is ENTRANCER where all expansion techniques are combined in a single approach.

Our goal is to address RQs by running experiments with all combinations of IR and corpus treatment methods.

2) Dependent Variables: Dependent variables for ENTRANCER are precision (P) and recall (R). To evaluate the accuracy of each IR method, the number of correct links and false positives were collected for each recovery activity performed by a tool. The tool takes as an input the ranked list of candidate links and classifies each link as correct link or false positive until all correct links are recovered. Such a classification is automatically performed by the tool exploiting the original traceability matrix as an oracle.

The values of P and R are computed as follows: $R = \frac{|\texttt{cor} \cap \texttt{ret}|}{|\texttt{cor}|}$ and $P = \frac{|\texttt{cor} \cap \texttt{ret}|}{|\texttt{ret}|}$, where `cor` and `ret` represent the sets of correct links and links retrieved by the tool, respectively. Other than recall and precision, we also use average precision [19], which returns a single value for each ranked lists of candidate links provided.

In this paper we report the values of average precision for $R = 100\%$, i.e., when we apply IR methods and we change the similarity threshold value to ensure that all TLs from the oracle are in the set of recovered TLs. However, making this threshold too low leads to the decreased precision, since many incorrect TLs are added to the set of accepted TLs. Determining the range of acceptable similarity threshold values is beyond the scope of this paper and is a subject of future research.

E. Threats to Validity

In this section, we discuss threats to the validity of this experimental design and how we address and minimize these threats.

1) Internal Validity: **Expansion techniques.** Since evaluating hypotheses is based on the data collected from external sources such as the JDK documentation and Wordnet, we identify three threats to internal validity: richness of the vocabularies and their relation to the domains to which the subject applications belong, and the uniformity of word distribution among these sources.

Even though the JDK documentation offers a richer vocabulary, a threat to validity is that this vocabulary is generic, since it does not relate to specific domains for which subject applications are built. Therefore, it is likely that many words that were added to the corpus from the JDK documentation and Wordnet did not result in computing correct TLs. Moreover, there is a possibility that by adding some generic words to the corpus, it is possible to compute TLs that are incorrect. In addition, vocabularies can be much richer if domain-specific dictionaries or SDKs are used to expand the corpora. Also, an inequality in distributions of words among different topics may result in computing more correct TLs for some modules and fewer correct TLs for some other modules for which there are fewer API calls. We address this threat to validity by recording TLs for different components and using statistical information about distributions of TLs to make conclusions with respect to the RQs.

Traceability methods. Choosing ineffective IR-based traceability methods pose a big threat to validity. If methods are too general or trivial (e.g., exact matches among some words), then every possible TL that has some similar words in its source code and requirements will be retrieved, thus inundating stakeholders with TLs that are hard to evaluate. On the other hand, if an IR-approach is specific to a subject application (e.g., ontology-based techniques), high precision will be obtained, thus creating a bias towards this specific application and IR-approach. To avoid this threat, we implemented our experimental design in TraceLab (described in Section II-C), which offers a diversified set of widely used IR-method that are application independent. While this diversification of tasks does not completely eliminate this threat to validity, it reduces it significantly.

2) External Validity: To make results of this experiment generalizable, we must address threats to external validity, which refer to the generalizability of a casual relationship beyond the circumstances of our experiment. The fact that supports the validity of this experimental design is that the subject applications and traceability methods are representative of methods that are used in industry and research. A threat to external validity concerns the usage of software traceability tools in the industrial settings, where applications may not use third-party API call libraries. However, it is highly unlikely that modern large-scale software projects can be effectively developed, maintained, and evolved without this reuse.

IV. Results

In this section, we describe the results of our experiments and using these results we give answers to our RQs.

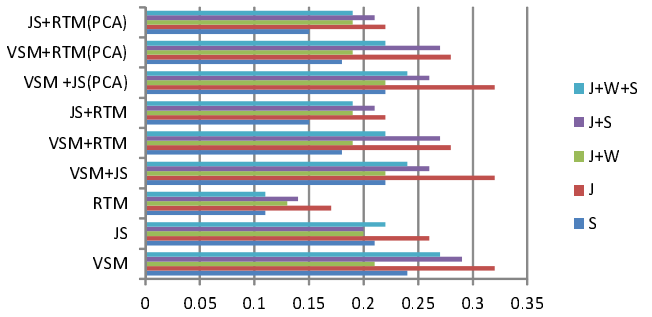

Fig. 1. Precisions for recovering TLs using different IR methods and corpus treatment methods for subject application Albergate.

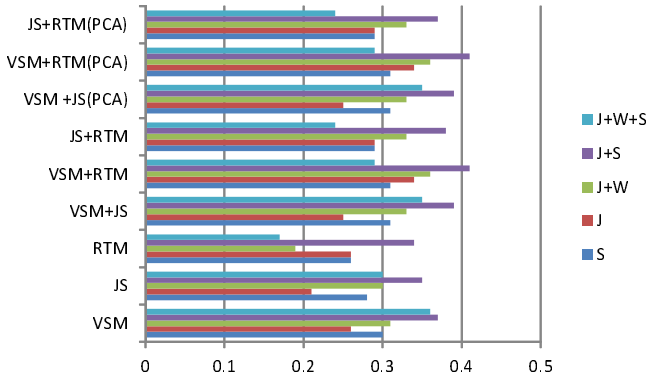

Fig. 2. Precisions for recovering TLs using different IR methods and corpus treatment methods for subject application eTour.

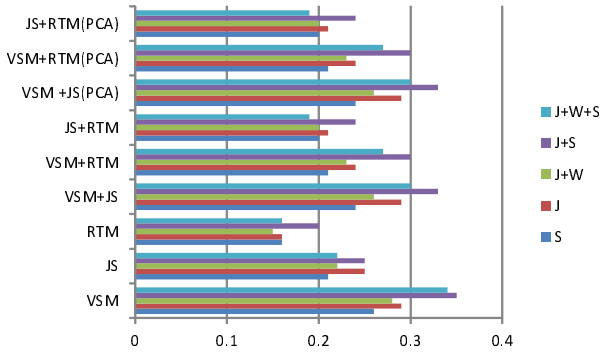

Fig. 3. Precisions for recovering TLs using different IR methods and corpus treatment methods for subject application SMOS.

The results of experiments with subject applications for different values of the dependent variable Precision, P are shown in Table I and they are visualized in Figure 1, Figure 2, and Figure 3 for the corresponding subject applications. ENTRANCER leads to the increase in P in the majority of cases (75.92%), thus positively answering RQ_1-RQ_3. Closer examination of the data reveals that for the method JS for the subject application Albergate the precision increases for the

TABLE I. RESULTS OF EXPERIMENTS WITH SUBJECT APPLICATIONS FOR DIFFERENT IR TECHNIQUES. COLUMNS 2-6 DESCRIBE THE SIZE OF THE DATASETS IN TERMS OF NUMBER OF REQUIREMENTS DOCUMENTS, JAVA SOURCE CODE FILES, NUMBER OF POSSIBLE LINKS, NUMBER OF ACTUAL LINKS IN THE GOLD SET, AND THE RATIO OF ACTUAL TO POSSIBLE LINKS. COLUMNS 7-9 SHOW THE SOURCE CODE AND REQUIREMENTS DOCUMENT SIZES IN KB AND THE RATIO OF THESE SIZES. THE NEXT COLUMN, "MAX EXP RATIO," SHOWS THE RATIO OF THE NUMBER OF THE UNIQUE WORDS IN THE EXPANDED CORPUS TO THE NUMBER OF UNIQUE WORDS IN THE ORIGINAL CORPUS. FINALLY, THE COLUMN LABELED "METHOD TRACEABILITY" AND THE FOLLOWING FIVE COLUMNS LIST THE MEAN VALUES OF THE PRECISION FOR CORPUS TREATMENT METHODS FROM SECTION III-D1 FOR THE VALUES OF RECALL, $R = 100\%$.

Subject App	Doc Files	Code Files	Posbl Links	Actual Links	PL/AL Ratio	Code Size,KB	Req Corp Size,KB	D/C Ratio	Max Exp Ratio	Method Traceability	P for corpus treatment methods				
											S	J	J+W	J+S	J+W+S
Albergate	16	55	880	54	0.06	516	72	0.13	362.08%	VSM	0.24	0.32	0.21	0.29	0.27
										JS	0.21	0.26	0.2	0.2	0.22
										RTM	0.11	0.17	0.13	0.14	0.11
										VSM+JS	0.22	0.32	0.22	0.26	0.24
										VSM+RTM	0.18	0.28	0.19	0.27	0.22
										JS+RTM	0.15	0.22	0.19	0.21	0.19
										VSM +JS(PCA)	0.22	0.32	0.22	0.26	0.24
										VSM+RTM(PCA)	0.18	0.28	0.19	0.27	0.22
										JS+RTM(PCA)	0.15	0.22	0.19	0.21	0.19
eTourITA	67	100	6700	1044	0.16	715	181	0.25	108.58%	VSM	0.3	0.26	0.31	0.37	0.36
										JS	0.28	0.21	0.3	0.35	0.3
										RTM	0.26	0.26	0.19	0.34	0.17
										VSM+JS	0.31	0.25	0.33	0.39	0.35
										VSM+RTM	0.31	0.34	0.36	0.41	0.29
										JS+RTM	0.29	0.29	0.33	0.38	0.24
										VSM +JS(PCA)	0.31	0.25	0.33	0.39	0.35
										VSM+RTM(PCA)	0.31	0.34	0.36	0.41	0.29
										JS+RTM(PCA)	0.29	0.29	0.33	0.37	0.24
SMOS	67	100	6700	1044	0.16	715	181	0.25	242.17%	VSM	0.26	0.29	0.28	0.35	0.34
										JS	0.21	0.25	0.22	0.25	0.22
										RTM	0.16	0.16	0.15	0.2	0.16
										VSM+JS	0.24	0.29	0.26	0.33	0.3
										VSM+RTM	0.21	0.24	0.23	0.3	0.27
										JS+RTM	0.2	0.21	0.2	0.24	0.19
										VSM +JS(PCA)	0.24	0.29	0.26	0.33	0.30
										VSM+RTM(PCA)	0.21	0.24	0.23	0.3	0.27
										JS+RTM(PCA)	0.2	0.21	0.2	0.24	0.19

corpus treatment method J when compared to S, but it drops when applying the corpus treatment methods J+W, J+S, and J+W+S. In fact, when comparing the results for corpus treatment methods J and S, precision increases for all IR methods for the application Albergate, but it drops for the application eTour when applying the IR methods VSM, JS, VSM+JS, and VSM+JS(PCA). A possible explanation is that by removing comments from the source code for this application when applying the corpus treatment method J, the expansion rate of the corpus is not sufficient to increase precision for methods where word matches are important for computing TLs. *A conclusion from this observation is that expanding the corpus with the documentation JDK API calls only is often not enough to get higher precision of traceability links when applying word match similarity methods thus addressing* RQ_1.

We make similar observations about the precision of different approaches when combining expansion of the corpus with documentation from the JDK API calls and Wordnet. When comparing the results for corpus treatment methods J+W and J, precision increases for all IR methods for the application eTour, but it drops for the applications Albergate and SMOS for all IR methods. The resulting precision is still higher in most cases when compared to the baseline corpus treatment method S, however, it points out to an interesting phenomenon. Clearly, expanding the corpus with relevant words works only if the words in the original corpus can serve as good semantic anchors. In our case, using Google translator in the presence of mixed language vocabulary reduces the semantic quality of information. Identifier splitting also has limited precision, leading to situations where split words may not have semantic relevancy. As a result, expanding semantically irrelevant corpus with more irrelevant words leads to reduced precision. A possible reason that the application eTour avoids the reduction in precision is because the rate of its corpus expansion is the smallest when compared to the two other subject applications, thus having fewer erroneous TLs in the end. *We conclude that expansion of the corpus with a combined documentation from the JDK API calls and Wordnet does not always result in a higher precision of traceability links thus addressing* RQ_2.

Adding comments from the source code to the corpus when expanding the JDK API calls with their documentation leads across the board increase in P with exception of the IR method RTM, which remains the worst method in our experiment to see the effect of expanding the corpus with relevant documentation. Part of this is addressed in the previous work, noting that in some studies RTM was found to provide orthogonal results, which could be used to complement other IR techniques [27].

We conclude that including words from comments results in a higher precision of traceability links when expanding the corpus with a combined documentation from the JDK API calls and Wordnet.

We investigate the question of correlation between the

size of the corpus and the precision of traceability links. This research question can be answered with more rigorous experimentation in the future. To understand how each of our corpus modification methods effects the source code corpus, we measure the change in the total word count and the number unique words for each source code file from the subject applications. Table II shows the average percentage difference in the length of source code files when measured relative to the unmodified original source code files. eTour source code files are extensively commented with 43,211 words which is 19.21 and 2.21 times greater than Albergate and SMOS respectively. As a result eTour's corpus size actually shrinks and leads to a decrease in the precision values for the corpus treatment method **J** (which excludes comments). On the other hand, when using corpus treatment methods **J+S** and **J+W+S**, which include the comments, we observe an improvement in precision of recovering TLs across all IR methods.

TABLE II. AVERAGE PERCENTAGE DIFFERENCE IN THE LENGTH OF SOURCE CODE FILES WHEN MEASURED RELATIVE TO THE UNMODIFIED ORIGINAL SOURCE CODE FILES.

	Type	J	J+W	J+S	J+W+S
Albergate	total	202.05	476.80	200.02	624.87
	unique	96.71	261.06	136.81	362.08
eTour	total	-18.50	238.44	100.79	357.75
	unique	-60.41	32.20	16.50	108.58
SMOS	total	69.08	592.92	210.43	734.36
	unique	-18.34	161.68	62.36	242.17

Based on our experimental results, we conclude that there is a correlation between the size of the corpus and higher precision of recovered TLs thus addressing RQ_4.

ENTRANCER is not equally effective for different IR approaches for computing software traceability – VSM remains the biggest winner and RTM is the biggest loser among the used IR approaches. Adding VSM to a combination of other IR methods improves the overall precision. Best ratios of precision improvements for ENTRANCER are shown in Table III, where the maximum obtained precision increase is 31% for SMOS using the IR method VSM and the average precision gain is nine percent. The lowest precision is obtained with the IR method RTM. These best ratios are shown in Figure 4 for precision improvements when compared to the baseline **S** for different IR methods applied to subject applications with ENTRANCER that is shown in Table III. *We conclude that using VSM results in a higher precision of traceability links when expanding the corpus with a combined documentation from the JDK API calls and Wordnet thus addressing RQ_5.*

V. RELATED WORK

The first part of this section discusses in detail the most relevant papers for ENTRANCER. The second part of the section briefly enumerates a subset of the traditional TLR techniques and tools.

Cleland-Huang et al. [13] proposed a technique to recover TLs between regulatory codes and requirements by enhancing the original query with similar terms that would help address the polysemy and synonymy problem. More specifically, their technique uses the original query as an input to web search engines which retrieve a set of documents related to the query.

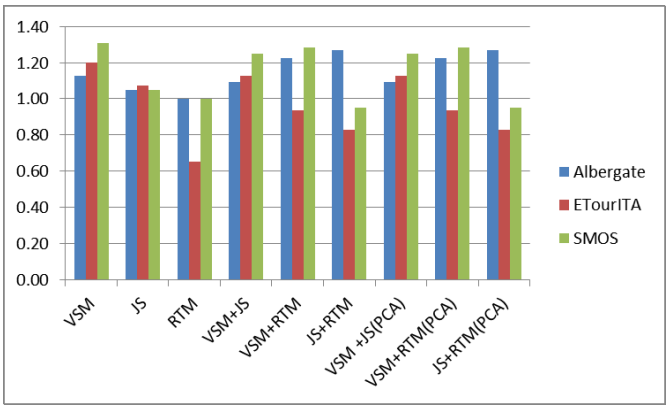

Fig. 4. Best ratios for precision improvements when compared to the baseline **S** for different IR methods applied to subject applications with ENTRANCER that is shown in Table III.

Among the words of these documents, the nouns and noun-phrases are extracted, and term specific metrics are computed (e.g., domain term frequency, domain specificity, concept generality). The nouns with the highest metric values are appended to the original query, and the enhanced query is used for retrieving TLs. A comparison with a basic IR TLR technique revealed that, in some cases, their proposed technique is more accurate in retrieving TLs [13]. Gibiec et al. [28] improved the traceability technique proposed by Cleland-Huang et al. [13], by automatically identifying the set of domain specific terms (from the retrieved web-mining results), which will be used to enhance the original query. Our approach is similar to Cleland-Huang et al.'s [13] and Gibiec et al.'s [28] approaches in terms of mining additional data to enhance the existing information. However, the main difference is that ENTRANCER uses API documentation to enhance the existing API calls found in the corpus, as opposed to expanding only the query.

Dekel and Herbsleb [21][22] introduced an approach based on the concept of knowledge push, which directs the attention of developers to certain API calls that might need extra consideration. First, the approach extracts some directives (e.g., restrictions, limitations, performance issues, alternatives, etc.) in the form of sentences that are embedded in the API documentation. Second, this knowledge, which could be easily skipped by a developer reading the API documentation, is "pushed" or presented to the developer by highlighting in the Eclipse IDE the method invocations that have associated these directives [21][22].

The identifier mismatch problem was investigated in the literature, and a different number of solutions have been suggested. Deissenboeck and Pizka [20] proposed a tool that enforces unique mappings between identifier names and concepts, in order to reduce the mismatch problem. The tool constantly updates the mappings while the system evolves. Lawrie and Binkley [36] proposed a solution for automatically expanding the splitted identifiers to their unabbreviated form. However, the impact of these tools and techniques [20][36][32] has not yet been evaluated for TLR.

Cleland-Huang et al. [11] advocated for writing requirements in a more concise and clear way, and to use a consistent domain-specific vocabulary. De Lucia et al. [17] proposed COCONUT, an IR-based TLR system that helps developers

TABLE III. BEST RATIOS OF PRECISION IMPROVEMENTS FOR ENTRANCER.

	VSM	JS	RTM	VSM+JS	VSM+RTM	JS+RTM	VSM +JS(PCA)	VSM+RTM(PCA)	JS+RTM(PCA)
Albergate	1.13	1.05	1.00	1.09	1.22	1.27	1.09	1.22	1.27
ETourITA	1.20	1.07	0.65	1.13	0.94	0.83	1.13	0.94	0.83
SMOS	1.31	1.05	1.00	1.25	1.29	0.95	1.25	1.29	0.95

to select the most meaningful identifier names, which are consistent with the domain terms found in the high-level artifacts (e.g., requirements).

Antoniol et al. [3] proposed an IR-based technique to recover TLs between documentation and source code using the vector space model (VSM). Their work was extended by Marcus et al. [37] who used latent semantic indexing (LSI) to recover TLs, and showed that this technique produces better results. Oliveto et al. [42] compared the performances of three IR-based techniques (i.e., VSM, LSI and Jensen-Shannon (JS)) and one topic-model technique, LDA (Latent Dirichlet Allocation). Their results show the IR-techniques produce equivalent results, whereas LDA produced complementary results, which are orthogonal to the ones produced by the IR techniques. The orthogonality of results was leveraged by Gethers et al. [27] who combined the information produced by the IR techniques VSM and JS with the information produced by the topic model technique RTM (Relational Topic Model) to produce superior results than using standalone techniques.

Among the numerous tools for supporting TLR that have been introduced we mention a few representative ones. TOOR [44] is a traceability tool that supports manual tracing between various software artifacts, as well as management of existing TLs. ADAMS [18] is a tool that automatically generates TLs between different software artifacts, using LSI.

VI. CONCLUSION

We created a novel and automatic approach for expanding corpora with relevant documentation that is obtained using external function call documentation and sets of relevant words. We experimented with three Java applications using the TraceLab framework and we evaluate four methods of systematically expanding source code corpus using relevant words and semantic anchors. These methods improve precision the of TL recovery in more than 75% of the cases, with 31% in the best case and approximately 9% on average.

ENTRANCER is most effective using the IR method *Vector Space Model (VSM)*, where artifacts are represented as vectors of words and a similarity measure is computed as the cosine between these vectors. Since the main idea of ENTRANCER is to expand the corpora with words that are relevant to words in the documents, applying ENTRANCER increases the probability of establishing matches between words in requirements and other software artifacts, thereby leading to sizeable increase in the precision of recovering TLs. However, other IR methods like RTM benefit less from ENTRANCER for a variety of different reasons, one of them being that the expanded corpora contains words that reduce the effectiveness of computing probabilistic distributions of these words across different topics. Thus, we conclude that ENTRANCER produces the best results when applied with VSM.

ACKNOWLEDGMENTS

This work is supported by NSF CCF-1217928, CCF-1017633, NSF CCF-0916139, NSF CCF-1218129, NSF CNS-0959924, and NSF CCF-1016868. We warmly thank our ICSM reviewers whose comments helped us improve the quality of this paper.

REFERENCES

[1] A. Abadi, M. Nisenson, and Y. Simionovici. A traceability technique for specifications. In *Proceedings of the 16th IEEE ICPC*, ICPC '08, pages 103–112, Washington, DC, USA, 2008. IEEE Computer Society.

[2] N. Anquetil and T. C. Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *CASCON*, page 4, 1998.

[3] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE TSE*, 28(10):970–983, 2002.

[4] G. Antoniol, G. Canfora, A. de Lucia, and G. Casazza. Information retrieval models for recovering traceability links between code and documentation. In *IEEE ICSM'00*, ICSM '00, pages 40–, Washington, DC, USA, 2000. IEEE Computer Society.

[5] G. Bavota, A. D. Lucia, R. Oliveto, A. Panichella, F. Ricci, and G. Tortora. The role of artefact corpus in lsi-based traceability recovery. In *7th TEFSE'13*, page to appear, San Francisco, California, May 19, 2013.

[6] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster. The concept assignment problem in program understanding. In *ICSE*, pages 482–498, 1993.

[7] D. Blei and J. Lafferty. *Text mining: Theory and Applications, chapter Topic Models*, chapter 5, pages 365–452. Taylor and Francis, London, 2009.

[8] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[9] J. Chang and D. M. Blei. Hierarchical relational models for document networks. *Annals of Applied Statistics*, 4(1):124–150, 2010.

[10] S. Chatterjee, S. Juvekar, and K. Sen. Sniff: A search engine using free-form queries. In *FASE*, pages 385–400, 2009.

[11] J. Cleland-Huang, B. Berenbach, S. Clark, R. Settimi, and E. Romanova. Best practices for automated traceability. *IEEE Computer*, 40(6):27–35, 2007.

[12] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, G. O., J. Huffman Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder. Grand challenges, benchmarks, and tracelab: Developing infrastructure for the software traceability research community. In *6th TEFSE2011*, Honolulu, HI, USA, May 23, 2011.

[13] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regu-

latory codes to product specific requirements. In *32nd ACM/IEEE ICSE'10*, pages 155–164, 2010.

[14] J. Cleland-Huang, Y. Shin, E. Keenan, A. Czauderna, G. Leach, E. Moritz, M. Gethers, D. Poshyvanyk, J. H. Hayes, and W. Li. Toward actionable, broadly accessible contests in software engineering. In *34th IEEE/ACM ICSE'12 NIER Track*, pages 1329–1332, Zurich, Switzerland, June 2-9, 2012.

[15] I. Consulting. Iag consulting business analysis benchmark. In *IAG*, 2009.

[16] A. Czauderna, M. Gibiec, G. Leach, Y. Li, Y. Shin, E. Keenan, and J. Cleland-Huang. Traceability challenge 2011: Using tracelab to evaluate the impact of local versus global idf on trace retrieval. In *6th TEFSE'11*, volume 6, Honolulu, HI, USA, 2011.

[17] A. De Lucia, M. Di Penta, and R. Oliveto. Improving source code lexicon via traceability and information retrieval. *IEEE TSE*, page (to appear), 2010.

[18] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM TOSEM*, 16(4), 2007.

[19] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[20] F. Deissenboeck and M. Pizka. Concise and consistent naming. *Software Quality Journal*, 14(3):261–282, 2006.

[21] U. Dekel and J. D. Herbsleb. Improving api documentation usability with knowledge pushing. In *31st IEEE/ACM International Conference on Software Engineering (ICSE'09)*, pages 320–330, 2009.

[22] U. Dekel and J. D. Herbsleb. Reading the documentation of invoked api functions in program comprehension. In *17th IEEE ICPC'09*, pages 168–177. IEEE, 2009.

[23] B. Dit, E. Moritz, and D. Poshyvanyk. A tracelab-based solution for creating, conducting, and sharing feature location experiments. In *20th IEEE ICPC'12*, pages 203–208, Passau, Germany, June 11-13, 2012.

[24] B. Dit, A. Panichella, E. Moritz, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. Configuring topic models for software engineering tasks in tracelab. In *7th TEFSE'13*, page to appear, San Francisco, California, May 19, 2013.

[25] R. Dömges and K. Pohl. Adapting traceability environments to project-specific needs. *Commun. ACM*, 41(12):54–62, Dec. 1998.

[26] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Commun. ACM*, 30(11):964–971, 1987.

[27] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia. On integrating orthogonal information retrieval methods to improve traceability link recovery. In *27th IEEE ICSM'11*, pages 133–142, 2011.

[28] M. Gibiec, A. Czauderna, and J. Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *25th IEEE/ACM ASE'10*, pages 245–254, Antwerp, Belgium, 2010. ACM.

[29] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *RE*, pages 94–101, 1994.

[30] M. Grechanik, K. M. Conroy, and K. Probst. Finding relevant applications for prototyping. In *MSR*, page 12, 2007.

[31] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. M. Cumby. A search engine for finding highly relevant applications. In *ICSE (1)*, pages 475–484, 2010.

[32] M. Grechanik, K. S. McKinley, and D. E. Perry. Recovering and using use-case-diagram-to-source-code traceability links. In *ESEC/SIGSOFT FSE*, pages 95–104, 2007.

[33] E. Hull, K. Jackson, and J. Dick. *Requirements Engineering*. SpringerVerlag, 2004.

[34] R. Jacobs. Methods for combining experts' probability assessments. *Neural Computation*, 7(5):867–888, 1995.

[35] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. H. Hayes, A. Dekhtyar, D. Manukian, S. Hussein, and D. Hearn. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *34th IEEE/ACM ICSE'12*, pages 1375–1378, Zurich, Switzerland, June 2-9, 2012.

[36] D. Lawrie and D. Binkley. Expanding identifiers to normalize source code vocabulary. In *27th IEEE ICSM'11*, pages 113–122, Williamsburg, Virginia, USA, September 25-30, 2011. IEEE Computer Society.

[37] A. Marcus, J. Maletic, and A. Sergeyev. Recovery of traceability links between software documentation and source code. *International Journal of Software Engineering and Knowledge Engineering*, 15(4):811–836, 2005.

[38] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *ICSE*, pages 125–137, 2003.

[39] C. McMillan, M. Grechanik, and D. Poshyvanyk. Detecting similar software applications. In *ICSE*, pages 364–374, 2012.

[40] S. Mizzaro. Relevance: The whole history. *JASIS*, 48(9):810–832, 1997.

[41] S. Mizzaro. How many relevances in information retrieval? *Interacting with Computers*, 10(3):303–320, 1998.

[42] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. On the equivalence of information retrieval methods for automated traceability link recovery. In *18th IEEE ICPC'10*, pages 68–71, 2010.

[43] 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 *17th CSMR'13*, pages 199–208, Genova, Italy, March 5-8, 2013.

[44] F. A. C. Pinheiro and J. A. Goguen. An object-oriented tool for tracing requirements. *IEEE Software*, 13(2):52–64, 1996.

[45] R. Rapp. The computation of word associations: comparing syntagmatic and paradigmatic approaches. In *19th ICCL*, pages 1–7, Morristown, NJ, USA, 2002.

[46] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[47] J. Stylos and B. A. Myers. A web-search tool for finding API components and examples. In *IEEE Symposium on VL and HCC*, pages 195–202, 2006.

[48] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.