Using Code Ownership to Improve IR-Based Traceability Link Recovery

Diana Diaz[1], Gabriele Bavota[2], Andrian Marcus[3], Rocco Oliveto[4], Silvia Takahashi[1], Andrea De Lucia[5]
[1]Universidad de Los Andes, Bogotá, Colombia
[2]University of Sannio, Benevento, Italy
[3]Wayne State University, Detroit, USA
[4]University of Molise, Pesche (IS), Italy
[5]University of Salerno, Fisciano (SA), Italy
dm.diaz41@uniandes.edu.co, gbavota@unisa.it, amarcus@wayne.edu, rocco.oliveto@unimol.it,
stakahas@uniandes.edu.co, adelucia@unisa.it

***Abstract*—Information Retrieval (IR) techniques have gained wide-spread acceptance as a method for automating traceability recovery. These techniques recover links between software artifacts based on their textual similarity, i.e., the higher the similarity, the higher the likelihood that there is a link between the two artifacts. A common problem with all IR-based techniques is filtering out noise from the list of candidate links, in order to improve the recovery accuracy. Indeed, software artifacts may be related in many ways and the textual information captures only one aspect of their relationships. In this paper we propose to leverage code ownership information to capture relationships between source code artifacts for improving the recovery of traceability links between documentation and source code. Specifically, we extract the author of each source code component and for each author we identify the "context" she worked on. Thus, for a given query from the external documentation we compute the similarity between it and the context of the authors. When retrieving classes that relate to a specific query using a standard IR-based approach we reward all the classes developed by the authors having their context most similar to the query, by boosting their similarity to the query. The proposed approach, named TYRION (TraceabilitY link Recovery using Information retrieval and code OwNership), has been instantiated for the recovery of traceability links between use cases and Java classes of two software systems. The results indicate that code ownership information can be used to improve the accuracy of an IR-based traceability link recovery technique.**

***Index Terms*—Traceability Link Recovery, Code Ownership, Information Retrieval, Empirical Studies.**

I. Introduction

Traceability links between software artifacts play a paramount role in software comprehension, allowing to map high-level documents to low-level artifacts [1], [2]. They are also very useful during a variety of software engineering tasks, such as, testing or bug location. In some cases, process requirements (such as, regulatory requirements for safety critical systems) require developers to create and maintain traceability links throughout the life-cycle of the system. Unfortunately, in most cases traceability links are often an after thought and in such cases the traceability links need to be recovered from existing artifacts. The recovery and management of traceability links is largely a manual effort, which is person-power intensive and error prone [3]. It comes as no surprise that a lot of research has been conducted to automate at least part of this work. Since most software artifacts are made of or contain text data, most approaches proposed in the past decade for traceability link recovery and management are based on Information Retrieval (IR) [4] techniques. The idea behind such approaches is that pairs of artifacts having high textual similarity have a high probability to be related to each other [2]. A traceability recovery process can be viewed as a document retrieval task, where a source artifact is used as "query" (e.g., a use case) and another set of artifacts (e.g., source code classes) form the document space and the ones relevant to the query are retrieved by the IR tool. The pairs between the query and the retrieved documents form the list of candidate links. The software engineer analyzes such a list and determines the correct as well as the incorrect links.

IR-based techniques work well when the text used in the document space is consistent with the one used in the query artifacts. In practice, there is often a mismatch between the text used in artifacts at various abstraction levels, or in artifacts developed by different people, known as the *vocabulary mismatch problem* [5]. In consequence, a common problem with all IR-based techniques is filtering out candidate links due to accidental high textual similarities. At the same time, it is hard for such techniques to retrieve links affected by the vocabulary mismatch. Also, some artifacts are by nature less verbose (e.g., a requirement, an interface, or an abstract class) and most IR techniques would return a very low similarity value between them even if the two artifacts are related. As a result, such correct links will appear in the lower part of the list of candidate links. All these issues make the recovery of correct links time-consuming and error-prone both at the top of the candidate link list and especially at the bottom [6].

Several approaches have been proposed to overcome the limitations of IR-based traceability link recovery techniques. Some approaches focus on transformations at the vocabulary level (e.g., [7], [8]), while others leverage relationships between source code-based artifacts, based on different types of information (e.g., structural dependencies between source code [9]). Following the same research direction, in this paper we propose to leverage *code ownership* information for improving

978-1-4673-3092-3/13 © 2013 IEEE

ICPC 2013, San Francisco, CA, USA

the accuracy of IR-based techniques during documentation-to-source code traceability recovery tasks. Our conjecture is that developers generally work on specific (as opposed to opportunistic) and related functionalities during software evolution. This means that the code (e.g., classes) authored by a specific developer may be related to specific high-level artifacts (e.g., use cases). In consequence, during the recovery of traceability links between high level artifacts and source code, the code documents developed by the same author can be considered together and this cluster of document can be used to improve low textual similarities to the query, caused by vocabulary mismatch or poor verbosity. The idea to combine code ownership and textual similarities between documents was exploited in other software engineering context, i.e., bug triage, while determining the developers who should fix a bug [10], [11], [12]. In short, developer recommenders are based on the idea that if a developer authored code that is textually similar to a new bug report, she should likely be in charge with the new bug report. We adopt here the reverse idea, *if a developer authored code that is linked to a high level artifact (e.g., a use case), then other code developed by the developer would likely relate to the same artifact.*

Specifically, we extract the author of each source code artifact and for each author we define the context she worked on (*author's context*), i.e., all the code components authored by the developer. Then, for a given query from the external documentation (e.g., requirement or use case) we use an IR tool to compute the similarity between the query and all the authors' contexts to identify the author having the most similar context to the query. This author is probably the one that mainly contributed to the implementation of the functionality described by the query. Then, when retrieving classes that relate to a specific query using a standard IR-based approach we reward all the classes developed by the authors having their context most similar to the query, by boosting their similarity to the query. Our expectation is that code documents with low verbosity and those with a problematic vocabulary will rise in the ranked list of candidate links, improving their recovery and thus making their identification less time-consuming.

The proposed approach, named TYRION (**T**raceabilit**Y** link **R**ecovery using **I**nformation retrieval and code **O**w**N**ership), has been instantiated for the recovery of traceability links between use cases and Java classes in the context of a case study conducted on two software systems. The results indicate that TYRION helps in improving the traceability recovery accuracy when compared to a standard IR-based that exploits only the textual similarity between software artifacts.

Structure of the paper. Section II presents background information on an IR-based traceability recovery process and discusses the related literature, while Section III presents TYRION. Sections IV and V report the design and the results, respectively, of an empirical study aimed at evaluating TYRION. Finally, Section VII concludes the paper after a discussion of the threats that could affect the validity of the achieved results (Section VI).

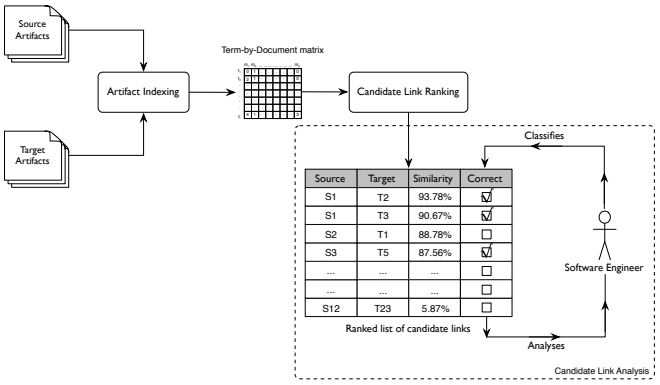

Fig. 1. An IR-based traceability recovery process

II. Background and Related Work

This section overviews the phases of an IR-based traceability recovery process and discusses the different approaches proposed in the literature.

A. IR-based Traceability Recovery Process

An IR-based traceability recovery process follows the steps depicted in Figure 1. The process starts by indexing the artifacts in the artifact corpus through the extraction of terms from their content. The indexing process of the artifacts and the construction of the artifact corpus are preceded by a text normalization phase aimed at (i) pruning out white spaces and most non-textual tokens from the text (e.g., special symbols, some numbers) and (ii) splitting source code identifiers composed of two or more words into their constituent words, e.g., *getName* is split into *get* and *name*.

A stop word function and/or a stop word list are then applied to discard common words (i.e., articles, adverbs, etc.) that are not useful to characterize the semantic of a document's content. The stop word function prunes out all the words having a length less than a fixed threshold, while the stop word list is used to cut-off all the words contained in a given list. Morphological analysis, such as stemming [13], could be also performed aiming at removing suffixes of words to extract their stems.

The output of the indexing process is represented by a $m \times n$ matrix (called *term-by-document* matrix), where m is the number of all terms that occur within the artifacts, and n is the number of artifacts 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 [4]. A widely used weighting schema (also used in this paper) is the *tf-idf* [4], which gives more importance to words having a high frequency in a document (high *tf*) and appearing in a small number of documents, thus having a high discriminant power (high *idf*).

Once the artifacts are indexed, different IR methods can be used to compare a set of source artifacts—used as "queries" by the IR method (e.g., requirements)—against another set of artifacts—considered as "documents" by the IR method (e.g., source code files)—and rank the similarity of all possible

pairs of artifacts (see Figure 1). Probabilistic models [2], Vector Space Model (VSM) [4], and Latent Semantic Indexing (LSI) [14] are the three most frequently used IR methods for traceability recovery [15].

Once the list of candidate links has been generated, it is provided to the software engineer for examination (see Figure 1). The software engineer reviews the candidate links, determines those that are correct links and discards the false positives.

B. IR Methods for Traceability Recovery

Antoniol *et al.* [2] are the first to apply IR methods to the problem of recovering traceability links between software artifacts. They use both the probabilistic and vector space models to trace source code onto software documentation. The results of the experimentation show that the two methods exhibit similar accuracy. Marcus and Maletic [16] use LSI to recover traceability links between source code and documentation. They perform case studies similar in design to those in [2] and compare the accuracy of LSI with respect to the vector space and probabilistic models. The results show that LSI performs at least as well as the probabilistic and vector space models combined with full parsing of the source code and morphological analysis of the documentation. Abadi *et al.* [17] compare several IR techniques to recover traceability links between code and documentation. They compare dimensionality reduction methods (e.g., LSI), probabilistic and information theoretic approaches, i.e., Jensen & Shannon (JS), and the standard VSM. The results achieved show that the techniques that provide the best results are VSM and JS. Recently, Latent Dirichlet Allocation (LDA) and a combination of Relational Topic Model (an extension of LDA) with other IR methods have been proposed for traceability recovery [18], [19]. While LDA provides performance comparable to that obtained by IR methods [18], the combination of different techniques outperforms stand-alone IR methods [19].

However, a recent empirical study highlighted that none of these techniques sensibly outperforms the others [20]. For this reason, most of the effort in the literature has been devoted to the definition of enhancements to improve the accuracy of the IR-based traceability methods and make the traceability recovery process less time consuming and error prone. Some improvements focus on the terms that are extracted from the artifacts, and some pre-processing techniques. Capobianco *et al.* have suggested that domain-specific terms (e.g., jargon) best describe the concepts in the code. They propose to use only the nouns from the software artifacts [21]. Other work has also adapted the weights of the artifacts' terms depending on the length of the artifacts [22], a project glossary [8], or external dictionaries [7]. Recently, smoothing filters have been shown to improve the precision of IR-based traceability [23].

We share with all the aforementioned approaches the need for improving—in various ways—results achieved by the standard IR methods, and we show that code ownership represents a valuable source of information to improve IR-based traceability recovery. In addition, since TYRION uses a different source of information as compared to all the enhancing strategies described above, it might be used to complement them.

An issue which hinders the performance of IR techniques when applied to traceability recovery is the presence of vocabulary mismatch between source and target artifacts. More specifically, if the source artifacts are written using one set of terms and the target artifacts are written using a complementary set of terms, IR techniques will have difficulties identifying links between the sets of artifacts. One way to address this problem is to enhance the textual description of software artifacts or to complement textual information with other source of information. Following the former research direction, one technique that attempts to alleviate the vocabulary mismatch has been recently proposed [24], [25]. The proposed approach uses search engines to identify a set of terms related to the query and expand the query in an attempt to improve recovery accuracy. Empirical studies indicate that using web mining to enhance queries improves retrieval accuracy.

As for the latter direction, structural information has been shown to augment IR-based methods for traceability. The structural information generally refers to the relationships in the software's source code, such as function calls, inheritance, or realization relationships [9]. These relationships are valuable for traceability recovery because the links among the source code are reflected as links among high-level artifacts (an idea known as *software reflexion* [26]). However, the accuracy of such methods is highly dependent on the IR methods. If the IR methods perform poorly, the combined approaches may perform even worse. In order to mitigate such a problem, Panichella *et al.* [27] proposed to use structural information only when the traceability links from the IR methods are verified by the software engineer and classified as correct links. An empirical evaluation conducted on three systems suggests that the supervised approach outperforms both a pure IR-based method and the unsupervised approach.

We share with these approaches the need to complement IR methods with other source of information in order to mitigate the vocabulary mismatch. What sets our work apart from previous approaches is that we use code ownership as a novel source of information and empirically demonstrate that such a source of information can be effectively used to improve the accuracy of IR-based traceability recovery techniques. Moreover, our approach is fully unsupervised.

III. The Proposed Approach: TYRION

In a given software system, which is being developed following any typical software development process, we can conjecture that the ownership of the artifacts provides information about the division of work in developers teams, in general, and the division of requirements (or use cases) to be implemented, in particular. Developers generally work on specific and related functionalities [28]. This means that the code (e.g., classes) authored by a specific developer is related to specific high-level artifacts (e.g., use cases).

In this paper, we propose to explore this information by defining a traceability recovery method that combines textual information with code ownership in order to improve the accuracy of IR-based traceability recovery methods. The proposed traceability recovery method, named TYRION (**T**raceabilit**Y** link **R**ecovery using **I**nformation retrieval and code **O**w**N**ership), is based on the following steps:

- **Step 1**. *Compute textual similarity between high-level artifacts and source code.* TYRION first computes the textual similarity between all the possible pairs of a set of high-level artifacts (e.g., use cases) and a set of target artifacts (e.g., code classes). Such a step represents the canonical IR-based traceability recovery process described in Section II-A. It is worth noting that the set of source artifacts could be also a singleton. In this case a single high-level artifact is traced on source code artifacts. The output of this step is a list of candidate links between the source and target sets of artifacts.
- **Step 2**. *Identify the author of each source code artifact.* In the second step, TYRION identifies the owner of each source code artifact. Code ownership can be derived in different way. The simplest way is to look at source code comments. For instance, in software systems written in Java the author of a class (or method) is usually specified using the Javadoc tag `@author`. Other approaches derive code ownership by analyzing the history of a software system. For instance, the owner of a source code component is the developer who performed the highest number of changes (e.g., commits) on that component [29].
- **Step 3**. *Define the author context.* In this step, TYRION estimates the context where the developer is working/worked on. This information is crucial to identify the high-level artifacts related to a specific author. To define the context of a developer, TYRION merges together (in a single document) all the source code components authored by that developer. Such a document (*author's context*) contains information (encapsulated in source code comments and identifiers) about the system functionalities implemented by the developer. Each author's context is then indexed in the corpus of high-level and source code artifacts, following exactly the same normalization phase. In this way the author context is represented on the same term space of the high-level and source code artifacts to be traced.
- **Step 4**. *Integrate code ownership information with textual information.* TYRION integrates the textual similarity between the source and target artifacts (computed using an IR technique) with code ownership information. First, for each source artifact (i.e., high-level artifact) TYRION identifies the author that most likely developed the functionality described in the high-level artifact. This information is derived by computing the textual similarity between the high-level artifact (used again as query) and the authors' contexts. The developer with the context most similar to the high-level artifact is the principal contributor to the implementation of the functionality described in the high-level artifact. Assuming that all the classes authored by a developer are in some way related to the high-level artifact, TYRION boosts the similarity (adding a bonus β) between such classes and the high-level artifact aiming at increasing the ranking of links between artifacts with low verbosity and a problematic vocabulary.

A crucial input for TYRION is the choice of the bonus β. The simplest way to define the bonus is to fix it as a constant value in the range of [0, 1], i.e., the theoretical range of values of textual similarities. However, defining a bonus *a priori* is quite difficult and rather subjective. The range of the ranked list (measured as the difference between the max and min similarity values) can differ significantly from one system to another, or when tracing different types of artifacts. Indeed, we may get a ranked list where all the similarity values are concentrated in a small interval. On the other hand, we may get a ranked list where the similarity values are spread over the entire interval [0, 1]. In the former case, a small bonus will have a large effect, while in the latter case a higher bonus is required.

For this reason, we propose an *adaptive* bonus that is proportional to the median half range of the similarity values computed for each software artifact. More precisely, we set the adaptive bonus as $\beta = median\{v_1, \ldots, v_n\}$ where a generic v_i value denotes the *half range* of the similarity values of the $i-th$ source artifact, i.e., $v_i = (max_i - min_i)/2$ where max_i and min_i are the maximum and minimum similarity values between the $i-th$ source artifact and the target artifacts.

Compared to a constant bonus, the adaptive bonus takes into account ranked list range. In addition, the adaptive bonus allows its use without the need to set any parameter, that is, the bonus definition is completely automatic and data driven.

IV. Empirical Evaluation

This section describes the design of the empirical study we conducted to evaluate TYRION. The study was conducted following the Goal-Question-Metric paradigm proposed by Basili *et al.* [30]. Raw data and working data sets are available for replication purposes[1].

A. Definition and Context

The *goal* of the evaluation was to analyze whether the accuracy of IR-based traceability recovery methods improves when combining the textual similarity computed by the IR method with code ownership information. The *perspective* is of a researcher, who wants to assess to what extent code ownership information complements textual information and improves IR-based traceability recovery methods. The *context* of our study is represented by two software systems, namely eTour and SMOS. These systems have been developed by final year Master's students at the University of Salerno (Italy) and have been used in previous empirical work on traceability link

[1] http://distat.unimol.it/reports/TYRION/

TABLE I
CHARACTERISTICS OF THE SYSTEMS USED IN THE CASE STUDY.

System	Description	KLOC	Source Artifact (#)	Target Artifact (#)	True link	Language
eTour	An electronic touristic guide developed by students	45	Use Cases (58)	Classes (174)	366	English
SMOS	A system used to monitor high school students	23	Use Cases (67)	Classes (100)	1,044	Italian

recovery [21], [27], [23]. Table I shows some characteristics of the two software systems. Other than software artifacts, each repository also contains the traceability matrix built and validated by the original developers. We consider such a matrix as the "oracle" to evaluate the accuracy of the different traceability link recovery methods. The natural language of the artifacts is English for eTour and Italian for SMOS.

The choice of these two projects is justified by several factors. The most compelling factor is to have systems with code ownership available and with traceability matrices validated by developers. The two systems address problems in different domains, with different vocabularies, hence the performance of an IR-based traceability link recovery technique may vary from a domain to another. Thus, the effect of code ownership could be more or less evident on specific contexts.

B. Instantiating TYRION

In the context of our study TYRION has been instantiated to identify links between use cases and Java classes. It is worth noting that the use of classes as source code artifacts is widely adopted in traceability [15], while method-level granularity is much more exploited in feature location tasks [31]. Changing the document granularity or the type of artifacts to use is an implementation issue. TYRION would work the same way.

As for the indexing of software artifacts, we applied both a stop word function (with threshold equal to three) and a stop word list. The stop word list we used includes the standard natural language stop words, the Java keywords (e.g., String, int, public), and some terms identified manually looking at the software artifacts (e.g., terms occurring in the artifacts' templates). Also, while sophisticated methods for identifier splitting have been proposed in the literature (e.g., [32], [33]), we use here simple conventions (i.e., camel case and under score separator) to split identifiers. This choice is due to the programming language of the object systems, i.e., Java. As indicated by a recent study, in Java systems camel case convection is widely adopted and the use of a sophisticated identified splitting is not needed [32]. We also used stemming and in particular the Porter stemmer [13]. We chose Lucene for the IR engine. Lucene combines a Boolean model (BM) [4] with VSM. Specifically, the boolean model is used to perform a preliminary filtering of the results and only the documents "approved" by BM are scored by VSM. The weighting schema adopted in the VSM is the *tf-idf*. Note that the use of the code ownership to complement textual similarity is independent of the IR method and can be integrated in any IR-based traceability recovery tool.

Finally, TYRION needs to identify owners for each Java class of the object systems. Since these systems are written in Java and they do not have version history data, we extract code ownership by analyzing the Javadoc comments.

C. Research Questions

In the context of our study, we formulated the following research question:

> **RQ**$_1$: *Can code ownership information be used to complement textual information and improve IR-based traceability recovery methods. Specifically, does TYRION obtain better precision and recall than a standard IR-based traceability recovery?*

In order to respond to this research question, we compared the accuracy of a Lucene-based traceability recovery method with and without the use of code ownership information. Thus, the two methods compared in our study are: (1) a Lucene-based traceability recovery method (which we call the *baseline*) and (2) TYRION (i.e., Lucene augmented with code ownership information). We provided to the two traceability recovery methods identical term-by-document matrices in order to eliminate any bias towards either IR method.

We applied both techniques on a massive traceability link recovery task [15], where a set of source artifacts is traced on a set of target artifacts. In the context of our study, we traced all the use cases on all the classes of the two objects systems. Thus, we obtained four (two systems x two recovery methods) different ranked lists of candidate links.

Once obtained a list of candidate links, we used a tool that takes as input such a list and classifies each link as correct link or false positive, simulating the work behavior of the software engineer when she classifies the proposed links. Such a classification of the candidate links is performed on the basis of the original traceability matrix provided by the developers. For each ranked list analyzed, the classification process starts from the top of the ranked list and stops when all correct links are recovered.

D. Metrics

To evaluate the different traceability recovery methods, we use two well-known IR metrics [4]:

$$recall = \frac{|correct \cap retrieved|}{|correct|}\%$$

$$precision = \frac{|correct \cap retrieved|}{|retrieved|}\%$$

where $correct$ represents the set of correct links and $retrieved$ is the set of all links retrieved by the traceability recovery technique (correct or otherwise).

A common way to evaluate the performance of retrieval methods consists of comparing the precision values obtained at different recall levels. This result is a set of recall/precision points which are displayed in precision/recall graphs.

We confirmed our results by using a test for statistical significance. We used this test to verify that the number of false

positives retrieved by one method is statistically significantly lower than the number of false positives retrieved by another method. The dependent variable of our study is represented by the number of false positives retrieved by the traceability recovery method for each correct link identified. Since the number of correct links is the same for each traceability recovery activity (i.e., the data were paired), we used the Wilcoxon Rank Sum test [34] to test the following null hypothesis:

> *There is no statistically significant difference between the number of false positives retrieved by the baseline and TYRION*

Basically, we pairwise compare the difference between the number of false positives one has to analyze—for each true positive link—when using code ownership information and when not. We use a one-tailed test as we are interested to test whether TYRION introduces significant reduction in the number of false positives. Results are interpreted as statistically significant at $\alpha = 5\%$. We apply a non-parametric statistical procedure (Wilcoxon Rank Sum test) because, after using a normality test (Shapiro-Wilk) on all data sets involved in the study, such a test indicated a significant deviation from normality (p-value$<$0.01).

We also estimated the magnitude of the difference between the false positives. We used the Cliff's Delta (or d) [35], a non-parametric effect size measure for ordinal data. The effect size is considered small for $d < 0.33$, medium for $0.33 \leq d < 0.474$ and large for $d \geq 0.474$ [35]. We chose the Cliff's Delta d effect size because it is appropriate for our variables and given the different levels (small, medium, large) defined for it, it is quite easy to be interpreted.

We are aware that with our statistical analysis we are considering only precision at various levels of recall. However this is a quite consolidated practice in IR, e.g., there exist a metric obtained by averaging precision across different levels of recall [4], and such a metric has been used in previous traceability recovery work [24], [25].

V. Analysis of the Results

This section discusses the results of our empirical study, aimed at answering the research question stated in Section IV-C. Specifically, we first quantitatively discuss the results and then we qualitative analyze them.

A. Quantitative Analysis

Figure 2 reports the precision/recall curves obtained by Lucene, i.e., the baseline approach (gray line), and by TYRION (black line) on eTour (a) and SMOS (b). In addition, Table II reports the improvement of precision and the reduction of retrieved false positives provided by TYRION (with respect to the baseline approach) at different levels of recall.

From the analysis of Figure 2 and Table II it is evident that TYRION provides significant improvements in terms of precision when the recall is between 30% and 70%. At these recall values we observe an improvement of precision ranging from 7% to 17% on eTour and from 10% to 26% on SMOS.

TABLE III
Wilcoxon test (p-value), descriptive statistics of differences between false positives (FP), and Cliff's effect size (d).

Data set	p-value	TYRION$FP - baseline$FP			d
		mean	median	st. dev.	
eTour	$<$**0.0001**	-267	-194	271	0.12
SMOS	$<$**0.0001**	-586	-629	447	0.22

The largest improvement of precision (+26%) represents the maximum benefit provided by TYRION in our study and it was reached on SMOS when the recall is 50%.

Such an improvement mirrors a considerable reduction of retrieved false positives going from a modest 8% (on eTour at 20% recall) up to 209% (on SMOS at 50% recall). Such a result indicates a notable improvement for the end user. For example, the end user of a traceability recovery tool built around the baseline approach should discard 1,384 false positives on SMOS to retrieve 522 correct links (that is, 50% of recall). On the other hand, by using TYRION, the number of false positives to discard in order to reach 50% recall falls to 447 (-209%), saving a lot of effort to the end user.

Overall, we can observe that TYRION always provides better performances than the baseline approach, except at recall 100%, where the two techniques are almost equivalent. This result confirms that when the goal is to recover all correct links there is an upper bound to the performance improvements that is very difficult to overcome [36], [37].

These findings are also supported by our statistical analysis. Table III reports results of the Wilcoxon test (p-value) together with the descriptive statistics of the false positive differences and Cliff's d effect size. The Wilcoxon test indicates that the number of false positives retrieved by TYRION is significantly lower than the number of false positives retrieved by the baseline method. On average, by considering all possible levels of recall (i.e., the recall is computed each time a correct link is retrieved), TYRION retrieves 194 and 629 fewer false positives on eTour and on SMOS, respectively, when compared to the baseline approach.

Despite this result, the d effect size is small on both systems. This is likely due to the very high number of recall levels considered (i.e., one for each correct link, leading to 1,044 levels on SMOS, and 366 levels on eTour). At many recall levels, especially when the recall is particularly low ($<$10%) or particularly high ($> 90\%$), the performance of TYRION is comparable to that of the baseline method. Indeed, by only considering the levels of recall where TYRION shows substantial improvements (i.e., between 20% and 80% - see Figure 2), the effect size becomes medium on eTour (0.33) and large on SMOS (0.59).

Overall, the results indicate a practical evidence of the improvement obtained by considering code ownership when retrieving traceability links. Such an improvement is particularly evident when the recall is between 20% and 80%.

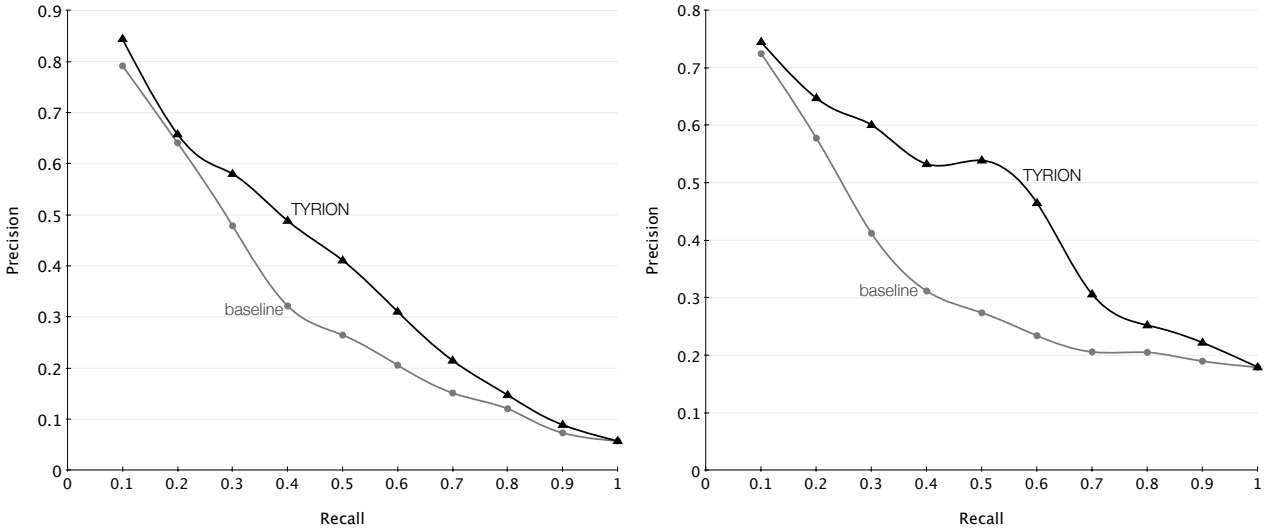

Fig. 2. Precision/Recall curves achieved on the eTour (a) and the SMOS (b) repository.

TABLE II
PERCENTAGE OF PRECISION IMPROVEMENT AND REDUCTION OF NUMBER OF FALSE POSITIVES AT DIFFERENT LEVEL OF RECALL.

Data set	Rec(20%)		Rec(40%)		Rec(60%)		Rec(80%)		Rec(100%)	
	Prec	FP	Prec	FP	Prec	FP	Prec	FP	Prec	FP
eTour	+2%	-8%	+17%	-51%	+11%	-46%	+3%	-26%	0%	0%
SMOS	+7%	-34%	+22%	-151%	+23%	-184%	+5%	-31%	0%	-1%
Overall	+5%	-21%	+20%	-101%	+17%	-135%	+4%	-29%	0%	0%

B. Qualitative Analysis

In addition to the quantitative analysis, we also performed a qualitative analysis of the results in order to better understand where did TYRION's benefits came from. Specifically, we compared the ranked lists of eTour and SMOS retrieved by using the baseline and TYRION.

Figure 3(a) shows—through a relation diagram—the positive effect of TYRION on the eTour ranked list of suggested links, while Figure 3(b) reports the same information for the SMOS system. The relation diagrams show the precision which each correct link is recovered with. It can be noticed that the higher the precision, the higher the position in the ranked list of the correct link. Thus, the relation diagram visually indicates the effect of TYRION on the list of candidate links.

As we can see, TYRION tends to increase the rank of correct links facilitating their identification. Also, in the middle part of the ranked lists (for both eTour - Figure 3(a) and SMOS - Figure 3(b)) it is possible to observe a very stable trend in the increase of the correct links ranks, while in the bottom part of the ranked lists, even if some improvements are visible, not all correct links are pushed up.

Some interesting correct links for which TYRION provides a strong rank improvement in the eTour system are those related to the advertisement management subsystem. This subsystem provides features to allow the restaurants registered to the system to insert advertisements. Such advertisements are shown to the tourists when they are near by them. This subsystem has been mainly implemented by the developer *Fabio P.* As a result, all use cases describing the features of this subsystem exhibit their maximum similarity with the context of this developer. By using this information, TYRION is able to retrieve in the top part of the ranked list several correct links related the advertisement management, by improving their position in the ranked list as compared to the baseline approach. This is particularly evident in Figure 3(a) from the analysis of the three links pushed up from a precision of 80%, 67%, and 50%, respectively, to a precision of 100%. Those three correct links relate the use cases *Insert new banner*, *Update existing banner*, and *Delete existing banner* to the class `AdvertisementManagement`, and they are retrieved by TYRION in position one, two, and three of the ranked list, respectively.

We also noticed several improvements in the ranking of correct links having interfaces as target artifact. For example, the rank of the link between the use case *Update existing banner* and the interface `IAdvertisementManagement` improved by 2,259 positions, the one between *Update existing tourist* and `IDBtourist` improved by 997, between *Insert new tourist* and `IDBtourist` by 984, between *Insert new Restaurant* and `IRestaurantManagement` by 856, and so on. This result is likely due to the fact that interfaces are usually not verbose and thus the standard IR-method returns

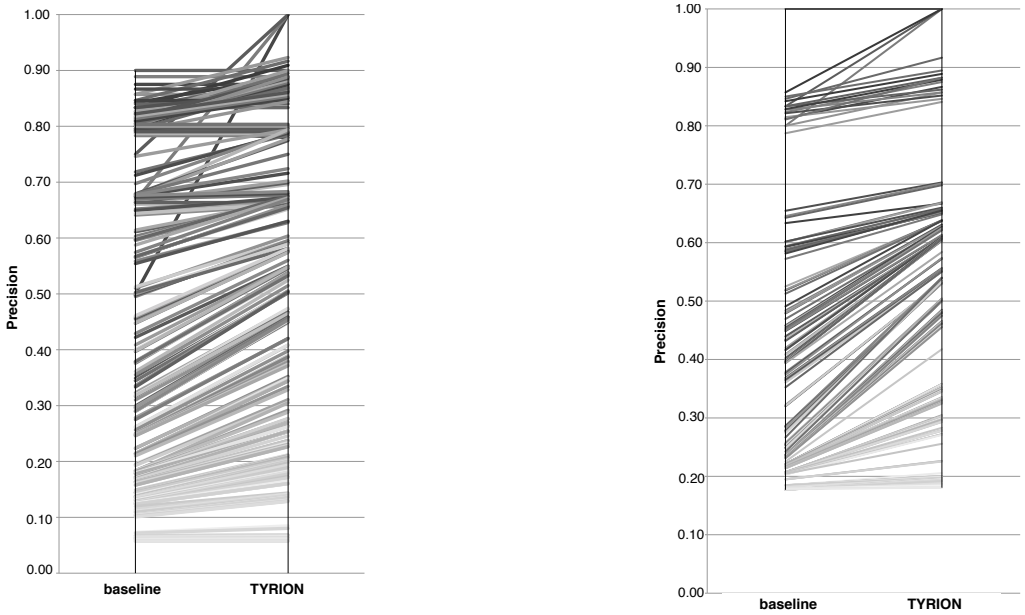

Fig. 3. Effect of TYRION on the ranking of correct links in the list of candidate links for eTour (a) and SMOS (b).

a very low similarity value for them even for the use cases they are related to. On the other side, TYRION exploits the knowledge about the authors of those interfaces to increase the similarity between them and the use cases more related to the authors' contexts.

Concerning the SMOS dataset (see Figure 3(b)), we also observed several improvements in the ranking of correct links when using TYRION. Interesting are the cases involving the set of classes implementing features related to the management of users. All the links related to those classes showed very strong improvements in their ranking. For example, the classes related to the use case *Delete user* are `User`, `UserListItem`, `ServletShowUserForm`, and `ServletDeleteUser`. All these classes are implemented by the same developer, i.e., *Vincenzo N*. He is also the one having the most similar context to the use case *Delete user*. Using such information, TYRION is able to increase the ranking of *ServletShowUserForm*, *User*, and *UserListItem* by 4,704 positions, and the ranking of *ServletDeleteUser* by 4,690 positions. It is worth noting that such an improvement probably makes the recovery of such links feasible for the software engineer, otherwise it is likely that these links are at the bottom part of the ranked list and are excluded from the analysis by the software engineer [38].

C. Summary of the Results

The quantitative and qualitative analysis of the results highlights the positive influence of considering code ownership when retrieving traceability links between high-level documentation and source code. In particular, TYRION (as instantiated in our study) exhibits much better performance than a standard Lucene-based approach, allowing improvements of precision up to 26% with a reduction of false positive links to analyze up to 209%. While in the top-part and in the very bottom part of the ranked list the accuracy of TYRION is comparable with the Lucene-based approach, TYRION produces significant improvements in terms of precision for recall levels between 20% and 80%. Specifically, TYRION proves to be very effective in increasing the rank of correct links involving artifacts that are not verbose, e.g., interfaces.

The results allow us to answer positively our research question. In particular, *code ownership information represents a useful source of information and it can be used to complement textual information in order to improve IR-based traceability recovery methods. Specifically, in the context of our study, TYRION provides more accurate list of candidate links than a standard IR-based traceability recovery technique.*

VI. Threats to Validity

This section discusses the threats that might affect the validity of our results. A relevant *external* threat is related to the repositories used in the empirical study. They are software systems implemented by final year Master's students at the University of Salerno (Italy), thus they are hardly comparable to real industrial projects. However, they are comparable to repositories used by other researchers [2], [39], [40] and one of them, i.e., eTour, has been used as benchmark repositories in the 2011 edition of the traceability recovery challenge organized at TEFSE[2]. Nevertheless, we are planning to replicate the experiment using other artifact repositories in order to corroborate our findings.

Concerning the *internal* validity, several factors may affect our results. First, choosing the right values for the similarity

[2]http://www.cs.wm.edu/semeru/tefse2011/Challenge.htm

bonus is a critical issue. We proposed to use an adaptive bonus that is proportional to the size of the ranked list. We are aware that other heuristics might be used to define the bonus. However, we decided to use the adaptive bonus since it frees the software engineer from tuning parameters required by other heuristics (e.g., fixed bonus). Future work will be devoted to define different strategies and compare them through a rigorous empirical analysis.

The approach proposed in this paper (TYRION) exploits code ownership information. In our experiment we derived such information using comments in the source code. Specifically, we identified the Javadoc tag `@author` in order to identify who is the author of the class. The identification of code ownership could be based also on other sources of information, such as, commits in the version control system [29]. Unfortunately, our object systems do not have a version control repository, thus the only way to derive code ownership is the one used in our experimentation. Future work will be devoted to replicate the experiment on other systems having version control information. This will also allow us to define other heuristics to identify code ownership and analyze how such heuristics affect the accuracy of TYRION.

Concerning the threats to the *construct* validity, we adopted two widely used metrics for assessing IR techniques, namely recall and precision. Moreover, the number of false positives retrieved by a traceability recovery tool for each correct link retrieved reflects well its retrieval accuracy.

Finally, as for the *conclusion* validity attention was paid to not violate assumptions made by statistical tests. Whenever conditions necessary to use parametric statistics did not hold, we used non-parametric tests, in particular the Wilcoxon test for paired analysis. In addition, we used the Wilcoxon test to compare the cumulative number of false positives for each correct links. Alternatively, we could have evaluated the cumulative number of correct links and false positives for each retrieved link. This allows to compare the precision of two methods for the same level of recall, thus using a single dependent variable for the statistical analysis. However, this type of analysis would have required the use of two different dependent variables, thus making the statistical analysis more complex. Moreover, the analysis of the precision/recall curves did not give contrasting results.

VII. Conclusion and Future Work

Maintaining a list of up-to-date traceability links inevitably becomes an overwhelming, error-prone task. Semi-automated tools for traceability recovery offer an opportunity to reduce this manual effort and increase productivity. Promising results have been achieved using IR methods to identify candidate links between software artifacts on the basis of their textual similarity. However, IR-based techniques are able to correctly identify relationships between software artifacts when developers consistently use terms in those artifacts. Unfortunately, in practice, there is often a mismatch between the terms used in artifacts at various abstraction levels, or in artifacts developed by different people, known as the vocabulary mismatch problem. Also, some artifacts are by nature poor verbose. This inhibits IR techniques to discriminate between relevant and irrelevant artifacts, in such cases.

Several approaches have been proposed to mitigate such issues. A common solution is represented by the integration of other source of information (such as dependencies between source code components) with textual information aiming at improving the accuracy of IR-based methods.

Following the same direction, in this paper we have proposed to leverage code ownership information for improving the recovery of traceability links between documentation and source code. The proposed approach, TYRION, extracts the author of each source code element and for each author identifies the topic on what she worked on. Thus, for a given query from the external documentation TYRION computes the similarity between it and the topic of each author. When retrieving classes that relate to a specific artifact, TYRION uses the *author-to-query similarity* to increase the textual similarity between the query and all the classes developed by the authors of the classes.

TYRION has been instantiated for the recovery of traceability links between use cases and Java classes of two software systems. In the context of our study, TYRION provides more accurate list of candidate links than a standard IR-based traceability recovery technique. It proves to be especially effective for retrieving code artifacts that are naturally not verbose, such as, interfaces. These results indicate that code ownership information represents a useful source of information and it can be used to complement textual information aiming at improving IR-based traceability recovery methods.

Future work will be devoted to further experiment and assess the proposed traceability recovery method. Replication in different contexts and with different objects is the only way to corroborate our findings. Moreover, there are a number of directions to improve the accuracy of the proposed method. One direction aims at considering a more sophisticated approach to extract code ownership (for instance, by mining version control systems). We also plan to explicitly consider the case where more than one developer contributed to the implementation of a source code component. To this aim, the analysis of the developer communication network could provide important insights in identifying latent relationships between artifacts.

A second direction is related to the identification of author context. We plan to use more sophisticated analysis, such as LDA, in order to derive a set of "topics" (instead of a single document) related to the developer and improve the identification of the contributors to the implementation of specific functionalities. Finally, we plan to investigate other approaches for combining code ownership with textual information.

Acknowledgements

This work was supported in part by grants from the National Science Foundation (CCF-0845706 and CCF-1017263) and the Francisco Jose de Caldas fellowship from COLCIENCIAS(Colombia) under conv. 528 - 2011.

REFERENCES

[1] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. of 1st International Conference on Requirements Engineering*, 1994, pp. 94–101.

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

[3] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE Transactions on Software Engineering*, vol. 27, pp. 58–93, 2001.

[4] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[5] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary problem in human-system communication: an analysis and a solution," *Communications of the ACM*, vol. 30, no. 11, pp. 964–971, 1987.

[6] A. De Lucia, R. Oliveto, and G. Tortora, "IR-based traceability recovery processes: an empirical comparison of "one-shot" and incremental processes," in *Proceedings of 23rd International Conference on Automated Software Engineering*, 2008, pp. 39–48.

[7] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proceedings of the 11th IEEE International Requirements Engineering Conference*, 2003, pp. 138–147.

[8] X. Zou, R. Settimi, and J. Cleland-Huang, "Term-based enhancement factors for improving automated requirement trace retrieval," in *Proceedings of International Symposium on Grand Challenges in Traceability*, 2007, pp. 40–45.

[9] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Proceedings of the International Workshop on Traceability in Emerging Forms of Software Engineering*, 2009, pp. 41–48.

[10] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, p. 10, 2011.

[11] G. Canfora and L. Cerulo, "Supporting change request assignment in open source development," in *Proceedings of the 2006 ACM Symposium on Applied Computing*. Dijon, France: ACM Press, 2006, pp. 1767–1772.

[12] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference*. Szeged, Hungary: ACM Press, 2011, pp. 365–375.

[13] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[14] 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*, vol. 41, no. 6, pp. 391–407, 1990.

[15] A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, *Software and Systems Traceability*. Springer Press, 2012, ch. Information Retrieval Methods for Automated Traceability Recovery.

[16] A. Marcus, J. I. Maletic, and A. Sergeyev, "Recovery of traceability links between software documentation and source code," *International Journal of Software Engineering and Knowledge Engineering*, vol. 15, no. 5, pp. 811–836, 2005.

[17] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proceedings of the 16th IEEE International Conference on Program Comprehension*, 2008, pp. 103–112.

[18] H. U. Asuncion, A. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010, pp. 95–104.

[19] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*. Williamsburg, VA: IEEE Press, 2011, pp. 133–142.

[20] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *Proceedings of the 18th International Conference on Program Comprehension*, 2010, pp. 68–71.

[21] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery via noun-based indexing of software artifacts," *Journal of Software: Evolution and Process*, 2013.

[22] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. De Palma, "Supporting software evolution through dynamically retrieving traces to UML artifacts," in *Proceedings of 7th IEEE International Workshop on Principles of Software Evolution*, 2004, pp. 49–54.

[23] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery using smoothing filters," in *Proceedings of the 19th International Conference on Program Comprehension*, 2011, pp. 21–30.

[24] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proc. of ICSE*, 2010, pp. 155–164.

[25] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Proc. of ASE*, 2010, pp. 245–254.

[26] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, 2001.

[27] 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 *Proceedings of the 17th European Conference on Software Maintenance and Reengineering*. Genova, Italy: IEEE CS Press, 2013.

[28] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, ser. SIGSOFT '08/FSE-16. New York, NY, USA: ACM, 2008, pp. 24–35.

[29] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. T. Devanbu, "Don't touch my code!: examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering and 13rd European Software Engineering Conference*. Szeged, Hungary: ACM Press, 2011, pp. 4–14.

[30] V. Basili, G. Caldiera, and D. H. Rombach, *Encyclopedia of Software Engineering*. John Wiley and Sons, 1994, ch. The Goal Question Metric Paradigm, pp. 528–532.

[31] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[32] L. Guerrouj, M. Di Penta, G. Antoniol, and Y.-G. Guéhéneuc, "Tidier: an identifier splitting approach using speech recognition techniques," *Journal of Software Maintenance and Evolution: Research and Practice*, p. in press, 2011.

[33] D. Lawrie and D. Binkley, "Expanding identifiers to normalize source code vocabulary," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*. Williamsburg, VA, USA: IEEE CS Press, 2011, pp. 113 –122.

[34] W. J. Conover, *Practical Nonparametric Statistics*, 3rd ed. Wiley, 1998.

[35] R. J. Grissom and J. J. Kim, *Effect sizes for research: A broad practical approach*, 2nd ed. Lawrence Earlbaum Associates, 2005.

[36] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *Proceedings of 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 299–309.

[37] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, p. 13, 2007.

[38] A. De Lucia, R. Oliveto, and G. Tortora, "Assessing IR-based traceability recovery tools through controlled experiments," *Empirical Software Engineering*, vol. 14, no. 1, pp. 57–93, 2009.

[39] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods." *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.

[40] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of 25th International Conference on Software Engineering*, 2003, pp. 125–135.