Improving IR-based Traceability Recovery via Noun-based Indexing of Software Artifacts

Giovanni Capobianco¹, Andrea De Lucia², Rocco Oliveto¹, Annibale Panichella²,
Sebastiano Panichella³

¹University of Molise, Contrada Fonte Lappone, 86090 Pesche (IS), Italy
²University of Salerno, Via Ponte don Melillo, 84084 Fisciano (SA), Italy
³University of Sannio, Palazzo ex Poste, Via Traiano, 82100 Benevento, Italy
giovanni.capobianco@unimol.it, adelucia@unisa.it, rocco.oliveto@unimol.it, apanichella@unisa.it,
spanichella@unisannio.it

SUMMARY

One of the most successful applications of textual analysis in software engineering is the use of Information Retrieval (IR) methods to reconstruct traceability links between software artifacts. Unfortunately, due to the limitations of both the humans developing artifacts and the IR techniques any IR-based traceability recovery method fails to retrieve some of the correct links, while on the other hand it also retrieves links that are not correct. This limitation has posed challenges for researchers that have proposed several methods to improve the accuracy of IR-based traceability recovery methods by removing the “noise” in the textual content of software artifacts (e.g., by removing common words or increasing the importance of critical terms). In this paper we propose a heuristic to remove the “noise” taking into account the linguistic nature of words in the software artifacts. In particular, the language used in software documents can be classified as a technical language, where the words that provide more indication on the semantics of a document are the nouns. The results of a case study conducted on five software artifact repositories indicate that characterizing the context of software artifacts considering only nouns significantly improves the accuracy of IR-based traceability recovery methods. Copyright © 2010 John Wiley & Sons, Ltd.

Received …
1. INTRODUCTION

The rapid development of software engineering methods and tools and the increasing complexity of software projects have led in the last decades to a significant production of textual information contained in structured and unstructured project artifacts. As consequence, a lot of effort in the software engineering community (both research and commercial) has been devoted to the analysis of textual information contained in the artifacts of software project repositories to support activities such as impact analysis [1], clone detection [2], feature location [3, 4], to define new cohesion and coupling metrics [5, 6], to assess software quality [7, 8, 9, 10].

One of the most successful applications of textual analysis consists of using Information Retrieval (IR) [11, 12] methods to recover traceability links between software artifacts (see e.g., [13, 14, 15]). The applicability of IR-based traceability recovery is not limited to requirements artifacts that are mainly expressed in natural language, but also to other types of semi-formal or formal artifacts, including UML diagrams and even source code, provided that programmers use meaningful domain terms to define source code identifiers [13, 14]. An IR-based traceability recovery process indexes all the artifacts in the repository by extracting information about the occurrences of terms (words) within them. The extracted information is used to compare a set of source artifacts used as a query (e.g., requirements) against another set of artifacts (e.g., source code). Then, the tool ranks the textual similarity of all possible pairs of source and target artifacts. Pairs having a similarity above a certain threshold (fixed by the software engineer), or being in the topmost positions of the ranked list are candidate to be linked (candidate links). Unfortunately, the set of retrieved links does not in general coincide with the set of correct links between the artifacts in the repository because of the limitations of both the humans developing artifacts and the IR techniques. Indeed, any IR method will fail to retrieve some of the correct links, while on the other hand it will also retrieve links that are not correct (false positives). Clearly, the retrieval of too many false positives makes the traceability
recovery process a tedious task, as the software engineer has to spend much more time to discard false positives than to trace correct links.

One of the main challenges in traceability management is to improve the performances of traceability recovery methods by reducing the number of false positives. A common way to reach such a goal is to reduce the “noise” various types of artifacts carry on. For example, natural language contains common words that have to be removed [11] or could contain critical terms and phrases that can be regarded as more meaningful in identifying traceability links [16, 17, 18, 19].

In this paper, we present a simple way to reduce the number of false positives retrieved by IR-based traceability recovery methods. The proposed heuristic is based on the observation that the language used in software documents can be classified as technical language\(^1\), where the terms that provide more information on the semantics of a document are the nouns, while the verbs tend to play a connection role and have a generic semantics [20, 21]. For this reason, we propose to act on the artifact indexing process taking into account only the nouns contained in the contents when extracting information about the occurrences of terms (words) within them. The proposed approach was applied to a probabilistic model, namely the Jenson-Shannon (JS) method [22], and two vector space based methods, namely Vector Space Model (VSM) [11] and Latent Semantic Indexing (LSI) [12]. In the context of our empirical study such methods were used to recover traceability links between different types of artifacts of five artifact repositories written in different languages (Italian and English languages) and that contain software artifacts at different abstraction levels (e.g., use cases, test cases, and source code). The results achieved indicated that, in general, the proposed approach improves the accuracy of all the experimented IR methods.

This paper extends our previous conference paper [23] where we presented the approach and its preliminary evaluation. The specific contributions of this paper with respect to our previous paper can be summarized as follows:

\(^1\)The language used by people who work in a particular area or who have a common interest [20, 21].
• a more detailed description of the proposed heuristic and the underlying rationale behind it (i.e., the use of linguistics);
• a more extensive evaluation of the proposed approach, aiming at improving the generalizability of the results. In particular, the proposed approach has been experimented with: (i) the Vector Space Model (VSM), in addition to the Latent Semantic Indexing (LSI) and the Jensen-Shannon (JS) methods, experimented in our previous paper; (ii) the artifact repositories of four software systems of different types, namely eTour, Pine, MODIS, and CM1, in addition to the software system EasyClinic used in the our previous paper;
• a more extensive statistical analysis of the results, demonstrating that the new approach produces a significant improvement of the accuracy of all the experimented techniques;
• a more extensive qualitative analysis, aiming at providing a deeper explanation of the achieved results.

The rest of the paper is organized as follows. Section 2 provides background information and discusses related work, while Section 3 presents the proposed indexing process. Section 4 provides details on the design of the case study and discusses on the results achieved, while Section 5 gives concluding remarks.

2. BACKGROUND AND RELATED WORK

This section provides background notions and state of the art of IR-based traceability recovery.

2.1. IR-based Traceability Recovery Process

In general, an IR-based traceability recovery process follows the process described in Figure 1. The artifacts are first indexed aiming at identifying keywords that characterize the artifact contents. Such an extraction is performed to prune out white spaces and most non-textual tokens from the text (i.e., operators, special symbols, some numerals, etc.). Then, compound identifiers are separated into their constituent words since IR techniques may miss occurrences of concepts if identifiers are not split correctly [24]. To split compound identifiers, most existing automatic software analysis tools...
rely on coding conventions [13] or on more sophisticated natural language processing methods (see e.g., [24, 25, 26, 27, 28]). In our work we used simple coding conventions to split identifiers.

During artifact indexing a stop word function and/or a stop word list are also applied to discard common words (i.e., articles, adverbs, etc) that are not useful to capture the semantics of the artifact 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 word list. Stop word lists are language specific, for example, English has different stop words than Italian. Generally, good results are achieved using both the stop word functions and lists [11, 29]. In this work, we applied both stop word functions (with threshold equal to three) and lists for English and Italian languages. The used stop word lists included, other than Italian or English standard stop words: (i) programming language (C/Java) keywords, (ii) recurring words in document templates (e.g., use case, requirement, or test case template) and (iii) author names.

A more complicated document pre-processing is represented by morphological analysis, like stemming. Stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form. There are several existing stemming algorithms, one of the most popular stemmers for the English language is the Porter stemmer [30].

Figure 1. An IR-based traceability recovery process.
The terms extracted from the documents are stored in a $m \times n$ matrix (called term-by-document matrix [11]), where $m$ is the number of all terms that occur within the documents (software artifacts), and $n$ is the number of documents in the repository. A generic entry $w_{i,j}$ denotes a measure of the weight (i.e., relevance) of the $i^{th}$ term in the $j^{th}$ document [11]. A widely used weighting schema is the tf-idf [11], which gives more importance to words having a high frequency in a document (high term frequency) and appearing in a small number of documents, thus having a high discriminant power (high inverse document frequency).

Based on the term-by-document matrix representation, 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). Pairs having a similarity above a certain threshold (fixed by the software engineer), or being in the topmost positions of the ranked list, are candidate to be linked (candidate links). The ranked list of candidate links is then analyzed by a software engineer, that can trace a candidate link, or classify the link as a false positive.

2.2. IR Methods used for Traceability Recovery

Several IR methods have been proposed in the literature to recover links between different types of artifacts. Comprehensive analysis of available research papers reveals that probabilistic models [13, 31], VSM [11, 29, 32], and LSI [12] are the three most frequently used IR methods for traceability recovery. Only in few cases different methods have been used to recover traceability links between different types of artifacts [33, 22, 34]. In addition, a recent empirical study highlighted that none of these techniques sensibly outperforms the others [35].

In this paper, we compare the retrieval accuracy of three IR methods, namely VSM, LSI and JS. In the VSM, artifacts are represented as vectors of terms (i.e., columns of the term-by-document matrix) that occur within artifacts in a repository [11]. The similarity between two artifacts is measured as the cosine of the angle between the corresponding vectors, which increases as more terms are shared. VSM does not take into account relations between terms. For instance, having
“automobile” in one artifacts and “car” in another artifact does not contribute to the similarity measure between these two documents. The VSM has been used to recover traceability links among requirements [16, 36, 37], requirements and source code [22, 13, 37, 15, 38], manual pages and source code [13, 15, 38], UML diagrams and source code [37, 39], test cases and source code [37], and defect reports and source code [40].

LSI [12] is an extension of VSM. It was developed to overcome the synonymy and polysemy problems, which occur with VSM [12]. In LSI the dependencies between terms and between artifacts, in addition to the associations between terms and artifacts, are explicitly taken into account. For example, both “car” and “automobile” are likely to co-occur in different artifacts with related terms, such as “motor” and “wheel”. To exploit information about co-occurrences of terms, LSI applies Singular Value Decomposition (SVD) [41] to project the original term-by-document matrix into a reduced space of concepts, and thus limit the “noise” terms may cause. Also in this case, the similarity between artifacts is measured as the cosine of the angle between the reduced artifact vectors. LSI has been used to recover traceability links between requirements [36, 37], requirements and source code [22, 37, 7, 15, 38], manual pages and source code [13, 15, 38], UML diagrams and source code [37, 7, 39], test cases and source code [37, 7, 42].

In the probabilistic model, a source artifact is ranked according to the probability of being relevant to a particular target artifact. It represents each document through a probability distribution. This means that an artifact is represented by a random variable where the probability of its states is given by the empirical distribution of the terms occurring in the artifacts (i.e., columns of the term-by-document matrix). The empirical distribution of a term is based on the weight assigned to such a term for a specific artifact. Probabilistic models have been used to recover links between requirements and UML diagrams [31], requirements and source code [43, 44, 13, 45], and manual pages and source code [13]. In this work we use a probabilistic model named Jensen-Shannon (JS) [22]. In the JS method, the similarity between two artifacts is given by a “distance” of their probability distributions measured by using the Jensen-Shannon Divergence [46]. The JS method
has been used to recover links between requirements and source code [22] and between use cases, UML diagrams, test cases, and source code [34].

2.3. Improving the Accuracy of IR-based Traceability Recovery Methods

Different enhancing strategies—acting at different steps in the process shown in Figure 1—have also been proposed to improve the accuracy of IR-based traceability recovery methods.

During the indexing process a weighting schema is applied to define the importance of a term in an artifact. The weighting schema can also take into account the document length [39] and the importance of the term for the specific domain [16, 17, 18, 19]. For the former, when collections have documents of varying lengths, longer documents tend to score higher since they contain more words and word repetitions. This effect is usually compensated by normalizing for document lengths in the term weighting method [39]. Artifacts could also contain critical terms and phrases that should be weighted more heavily than others, as they can be regarded as more meaningful in identifying traceability links. These terms can be extracted from the project glossary [17, 18, 19] or external dictionaries [16]. Such approaches generally improve the accuracy of an IR-based traceability recovery method. However, the identification of key phrases (as well as the use of external dictionaries) is much more expensive than indexing single keywords. In addition, there might be cases where the project glossary is not available. In this paper we assume that the most important source of information in software artifacts is contained in the nouns, since the language used in software documents can be classified as technical language, where nouns characterize the semantics of a document and the verbs (and other terms) tend to play a connection role and have a generic semantics. Thus, the proposed indexing strategy tries to reduce the “noise” in software artifacts by analyzing the grammatical nature of the terms in the artifact contents in a completely automated way, without any external source of information.

The term weighting can be changed according to the classification performed by a software engineer during the analysis of candidate links (feedback analysis) [36, 37]. If the software engineer classifies a candidate link as correct link, the words found in the target artifact increase their weights.
in the source artifact. Otherwise, they decrease their weights. The effect of such an alteration of the original source artifact is to “move” it towards relevant artifacts and away from irrelevant artifacts, in the expectation of retrieving more correct links and less false positives in next iterations of the recovery process. The indexing strategy proposed in this paper can be considered as a complementary technique. In addition, while feedbacks require human intervention, the proposed strategy is completely automatic.

Recently, the usage of smoothing filters has been proposed to improve the accuracy of existing traceability recovery techniques, based on VSM and LSI [47]. The proposed filter is inspired by Gaussian filters used in image processing [48] and removes the “common” information among artifacts of the same type (e.g., use cases, or source code artifacts) that does not help to characterize the artifact semantics. The results of an empirical study indicated that the usage of smoothing filters significantly improves the traceability recovery performances of the experimented IR methods. Once again, the proposed indexing strategy is a complementary technique. In particular, smoothing filters can be used after applying the proposed indexing strategy in order to further remove the “noise” in the software artifacts.

An issue that hinders the performances 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 different set of terms, it might be difficult for IR techniques to identify traceability links. Recently, a technique that attempts to alleviate such an issue has been introduced [49, 50]. The proposed approach uses the artifacts to be traced as queries for web search engines and expands the terms in the query artifacts with the terms contained in the retrieved documents before indexing the artifacts. Empirical studies [49, 50] indicated that using web mining to enhance queries improves retrieval accuracy.

The vocabulary mismatch between source and target artifacts might be also due the presence of abbreviations and acronyms in source code identifiers. In this case, identifier expansion techniques [24, 25, 26, 27, 28] might be applied in order to increase the accuracy of IR-based traceability
recovery methods. The indexing strategy proposed in this paper is complementary to approaches based on both query and identifier expansion techniques. Indeed, such techniques might also introduce noise in a set of source or target artifacts and the proposed indexing strategy can be used to mitigate such a noise.

The approach proposed in this paper employs a part of speech (POS) tagger to characterize the grammatical nature of the terms identified in the software artifacts. POS taggers, and, in general, Natural Language Processing techniques, have been successfully applied to support other software engineering tasks (see e.g., [53, 54, 55, 56, 57]). In particular, Etzkorn et al. [56] used a POS tagger to create a meaningful summary of a software module. Attribute and method names are extracted and then tagged with part-of-speech information before being combined with the parsed comments to construct a concept graph. The text associated with the internal nodes of this graph provides high-level descriptions of each corresponding concept.

Abebe and Tonella [53] used a POS tagger to extract domain concepts and relations from program identifiers. The linguistic dependencies obtained after parsing the source code are used to recognize concepts and relationships in order to automatically build an ontology. The obtained ontology is then used to improve concept location. Ontologies have also been used to identify links between documentation and source code [58]. In particular, source code and document artifacts are represented by an ontology that captures structural and semantic information conveyed in these artifacts, and thus allowing to recover the traceability links between software implementation and documentation at semantic level [58].

Based on the traceability recovery methods proposed in the literature, several tools have also been implemented. Table I classifies them according to the IR method used to recover the links, the adopted enhancing strategies, and the technology used to develop the tool. It is important to note

---

Concerning query expansion techniques, the noise might be represented by terms (added during the expansion process) with a generic semantics such as “insert”, “modify”, and “delete”. As for identifier expansion techniques, the noise might be represented by a wrong expansion of the identifier. For instance, the identifier “floc” could be automatically expanded in “feature” and “location” but also in “file” and “lines-of-code”.

---
Table I. Summary of IR-based traceability recovery tools.

<table>
<thead>
<tr>
<th>Tool name</th>
<th>IR method</th>
<th>Enhancing strategies</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADAMS Re-Trace [59]</td>
<td>LSI</td>
<td>None</td>
<td>Web-based and Eclipse plug-in</td>
</tr>
<tr>
<td>Poirot:TraceMaker [60]</td>
<td>Probabilistic model</td>
<td>Hierarchical modelling</td>
<td>Web-based</td>
</tr>
<tr>
<td>ReqAnalyst [42]</td>
<td>LSI</td>
<td>None</td>
<td>Web-based</td>
</tr>
<tr>
<td>RETRO [36]</td>
<td>VSM and LSI</td>
<td>User feedback</td>
<td>Standalone</td>
</tr>
<tr>
<td>TraceViz [61]</td>
<td>LSI</td>
<td>None</td>
<td>Eclipse plug-in</td>
</tr>
</tbody>
</table>

that among them only ADAMS Re-Trace [59] has been integrated in an artifact management system [7, 62].

3. A LINGUISTIC APPROACH FOR IR-BASED TRACEABILITY RECOVERY

In this section we describe an indexing approach to improve the accuracy of IR-based traceability recovery methods. We first analyze the linguistic properties of the language used by developers when writing software artifacts focusing on the main linguistic differences between the technical language of software documents and the language of documents used in traditional IR applications. Then, we formalize the proposed approach to make it usable in the context of traceability recovery.

3.1. Why a Linguistic Approach?

Generally, IR repositories are heterogeneous from several points of view. Documents are written by different authors that use a different pragmatic to explain concepts and notions about different matters. For example, in web pages it is possible to find documents written using different pragmatics, such as documents about sports (colloquial language), news (journalistic language), and research papers (scientific language). In other words, documents on the web are lexically, syntactically and semantically heterogeneous.

A different situation can be observed looking at the software documents. During software development the developers typically use a technical language to describe a software system.
Table II. Congruence between semantic and grammatical categories.

<table>
<thead>
<tr>
<th>Semantic element</th>
<th>Grammatical element</th>
</tr>
</thead>
<tbody>
<tr>
<td>entity</td>
<td>noun or nominal group</td>
</tr>
<tr>
<td>quality</td>
<td>adjective (in nominal group)</td>
</tr>
<tr>
<td>process</td>
<td>verb (verbal group)</td>
</tr>
<tr>
<td>circumstance (1)</td>
<td>adverb or adverbial group</td>
</tr>
<tr>
<td>circumstance (2)</td>
<td>prepositional phrase</td>
</tr>
<tr>
<td>minor process</td>
<td>preposition</td>
</tr>
<tr>
<td>relator</td>
<td>conjunction</td>
</tr>
</tbody>
</table>

(its functionalities, its architecture, and the main dependencies between system components). A technical language can be defined as “a variety of natural language dependent on a knowledge sector or on a professional context: it seems an impenetrable and unchanging language, like the language of some exclusive club, interpretable by members only” [63]. Unlike the language of the documents used in traditional IR applications, the technical language of software documents aims to be denotative and objective, to capture as faithful as possible the features of a software system without bias or emotional and cultural connotation. Even if the engineering language and the common-sense language have the same grammatical features, it is important to highlight that the engineering language is a “dedicated and designed semiotic subsystem which reconstructs certain aspects of components of human experience in a different way, in the course of opening them up to be observed, investigated and explained” [64]. Indeed, in order to describe objectively a software system, or its component, the language of software artifacts is characterized by specific and well defined linguistic choices: lexical choices, syntactic choices, morphological transformation of terms, structure of sentences and semantic choices.
In this paper, we concentrate on the grammatical analysis of sentences to identify the grammatical category\(^3\) of words that has the most important semantic content according to the congruences between semantic and grammatical categories identified by Halliday and Webster [64]. Halliday and Webster showed how in scientific text there is a cross-coupling between the grammar and the semantics: each grammatical category corresponds to a specific semantic phenomenon (see Table II). In human experience the most recurring type of phenomenon is a thing, in its scientific form as an entity, and the category which construes such entities is the noun.

In the software development process, the whole information content of a software artifact orbits around the entities that are grammatically represented by nouns or nominal groups (as shown in Table II). Indeed, where the everyday mother tongue of commonsense knowledge construes reality as a balanced tension between things and processes, the elaborated paradigm of scientific knowledge reconstructs it as a hierarchical set of things. In this prospective, the sentences typically have only one clause\(^4\) consisting of just one or two nominal groups\(^5\), propped up by a verbal group\(^6\), usually a relational process and most typically the verb be. The nominal groups, may be enormously long and complex, since all the lexical material is compressed into these one or two groups. For a better comprehension, Table III shows an example of congruence between semantic and grammatical categories in a sentence extracted from a high level requirement of Pine, one of the software repositories used in our empirical evaluation.

As a consequence, we can note a quantitative and qualitative increase in the importance of nouns counterbalanced by a decrease in importance of the other grammatical categories. Another supporting point to such a conjecture is represented by the linguistic phenomenon of nominalization [63]. It consists of the use of a verb, an adjective, or an adverb as the head of a noun phrase, with or without morphological transformation. With nominalization, a process, such as “connect”, is

---

\(^3\)The term grammatical category has been used to denote the parts of speech, such as nouns, verbs, adverbs, and pronouns.

\(^4\)In grammar, a clause is the smallest grammatical unit that can express a complete proposition: it consists of a subject, a verb and an object.

\(^5\)A nominal group typically comprises a noun surrounded by other items (words) that all in some way characterize that noun.

\(^6\)A word or group of words that functions as a verb.
Table III. Example of congruence between semantic and grammatical categories.

<table>
<thead>
<tr>
<th>Elements of sentence</th>
<th>Grammatical elements</th>
<th>Semantic elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>The system</td>
<td>Nominal group</td>
<td>Entity</td>
</tr>
<tr>
<td>shall have</td>
<td>Verbal group</td>
<td>Process</td>
</tr>
<tr>
<td>address book</td>
<td>Nominal group</td>
<td>Entity</td>
</tr>
<tr>
<td>available</td>
<td>Adjective</td>
<td>Quality</td>
</tr>
<tr>
<td>to store contacts</td>
<td>Nominal group</td>
<td>Entity</td>
</tr>
</tbody>
</table>

observed, generalized, and then theorized about, so that it becomes a virtual entity “connection”. In this way, a process, congruently construed as a verb, is reconstructed metaphorically as a noun.

While there is some minor variation among different languages in the way this is typically done, the grammar of technical-scientific argumentation is largely in common. Indeed, even if the Indo-European languages have different types of inflectional morphology for the different categories of morphemes, often this does not represent an impediment to nominalization: the root or stem of the adjective can be readily deprived of its adjectival inflections and adorned with nominal inflections. For example, the Italian language has a number of nominalization suffixes, and some of these suffixes have been borrowed from English, either directly or through other romance languages. Other examples can be seen in German, Spanish and Portuguese, as well as in Chinese and Japanese.

How noticed by Halliday and Webster [64], the nominalization is more than just turning a verb into a noun: it allows to condense a verb into a virtual entity that has more semantic density. In other words, the nominalized verbs are more important than other verbs because the nominalized verbs denote the abstraction of processes considered important in a technical argumentation, while other verbs denote secondary processes.

For these reasons (i.e., centrality of the entities in the software artifacts and nominalization of the most important verbs) we propose a linguistic approach that considers the conceptual peculiarity of the technical language of software documents and adapt the indexing process aiming at improving the recovery accuracy of IR-based traceability recovery methods.
3.2. Definition of the Proposed Linguistic Approach

In order to take into account the peculiarities of the technical language used in software artifacts, we propose the use of a modified artifact indexing process that indexes the terms of the vocabulary according to their grammatical category (see Figure 2). For this purpose, we use a POS-tagger\(^7\) that tags all the terms extracted from an artifact by specifying its grammatical nature (e.g., verb, noun, adjective). Part-of-speech tagging is the process of adorning or tagging words in a text with each word’s corresponding part of speech. The process is based both on the meaning of the word and its positional relationship with adjacent words. A simple list of the parts of speech for English includes adjective, adverb, conjunction, noun, preposition, pronoun, and verb.

The POS-tagger can be viewed as a sub-module of the indexing process that takes a document as input and returns a tagged document as output. Once obtained the tagged artifact, the indexer builds the term-by-document matrix considering only the terms that are members of the class noun. In this way, the term filtering is not only based on a stop word list and/or a stop word function, but it is based on the grammatical nature of the extracted term. From this point of view, the proposed approach represents a grammatical-semantic space reduction method, since it indexes

---

\(^7\)In our study, we used a POS tagger called Tree-Tagger that is available for downloading at http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/. It is able to process several languages, including German, English, French, and Italian. For each language, a specific vocabulary and configuration file have to be used.

Figure 2. The modified indexing process proposed in this paper.
only nouns and nominalized verbs (grammatical) that in a technical language represent the main source of information (semantic), and removes all the other terms (e.g., verbs) that play only a marginal role (space reduction). Formally, let \( D = \{d_1, \ldots, d_n\} \) be the set of software artifacts and \( V_A = \{\text{term}_i : \text{tag}(\text{term}_i) = \text{noun}, \ i = 1 \ldots m\} \) be the set of nouns extracted from \( D \). At the end of the indexing process we obtain the following term-by-document matrix:

\[
M = \begin{pmatrix}
\text{term}_1 & w_{1,1} & w_{1,2} & \ldots & w_{1,n} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\text{term}_m & w_{m,1} & w_{m,2} & \ldots & w_{m,n}
\end{pmatrix}
\]

where the generic entry \( w_{i,j} \) denotes a measure of weight (i.e. relevance) of \( i^{th} \) noun in \( j^{th} \) document.

It is worth noting that using a POS tagger it is possible to evaluate different indexing strategies giving different weights to the different parts of the speech. In our case we give weight 1 to nouns and 0 to the other morphemes.

4. EMPIRICAL EVALUATION

This section describes the design and the results of the empirical study we conducted to evaluate the proposed indexing approach in the context of IR-based traceability recovery. The study was conducted following the Goal-Question-Metric paradigm by Basili et al. [65]. Raw data and working data sets are available for replication purposes [66].

4.1. Planning

The context of the study consists of five software repositories, namely EasyClinic, eTour, MODIS, CM1 and Pine. Table IV summarizes the main characteristics of the considered repositories. For EasyClinic, source code artifacts and different types of high-level artifacts (i.e., use cases, UML interaction diagrams, and test cases) are available, while the eTour repository only contains source code artifacts and use cases. The language of the software artifacts of these two projects is Italian.
Table IV. Characteristics of the software repositories used in the case study.

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>Language</th>
<th>Artifacts Type</th>
<th>Number</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>EasyClinic</td>
<td>A software system used to manage a doctor’s office developed by students</td>
<td>Italian</td>
<td>Use Cases</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>UML interaction diagrams</td>
<td>20</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Test Cases</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code Classes</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>eTour</td>
<td>An electronic touristic guide developed by students</td>
<td>Italian</td>
<td>Use Cases</td>
<td>58</td>
<td>174</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code Classes</td>
<td>116</td>
<td></td>
</tr>
<tr>
<td>MODIS</td>
<td>A software system developed by NASA</td>
<td>English</td>
<td>High Level Requirements</td>
<td>19</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Low Level Requirements</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>CM1</td>
<td>A science instrument developed by NASA</td>
<td>English</td>
<td>High Level Requirements</td>
<td>22</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Low Level Requirements</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Pine</td>
<td>A text-based client for emails management</td>
<td>English</td>
<td>High Level Requirements</td>
<td>49</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Use Cases</td>
<td>51</td>
<td></td>
</tr>
</tbody>
</table>

For the two NASA’s repositories (MODIS and CM1) low and high levels requirements written in English are available. Finally, the Pine repository contains two kinds of high-level artifacts written in English, namely requirements and use cases. For all the artifact repositories the traceability matrix provided by the original developers was used to evaluate the retrieval accuracy of the experimented IR-based traceability recovery methods.

The study aims at addressing the following research question:

*Does the proposed indexing process improve the accuracy of IR-based traceability recovery methods?*

To answer this research question we recovered traceability links between the artifacts of the object systems using two different artifact indexing processes:

- *All*: all the terms (e.g. nouns, adjectives, and verbs) contained in the artifact are equally considered during the indexing process. Only the terms filtered by stop-word list and function are removed;
- **Noun**: only the nouns contained in the artifact contents are considered during the indexing process. All the other terms (e.g., adjectives, verbs) are discarded, in addition to the terms pruned out by stop-word lists and functions.

In both strategies we considered each software artifact as a text document. For source code we considered both identifiers and comments. Each file is pre-processed in order to filter non-textual tokens (i.e., operators, special symbols, some numerals, etc.). In addition, identifiers composed of two or more words are separated into their constituent words using a tool that relies on coding conventions (e.g., `getFirstName` is decomposed into `get`, `first`, and `name`). We also applied both stop word functions (with threshold equal to three) and stop word lists for English and Italian languages. The used stop word list included, other than Italian or English standard stop word lists, (i) programming language (C/Java) keywords, (ii) recurring words in document templates (e.g., use case, requirement, or test case template) and (iii) author names. Finally, stemming is applied on the extracted terms.

In order to improve the generalizability of our results, we considered (i) different IR methods, namely VSM, LSI, and JS, and (ii) different types of artifacts (use cases, requirements, interaction diagrams, source code, and test cases), written in Italian or English, extracted from five different projects. We recovered links between documentation and source code of EasyClinic and eTour, between high and low level requirements of MODIS and CM1, and between high-level requirements and use cases of Pine. Table V reports the seven traceability recovery activities performed on the five software repositories, as well as the total number of correct links and the number of possible links for each activity. Note that on EasyClinic three different recovery activities were performed to recover links between source code and (i) use cases, (ii) interaction diagrams, (iii) and test cases, respectively.

To evaluate the accuracy of the experimented IR methods, we developed a tool that automatically classifies the proposed links and identifies the number of correct links and false positives. The classification made by such a tool is based on the original traceability matrix. The classification process starts from the top of the ranked list and stops when all correct links are recovered.
Table V. Number of links between source and target artifacts considered in the study.

<table>
<thead>
<tr>
<th>System</th>
<th>Source</th>
<th>Target</th>
<th>Correct Links</th>
<th>Possible links</th>
</tr>
</thead>
<tbody>
<tr>
<td>EasyClinic</td>
<td>Interaction diagrams</td>
<td>Source code</td>
<td>69</td>
<td>940</td>
</tr>
<tr>
<td></td>
<td>Test cases</td>
<td>Source code</td>
<td>200</td>
<td>9,400</td>
</tr>
<tr>
<td>eTour</td>
<td>Use cases</td>
<td>Source code</td>
<td>366</td>
<td>9,628</td>
</tr>
<tr>
<td>MODIS</td>
<td>Low-level Requirements</td>
<td>High-level Requirements</td>
<td>26</td>
<td>931</td>
</tr>
<tr>
<td>CM1</td>
<td>Low-level Requirements</td>
<td>High-level Requirements</td>
<td>45</td>
<td>1,166</td>
</tr>
<tr>
<td>Pine</td>
<td>High-level Requirements</td>
<td>Use Cases</td>
<td>246</td>
<td>2,499</td>
</tr>
</tbody>
</table>

For MODIS, CM1, and Pine, the traceability matrices are provided by the original developers together with the software artifacts. The other two systems, EasyClinic and eTour, were developed by students at the University of Salerno. During the project, students were required to manually maintain traceability links between the developed artifacts. The traceability matrix was also validated during project review meetings conducted by the development team with PhD students and academic researchers. This validation process aimed at identifying artifacts incorrectly traced by students due to their limited experience on traceability management.

The accuracy of the IR method can be evaluated at each point of the ranked list using two well-known IR metrics, namely recall and precision [11]. Recall measures the percentage of links correctly retrieved, while precision measures the percentage of links retrieved that are correctly identified:

\[
recall = \frac{|correct \cap retrieved|}{|correct|} \times \%
\]

\[
precision = \frac{|correct \cap retrieved|}{|retrieved|} \times \%
\]

where \(correct\) and \(retrieved\) represent the set of correct links and the set of links retrieved by the tool, respectively.
Precision and recall can be useful to evaluate the accuracy of a traceability recovery method as well as to compare different methods. However, we also used a statistical significance test to verify that the number of false positives retrieved by one method is significantly lower than the number of false positives retrieved by another method. Thus, 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. In this way it is possible to compare the precision of two methods at the same level of recall, thus using a single dependent variable for the statistical analysis. Since the number of correct links is the same for each traceability recovery activity (i.e., the data was paired), we decided to use the Wilcoxon Rank Sum test [67] to test the following null hypothesis:

\[ H_0: \text{Considering only the nouns when indexing software artifacts does not significantly improve the performances of an IR-based traceability recovery method} \]

The results were intended as statistically significant at \( \alpha = 0.05 \).

Other than the result of the Wilcoxon Rank Sum test, it is of practical interest to estimate the magnitude of the difference between the accuracy achieved with indexing approaches. To this aim, we used the Cliff’s Delta \( d \), a non-parametric effect size [68] for ordinal data, which indicates the magnitude of the effect of the main treatment on the dependent variables (“whereas p-values reveal whether a finding is statistically significant, effect size indicates practical significance” [68]). Cliff’s Delta ranges from \(-1\) to \(1\). The effect size is considered small for \( d < 0.33 \), medium for \( 0.33 \leq d < 0.474 \) and large for \( d \geq 0.474 \) [69]. We chose the Cliff’s Delta \( d \) effect size as it is appropriate for our variables (in ratio scale) and it is quite easy to be interpreted, given the different levels (small, medium, large) defined for it.

4.2. Analysis of the Results

This section reports on the results achieved when performing the traceability recovery activities described in Section 4.1. Table VI shows the results of the Wilcoxon test and the Cliff’s effect size used to evaluate our null hypothesis \( H_0 \) for each tracing activity. The results of the tests indicate that generally the retrieval accuracy improves considering only the nouns during the indexing process.
Indeed, in 18 out of 21 cases the number of false positives retrieved by an IR method with the use of the proposed approach is significantly lower than the number of false positives retrieved by the same method when indexing all the terms of the vocabulary (p-value < 0.05).

In addition, the Cliff’s Delta (see Table VI) generally reveals a large or medium effect size of the improvement achieved by indexing only nouns, where the improvement is intended as a reduction of false positives. The magnitude of the improvement is graphically represented by the diagram in Figure 3-a which shows the trend of false positives when tracing code classes onto test cases of EasyClinic using LSI. In this case, indexing only the nouns results in halving the number of false positives for recall values less than 90% (181 out of 200 correct links retrieved).

The role of the nouns in IR-based traceability recovery processes is emphasized by the relation diagrams in Figure 3-b. The diagram shows the precision each correct link is recovered with. It is worth noting how the linguistic approach tends to increase the rank of correct links facilitating their identification. Indeed, when indexing only the nouns contained in the artifact corpus, the correct links are recovered with a higher precision, if compared to the precision obtained indexing all the terms.

Figure 4 shows the effect of our linguistic approach on a use case and a code class from the EasyClinic repository. The Figure shows a pair of linked artifacts (i.e., correct link) where the nouns (in bold face) represent the bulk of the vocabulary overlap between these two artifacts.
case contains a high number of useless words (e.g., verbs, adjectives) that do not contribute to the textual similarity and only some of them are pruned out by stop word removal (underlined). For this reason, indexing only the nouns allows to further remove “noisy” words resulting in a considerable increment of the textual similarity between the two artifacts.

In all cases where the proposed approach is statistically better than the traditional indexing process, we can observe a drastic reduction of false positives, especially for increasing recall values (see our technical report for more details [66]). Such a result indicates a notable improvement for the end user. Indeed, in several cases we achieved a considerable reduction of false positives especially when the goal is to recover all correct links. For example, when tracing code classes onto test cases of EasyClinic our approach allows to save more than 500 false positives.

When tracing code classes onto UML interaction diagrams of EasyClinic, for all the experimented IR methods our approach achieved a small improvement if compared with the results achieved when tracing code classes onto use cases and test cases. This is because the UML interaction diagrams are used to explain how objects interact and such interactions are generally described using sentences.
This functionality allows the Operator to modify the data of a chemical laboratory.

INITIALISING: Operator.

ACTOR: He/she wants to add a new laboratory in the Hospital Informative System (H.I.S.)

PRE-CONDITIONS:
The Operator has been admitted by the System and he/she has the data of the new laboratory.

GUARANTEES
Failure: the data in the H.I.S. is not modified.
Success: the new laboratory is correctly stored in the H.I.S.

START
The System shows the list of all the laboratory and the Operator activates the functionality to add a new laboratory.

Main Scenario:
1. The system shows a form to insert the data of a new laboratory.
2. The operator fills in the form.
3. The operator submits the data.
4. The system verifies the data inserted by the Operator.
5. The system stores the data.
6. The system notifies that the operation was correctly completed.

Alternative scenario: Data not valid
4.1 The system shows a message highlighting the incorrect data
4.2 The system recovers the execution from point 1 setting the form fields with the data inserted by Operator.

Error scenario: The Operator aborts the functionality
3.1 Stop the execution of the use case with no success.

(a) Use case

(b) Code class

Figure 4. Example of two linked artifacts extracted from EasyClinic.

in active form, where a limited set of verbs is used to denote message sending. For this reason, the noise introduced by verbs in these diagrams is limited with respect to the noise introduced in use cases and test cases (scenarios) and then the difference between indexing all terms and only nouns is lower.

While our approach generally achieved a statistically significant improvement, there are two important exceptions to this general rule. There is no significant improvement of the performances when tracing high-level requirements onto low level requirements of MODIS (for VSM and LSI) and of CM1 (for JS only). To identify the factors that cause these exceptions a fine-grained analysis is required.

Table VII reports the differences between the precision values as well as the differences of the number of false positives achieved with the two artifact indexing processes (i.e., Noun and All) at different levels of recall for CM1 and MODIS. Even if the Wilcoxon tests did not reveal any
Table VII. Improvement of precision and reduction of number of false positives at different levels of recall.

Cases where there is an improvement of precision are reported in bold face.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Traced artifacts</th>
<th>IR Method</th>
<th>Prec(20%)</th>
<th>Prec(40%)</th>
<th>Prec(60%)</th>
<th>Prec(80%)</th>
<th>Prec(100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MODIS HLR→LLR</td>
<td>VSM</td>
<td>-16.67%</td>
<td>+1</td>
<td>-1.76%</td>
<td>+8</td>
<td>-2.01%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSI</td>
<td>-15.05%</td>
<td>+1</td>
<td>-1.76%</td>
<td>+28</td>
<td>+0.50%</td>
</tr>
<tr>
<td></td>
<td>CM1 HLR→LLR</td>
<td>VSM</td>
<td>-10.32%</td>
<td>-4</td>
<td>-5.05%</td>
<td>+22</td>
<td>-3.29%</td>
</tr>
<tr>
<td></td>
<td></td>
<td>LSI</td>
<td>+2.03%</td>
<td>-10</td>
<td>+4.42%</td>
<td>-10</td>
<td>+5.16%</td>
</tr>
</tbody>
</table>

statistical improvement for these recovery activities, the results showed in Table VII highlight that it is still possible to obtain a false positive reduction when the recall is 100% (although not statistically significant). On the other hand, for recall levels lower than 100% our approach does not always result in an improvement.

Figure 5 shows the precision/recall curves achieved by different IR methods when indexing all terms and when indexing only the nouns on MODIS. The analysis of such curves highlights that the proposed indexing process does not result in a clear improvement for any of the experimented IR methods. Even in the case of JS where a statistically significant improvement of the performances is achieved when indexing only the nouns, such an improvement is not clearly evident on the precision/recall curve. This result is probably due to the poor verbosity of the MODIS artifacts.

Figure 6 shows a high level requirement traced onto a low level requirement in MODIS. It is worth noting that in this case the number of terms appearing in both the artifacts is very low. Thus, there is no substantial difference between indexing all terms or indexing only the nouns (in bold face). Moreover, the MODIS artifacts contain several abbreviated names and acronyms that the POS-tagging software is not able to classify from a grammatical point of view. Probably, in this case techniques based on query expansion [49] and acronyms expansion could have helped to improve the performances.

For CM1, we obtained a statistically significant improvement only using vector space based models (LSI and VSM) while worst performances are achieved using our indexing strategy with
Each software process shall trap and properly process all exceptions that may produce an abnormal termination and report all such events using the SDPTK error message functions.

Shall write $L_0$ _open_ log _msgs_ to Log _messages_ when errors occur in the Open _Level_0 _file_ process.

(a) High level requirement  (b) Low level requirement

Figure 6. Example of two linked artifacts extracted from MODIS.

JS. Figure 7 shows the precision-recall curves achieved for the different IR methods on this dataset.

As we can see, for VSM and LSI the precision/recall curve obtained when indexing only the nouns is almost always above the curve obtained when indexing all terms of the vocabulary, while for JS the scenario is the opposite.
To identify the factors that affect the results achieved with JS, a qualitative analysis of the CM1 artifacts is needed. Figure 8 shows a high level requirement traced onto a low level requirement of CM1. Unlike MODIS, in CM1 only the high level requirements have a low verbosity while the low-level requirements are highly verbose. Thus, the empirical distribution of the nouns (in bold face) occurring in the high level requirements (which are about 38% of terms for the high level requirement in Figure 8) is very different with respect to the distribution of nouns occurring in the low level requirements (in the low level requirement of Figure 8 the nouns represents about 66% of terms contained in the artifact corpus). Such a difference negatively affects the probabilistic model (JS) that computes the textual similarity between different types of artifacts on the basis of the empirical distribution of terms (only nouns for the proposed indexing process).
The DPU-CCM shall implement a mechanism whereby large memory loads and dumps can be accomplished incrementally.

Control and Monitoring the CCM Control Task also handles memory dump commands. In the event of a data dump command, the CCM Control Task will break the dump into manageable pieces and dump a small portion at a time, each time the task is awakened. The purpose of this “deferred” activity is to prevent a large dump from consuming available CPU time by keeping the high-priority CCM Command Dispatch Task busy for an extended period.

(a) High level requirement SRS5.12.2.1
(b) Low level requirement DPUSDS5.12.1.2.4

Figure 8. Example of two linked artifacts extracted from Modis.

4.3. Threats to Validity

This section discusses the threats that might affect the validity of our results. A relevant threat is related to the repositories used in the empirical study. The chosen repositories have the advantage of containing various types of artifacts (use cases, requirements, design documents, source code, test cases). Also, in our experiments we have considered students’ (EasyClinic and e-Tour), as well as industrial (Modis and CM1) and open source (Pine) projects. They are the largest repositories available to experiment IR methods for traceability recovery. In addition, EasyClinic was previously used by other authors to evaluate IR methods [33], and the same happened for MODIS and CM1 [16, 36]. Nevertheless, replication on artifacts taken from larger industrial projects—as well as from projects coming from specific domains where the technical language could possibly affect the filtering performance—are highly desirable to further generalize our results.

Another threat is represented by the metrics used to evaluate the accuracy of the experimented IR-based traceability recovery methods. Recall and precision are widely used metrics for assessing an IR technique. Moreover, the number of false positives retrieved by a traceability recovery tool for each correct link retrieved well reflects its retrieval accuracy.

The quality of the POS tagger used in our experimentation could also affect the results achieved. We used a freely available tool with a high level of accuracy. As reported in its documentation, TreeTagger has an accuracy of about 95% for the English language, while about 70% is the accuracy for Italian. It is worth noting that we also applied POS tagger on source code. The natural language found in code artifacts differs from that found in standard prose [70]. This difference might...
Table VIII. The vocabulary size in term-by-document matrices.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Number of terms</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Noun</td>
<td>All</td>
</tr>
<tr>
<td>EasyClinic</td>
<td>401</td>
<td>895</td>
</tr>
<tr>
<td>eTour</td>
<td>591</td>
<td>1,405</td>
</tr>
<tr>
<td>Modis</td>
<td>205</td>
<td>269</td>
</tr>
<tr>
<td>CM1</td>
<td>381</td>
<td>619</td>
</tr>
<tr>
<td>Pine</td>
<td>202</td>
<td>353</td>
</tr>
</tbody>
</table>

potentially limit the effectiveness of off-the-shelf tools. However, in our study we did not observe a decrement of the accuracy of our POS tagger tool when it was applied on source code artifacts. This is probably due to the high meaningfulness and quality of the identifiers in the source code of our object systems. Nevertheless, in case the performances of the POS tagger are not adequate on source code identifiers, the tool can be appropriately customized in order to improve its accuracy [70].

The accuracy of the oracle could also affect the achieved results. To mitigate such a threat we used the traceability matrices provided by the original developers. Moreover, the links contained in the traceability matrices of the students’ projects were validated during project review meetings made by the development teams together with PhD students and academic researchers.

In the proposed indexing strategy, nouns are the only terms considered. We also tried to give a low weight (instead of zero) to other terms. However, the effects of such a variant of the approach are variable, sometimes resulting in small improvements, sometimes in small decrements in accuracy. Thus, we preferred to completely remove all the terms that are not nouns in order to have also a sensible reduction of the size of the term-by-document matrix. Table VIII shows the number of terms in the term-by-document matrix obtained by using the two different artifact indexing approaches (i.e., Noun and All). The drastic reduction of the number of terms (often greater than 30%) represents another important result, since it reduces the computation time (i) of the document similarity (for JS, LSI and VSM methods) and (ii) of the singular value decomposition (for LSI).
Finally, attention was paid to not violate assumptions made by statistical tests. Whenever conditions necessary to use parametric statistics did not hold (e.g., analysis of each experiment data), we used non-parametric tests, in particular Wilcoxon test for paired analysis. In addition, we used the Wilcoxon test to compare the cumulative number of false positives for each correct links. In alternative, 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.

5. CONCLUSION AND FUTURE WORK

One of the main challenges in traceability management is to improve the performances of IR-based traceability recovery by reducing the number of false positives retrieved. A common way to reach such a goal is to reduce the “noise” various types of artifacts carry on. For example, natural language contains common words that have to be removed (stop words) [11] or could contain critical terms and phrases that can be regarded as more meaningful in identifying traceability links [16, 17, 18, 19].

In this paper, we described how to improve the accuracy of IR-based traceability recovery methods by using a linguistic-based heuristic to index the artifacts. Such a heuristic is used to identify the extracted terms that have to be included in the artifact corpus and considered to define the textual similarity between two artifacts. We proposed to index only the nouns extracted from the artifact content, since in a technical language (the language of the software documents) the terms that provide more indication on the semantics of a document are the nouns [20, 21].

The results of an empirical evaluation confirm our conjecture. Indexing only the nouns generally improves the accuracy of all the experimented IR methods, which are VSM, LSI and JS. Thus, the proposed indexing process is largely independent of the particular IR method and can be used with
both probabilistic and vector space models. In addition it reduces the size of the term-by-document matrix and then the computation time.

The empirical evaluation highlighted that the improvement provided by the proposed indexing strategy is neutralized when the verbosity of the artifacts is very poor. In particular, on CM1 and MODIS, where artifacts are composed of one or two sentences, there is no substantial difference between indexing all terms and indexing only the nouns. This suggests that our approach might be applied after applying techniques based on query expansion [49] and acronyms expansion [24, 25, 26, 27, 28] that can be used to enrich the verbosity of the artifacts to be traced.

Replication in different contexts and with different objects is the only way to corroborate our findings. Replicating the experiment using different repositories, IR methods, and traceability recovery activities is part of the agenda of our future work. We also plan to experiment the proposed artifact indexing approach combined with other enhancing strategies, such as user feedback analysis [11], smoothing filters [47] and query expansion [49, 50].

6. ACKNOWLEDGMENTS

We would like to thank Jane Huffman Hayes, Wei-Keat Kong, Wenbin Li, Hakim Sultanov, and Alex Wilson for providing us the Pine dataset. We would like to thank the anonymous reviewers for their detailed, constructive, and thoughtful comments that helped us to improve the presentation of the results in this paper.

REFERENCES


Prepared using smrauth.cls DOI: 10.1002/smr


