Applying a Smoothing Filter to Improve IR-based Traceability Recovery Processes: An Empirical Investigation

Andrea De Lucia\textsuperscript{a}, Massimiliano Di Penta\textsuperscript{b}, Rocco Oliveto\textsuperscript{c}, Annibale Panichella\textsuperscript{a}, Sebastiano Panichella\textsuperscript{b}

\textsuperscript{a}University of Salerno, Via Ponte don Melillo - 84084 Fisciano (SA), Italy
\textsuperscript{b}University of Sannio, Viale Traiano - 82100 Benevento, Italy
\textsuperscript{c}University of Molise, C.da Fonte Lappone - 86090 Pesche (IS), Italy

Abstract

Context: Traceability relations among software artifacts often tend to be missing, outdated, or lost. For this reason, various traceability recovery approaches—based on Information Retrieval (IR) techniques—have been proposed. The performances of such approaches is often influenced by “noise” contained in software artifacts (e.g., recurring words in document templates or other words that do not contribute to the retrieval itself).

Aim: As a complement and alternative to stop word removal approaches, this paper proposes the use of a smoothing filter to remove “noise” from the textual corpus of artifacts to be traced.

Method: We evaluate the effect of a smoothing filter in traceability recovery tasks involving different kinds of artifacts from five software projects, and applying three different IR methods, namely Vector Space Models, Latent Semantic Indexing, and Jensen-Shannon similarity model.

Results: Our study indicates that, with the exception of some specific kinds

Email addresses: adelucia@unisa.it (Andrea De Lucia), dipenta@unisannio.it (Massimiliano Di Penta), rocco.oliveto@unimol.it (Rocco Oliveto), apanichella@unisa.it (Annibale Panichella), spanichella@unisannio.it (Sebastiano Panichella)

This paper is an extension of the work “Improving IR-based Traceability Recovery Using Smoothing Filters” appeared in the Proceedings of the 19th IEEE International Conference on Program Comprehension, Kingston, ON, Canada, pp. 21-30, 2011. IEEE Press.

Preprint submitted to Information and Software Technology September 17, 2012
of artifacts (i.e., tracing test cases to source code) the proposed approach is able to significantly improve the performances of traceability recovery, and to remove “noise” that simple stop word filters cannot remove.

**Conclusions:** The obtained results not only help to develop traceability recovery approaches able to work in presence of noisy artifacts, but also suggest that smoothing filters can be used to improve performances of other software engineering approaches based on textual analysis.

**Keywords:** Software Traceability, Information Retrieval, Smoothing Filters, Empirical Software Engineering.

---

1. **Introduction**

   In recent and past years, textual analysis has been successfully applied to several kinds of software engineering tasks, for example impact analysis [1], clone detection [2], feature location [3, 4], refactoring [5], definition of new cohesion and coupling metrics [6, 7], software quality assessment [8, 9, 10, 11], and, last but not least, traceability recovery [12, 13, 14].

   Such a kind of analysis demonstrated to be effective and useful for various reasons:

   • it is lightweight and to some extent independent on the programming language, as it does not require a full source code parsing, but only its tokenization and (for some applications) lexical analysis;

   • it provides information (e.g., brought in comments and identifiers) complementary to what structural or dynamic analysis can provide [6, 7];

   • it models software artifacts as textual documents, thus can be applied to different kinds of artifacts (i.e., it is not limited to the source code) and, above all, can be used to perform combined analysis of different kinds of artifacts (e.g., requirements and source code), as in the case of traceability recovery.

   Textual analysis has also some weaknesses, and poses challenges for researchers. It strongly depends on the quality of the lexicon: a bad lexicon often means inaccurate—if not completely wrong—results. There are two common problems in the textual analysis of software artifacts. The first is represented by the presence of inconsistent terms in related documents (e.g.,
requirements express some concepts using certain words, whereas the source code uses synonyms or abbreviations). In general, there could be different forms of the same word (verb conjugations, plurals), multiple words having the same meaning (synonymy) or cases where the same word has multiple meanings in different contexts (polysemy).

The second problem is related to the presence of “noise”2 in software artefacts, for example due to recurring terms that do not bring information relevant for the analysis task. Examples of such terms include programming language keywords and terms that are part of a specific document template, such as a test case specification, a use case, or a bug report (for example, in a use case, terms like use case, actor, flow of events, entry/exit condition).

In this paper we focus on one specific software engineering problem being solved using textual analysis (i.e., traceability recovery). Different authors have proposed the use of techniques such as Vector Space Model (VSM) [15], Latent Semantic Indexing (LSI) [16], or Jensen & Shannon (JS) similarity model [17] to recover traceability links between different kinds of software artifacts (see, e.g., [12, 13, 17]). All these methods are based on the assumption of a consistent lexicon between different artifacts to be traced (e.g., between requirements and source code). During recent and past years, several IR-based traceability recovery tools have also been proposed to support the software engineer during the traceability recovery process [8, 18, 19, 20, 21].

As any text-based approach, IR-based traceability recovery suffers of the problems mentioned above. Some problems, such as synonymy and polysemy, are partially solved using stemmers [22] or clustering and space reduction based IR-methods, such as LSI [16], while to deal with “noise” stop word removals or indexing mechanisms such as tf-idf [15] are often used. However, especially when source and target artifacts being traced contain recurring terms (e.g., when use cases and requirements follow a given template) such a “noise” remains, unless one manually builds a customized stop word filter for each kind of artifact.

In other fields, such as image processing, noise is also pruned out through smoothing filters [23]. Smoothing filters are approximating functions aimed at capturing relevant patterns in the data while pruning out other components (e.g., noise or certain frequencies). For instance, images are often

---

2In this context we mean by “noise” terms that do not allow to discriminate documents in our Information Retrieval process, rather than noise as formally defined in signal theory.
filtered by pruning out high frequencies (e.g., rapid changes of colors).

In this paper we propose the use of a smoothing filter to improve the performances of existing traceability recovery techniques, based on VSM, LSI, and JS. The proposed filter is inspired from a Gaussian filter used in image processing [23], and removes the “common” information among artifacts of the same kind (e.g., between use cases, or between source code artifacts) that does not help to characterize the artifact semantics.

We studied the use of the smoothing filter when recovering traceability links among different kinds of artifacts—use cases/requirements, UML diagrams, source code, and test cases—of five software projects belonging to different domains and developed with different programming languages. The obtained results indicate that the usage of a smoothing filter significantly improves the traceability recovery performances of the studied IR methods. Also, results indicate that a smoothing filter is more than a simple replacement of stop word filters, because (i) when used in combination it helps to remove additional “noise” with respect to the use of stop word filters alone, and (ii) when used alone it is even more effective than general stop word filters. Its performance is excelled only by stop word filters customized with respect to the kind of software artifacts, which in general require a substantial effort to be produced.

The paper is organized as follows. Section 2 provides background notions on IR-based traceability recovery and discusses related work. Section 3 introduces the notion of smoothing filters and explains how a smoothing filter can be applied to traceability recovery. Section 4 describes the empirical study we performed to evaluate the benefits of the proposed filter. Results are reported and discussed in Section 5, while Section 6 discusses the threats to validity. Finally, Section 7 concludes the paper.

2. IR-based Traceability Recovery: Background and Related Work

IR-based traceability recovery aims at identifying candidate traceability links between different artifacts by relying on the artifacts’ textual content—that is exactly how IR techniques aim at finding documents relevant to a given query. Traceability recovery works by applying an IR technique 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 candidates to be linked (candidate links). The software engineer cuts the ranked list by defining a threshold on the similarity value of the candidate links or a fixed number of candidate links and analyzes each candidate link in the top-most part of the ranked list. Based on this analysis, the software engineer can trace the link (i.e., classify it as true positive), or classify it as a false positive.

2.1. Instantiating an IR process for Traceability Recovery

In general, an IR-based traceability recovery process follows the process described in Figure 1. The artifacts are first indexed using the following steps. The first step consists of identifying terms contained in software artifacts, by means of (i) term extraction, aimed at extracting words from the artifacts and removing anything useless (e.g., punctuation or programming language operators); (ii) identifier splitting, aimed at splitting composite identifiers (e.g., using the camel case splitting heuristic); and (iii) term filtering, aimed at removing common terms, referred to as “stop words” (e.g., articles, prepositions, common use verbs, or programming language keywords). Words shorter than a given length (e.g., shorter than three characters [15]) are removed as well. In addition, morphological analysis of the extracted words is
often performed to bring back words to the same root (e.g., by removing plurals to nouns, or verb conjugations). The simplest way to do morphological analysis is by using a stemmer, e.g., the Porter stemmer [22].

The extracted information is stored in a \( m \times n \) matrix (called term-by-document matrix), where \( m \) is the number of terms occurring in all 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 [15]. A widely used measure is the \( tf-idf \) (term frequency-inverse document frequency), 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 discriminating power (high \( idf \)).

Different IR methods can be used to compute the textual similarity between two artifacts, such as VSM [12], LSI [13], JS [17], Latent Dirichlet Allocation (LDA) [24], B-Spline [25], and Relational Topic Models (RTM) [26]. The experiments conducted to evaluate the accuracy of all these IR methods indicated that there is no clear technique able to sensibly outperform the others. In recent studies [26, 27] it has been empirically proven that VSM and LSI are nearly equivalent, while topic modeling techniques (LDA and RTM) are able to capture some important information missed by the other experimented IR methods.

In this work, we compare the retrieval accuracy of three IR methods, namely VSM, LSI and JS. We chose these three techniques because (i) they are the most widely adopted (especially VSM and to some extent LSI), and (ii) they exhibit the highest performances [26, 27]. VSM is the simplest IR-based technique applied to traceability recovery. 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 [15]. The similarity between two artifacts is measured as the cosine of the angle between the corresponding vectors. VSM does not take into account relations between terms of the artifacts vocabulary. For instance, having “automobile” in one artifact and “car” in another artifact does not contribute to the similarity measure between these two documents.

LSI [16] is an extension of the VSM. It was developed to overcome the synonymy and polysemy problems, which occur with the VSM model. LSI explicitly takes into account the dependencies between terms and between artifacts, in addition to the associations between terms and artifacts. For example, both “car” and “automobile” are likely to co-occur in different artifacts with related terms, such as “motor” and “wheel”. To exploit in-
formation about co-occurrences of terms, LSI applies Singular Value Decomposition (SVD) [28] 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.

JS [29] is driven by a probabilistic approach and hypothesis testing techniques. As well as other probabilistic models, it represents each document through a probability distribution. This means that an artifact is represented by a random variable where the probability of its states is given by the empirical distribution of the terms occurring in the artifact (i.e., columns of the term-by-document matrix). The empirical distribution of a term is based on the weight assigned to such a term for a specific artifact. In the JS method, the similarity between two artifacts is given by a “distance” of their probability distributions measured by using the JS Divergence [29].

2.2. Enhancing strategies for IR-based Traceability Recovery

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

Capobianco et al. [30] observed that the language used in software documents can be classified as technical language (i.e., jargon), where terms that provide more information about the semantics of a document are the nouns [31]. Thus, they proposed to index the artifacts taking into account only nouns contained in software artifacts. Their results indicated that the proposed approach improves the accuracy of both JS and LSI methods. Smoothing filters also helps to remove noise that can, in part, be due to technical language. However, they do not focus necessarily on certain parts-of-speech, but rather consider as noise terms occurring too often either in source or target documents.

An issue which 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 complementary set of terms, IR techniques will unlikely identify links between the sets of artifacts. Recently, a technique that attempts to alleviate such an issue has been introduced [32, 33]. The proposed approach uses the artifacts to be traced as queries for web search engines, and expands the terms in the query with the terms contained in the retrieved documents before indexing
the artifacts. Empirical studies indicated that using web mining to enhance queries improves retrieval accuracy. Smoothing filters are complementary to that, as also query expansion might introduce noise in a set of source or target artifacts.

During the indexing process a weighting schema is applied to define the importance of a term in an artifact. A first enhancement of the weighting schema can be achieved by considering the document length [34]. 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. Interesting results have been achieved using the pivot normalization term weighting approach, that allows to specify the normalization factor depending on the specific collection of artifacts [34]. Smoothing filters are also complementary to that, as they focus on the frequencies of terms in each set of artifacts, rather than on the document length.

The term weighting could also take into account (i) the structure of the artifacts [18]; and (ii) the importance of a term for a specific domain [35, 36, 37]. As for the latter, artifacts could contain critical terms and phrases that should be weighted more heavily than others, as they can be regarded as more meaningful in identifying traceability links. These terms can be extracted from the project glossary [35, 36] or external dictionaries [37]. Such approaches generally improve the accuracy of an IR-based traceability recovery tool. However, the identification of key phrases (as well as the use of external dictionaries) is much more expensive than the indexing of single keywords. In addition, a project glossary might not be available. The approach based on smoothing filters tries to alleviate such problems reducing the “noise” in software artifacts 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) [19, 38]. 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. Also in this case, smoothing filtering can be considered as a complementary technique. While feedbacks require human intervention, smoothing
filtering is completely automatic.

3. Enhancing Traceability Recovery with a Smoothing Filter

This section describes how to enhance an IR-based traceability recovery process using a smoothing filter. First, we discuss the reasons for using a filter, and then we define a smoothing filter suited for textual analysis in the context of traceability recovery.

3.1. The Textual Noise in Traceability Recovery

As explained in Section 2, an IR-based traceability recovery method recovers links between software artifacts based on their textual similarity. However, when computing the textual similarity, such methods generally do not explicitly consider the linguistic mismatch between (i) the application domain vocabulary, and (ii) the vocabulary used in different kinds of artifacts. In fact, the information contained in software artifacts is characterized by

1. words providing functional and descriptive information about the software itself (e.g., referring to its entities, entity properties, and behavior);
2. linguistic forms specific of the language used in the software artifacts. For example use cases can contain recurring terms related to the use case template, while the source code can contain programming language keywords or library function/method names. In general, we can assume that each kind of artifact (e.g., use cases, code, test cases) is expressed through a different set of terms belonging to a technical language [31]);
3. common words that have low information content.

Two artifacts are semantically related if they concern with the same or related pieces of functionality. Thus, only words providing functional and descriptive information about the software are useful to classify the correct links. Although—as explained in Section 2—stop word removal is part of a document indexing process, deciding what are the words to be pruned might be difficult for the following reasons:

- *template variability*: a development team often uses templates to produce software documentation. Usually, each kind of artifacts has a specific template, characterized by a set of words repeating throughout all artifacts of that category (e.g., all requirements or use cases). Moreover, different templates for the same type of artifacts might be used
within a project, especially when this involves different, distributed development teams. An *ad-hoc* stop word list can be used to remove the words of the template. However, this would require the analysis of patterns of each possible template;

- *developer subjectivity:* although a good software development team tends to have a consistent domain/application vocabulary when developing an application—as also enforced by some documentation standards [39]—each team member has a unique way of processing and writing a software document with respect to her own personal vocabulary. In this case, the use of a stop word list or a stop word function is ineffective: the subjective linguistic preferences of developers cannot be known *a priori.* Although tools have been developed to enforce such a consistency, previous studies found that some developers, especially the most skilled ones, tend not to follow guidelines/tool suggestions, and instead use their own naming conventions [40].

The above mentioned factors contribute to the noise that compromises the performances of IR methods. It is important to note that (i) developers use different templates for each kind of artifact and (ii) an artifact that belongs to a particular kind is characterized by linguistic choices directly related to its template. Thus, there is a strong interaction between the used language and the used template: when a developer writes an artifact (e.g., a use case, a test case, a source code artifact) she adopts a specific template and a specific technical language. For example, use cases contain terms such as *actor, flow, events, precondition, postcondition,* etc. In the practice, it might not be so trivial to identify all words that do not bring information to distinguish one document from another, and thus to help recovery traceability links.

When tracing different kinds of artifacts, it might not be obvious to build a stop word list for each particular kind of artifacts, since (i) different kinds of artifacts might require different sets of stop words, and (ii) it might not be clear what has to be included in the stop word list (e.g., programming language keywords, library function/method names). This, in general, requires a manual effort and does not always produce satisfactory results. This motivates the need for an automatic approach able to remove such a type of noise. In the following we describe how noise can be removed using a smoothing filter.
3.2. A Primer on Smoothing Filters

In statistics and image processing a smoothing filter (filter cleaning) is represented by an approximating function that attempts to capture important patterns in the data, while leaving out noise or other fine-scale structures/rapid phenomena (i.e., high frequencies) [23]. Clearly, different algorithms are applied as smoothing filters on the basis of the different kinds of digital data (e.g., images, sound). Essentially, the choice to use a filter with respect to another strictly depends on the kind of noise that has to be removed. In image processing, for example, a digital image is represented as a numerical matrix, where an entry $a_{i,j}$ denotes a pixel intensity value in the image. An image often contains noise introduced during its acquisition, and is characterized by pixels with very high or very low intensity values, with respect to the mean of the surrounding pixels. In other words, there are high frequencies in color transitions. Formally, a filter for a bi-dimensional datum (e.g., an image) is a transformation defined as:

$$g(x, y) = T[f(x, y)]$$

where $f$ is the input image, $g$ is the processed image and $T$ is the transformation operator. Similarly, a filter for a mono-dimensional datum (e.g., a sound filter) is defined as:

$$g(x) = T[f(x)]$$

Figure 2 shows a typical example of Gaussian filter applied to a noisy image [23].

3.3. Smoothing Filters for Traceability Recovery

When applying smoothing filters to software artifacts—represented as a term-by-document matrix—we need to account for some conceptual differences these kinds of artifacts have, with respect to signals and images. In

![Figure 2: Example of noisy image filtering through a Gaussian filter [23].](image)

(a) Image before filtering  
(b) Image after filtering
an image, every pixel often conceptually depends on its neighbors, unless
the pixel belongs to the edge of a shape. In a term-by-document matrix,
it is likely that artifacts do not depend on neighbor artifacts (i.e., the
ordering in which artifacts occur in the term-by-document matrix does not
matter). Thus, while an image filtering accounts for the pixel position, and
for the intensity variations between neighbor pixels, in software artifacts a
filter should not depend on the order in which we represent artifacts in the
matrix. Indeed, the information we have is invariant to swapping the posi-
tion of columns in the term-by-document matrix. For this reason, we propose
the use of a smoothing filter that considers the conceptual peculiarity of a
term-by-document matrix produced in the context of an IR-based traceability
recovery process. In the context of traceability recovery, the noise is gener-
ally represented by the “common” information (i.e., the average information,
contained in the artifacts of the same kind). To this aim, the proposed filter
computes for each kind of indexed artifact (e.g., use cases, code classes) the
average vector of all artifacts of that specific kind. From a signal process-
ing point-of-view such a vector can be resembled to the continuous compo-
nent of a signal. The continuous component explains the shared information
among artifacts, that does not help to characterize the artifact semantics.
Thus, the common information is pruned out by subtracting the continuous
component for that kind of artifacts. This results in a re-weighting of the
term-by-document matrix that modifies and redistributes the weights (i.e.,
relevance) of the terms according to their artifact kinds.

Let $M$ be the $m \times n$ term-by-document matrix:

$$
M = \begin{pmatrix}
  d'_1 & \ldots & d'_h & d''_h+1 & \ldots & d''_n \\
  w_{1,1} & \ldots & w_{1,h} & w_{1,h+1} & \ldots & w_{1,n} \\
  \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
  w_{m,1} & \ldots & w_{m,h} & w_{m,h+1} & \ldots & w_{m,n}
\end{pmatrix}
$$

where the generic entry $w_{i,j}$ represents the weight of the $i^{th}$ term in the
$j^{th}$ artifact. $D' = \{d'_1 \ldots d'_h\}$ and $D'' = \{d''_{h+1} \ldots d''_n\}$ are two sets of
vectors corresponding to two different kinds of software artifacts that need
to be traced (e.g., use cases and source code classes). In principle, the same
process can be applied to multiple sets of artifacts.

To apply the smoothing filter, we compute $D'$ and $D''$ as the mean vectors
of artifact vectors in $D'$ and $D''$, respectively:

$$
\overline{D'} = \begin{bmatrix}
\frac{1}{h} \sum_{i=1}^{h} w_{1,i} \\
\vdots \\
\frac{1}{h} \sum_{i=1}^{h} w_{m,i}
\end{bmatrix}
$$

$$
\overline{D''} = \begin{bmatrix}
\frac{1}{n-h} \sum_{i=h+1}^{n} w_{1,i} \\
\vdots \\
\frac{1}{n-h} \sum_{i=h+1}^{n} w_{m,i}
\end{bmatrix}
$$

Finally, the smoothed term-by-document matrix is obtained by subtracting $\overline{D'}$ from columns $1, \ldots, h$ and $\overline{D''}$ from columns $h + 1, \ldots, n$ of $M$, respectively.

Since the above subtraction can produce negative values, in the case of the JS Model these are set to zero, as in that case matrix elements represent probabilities, which cannot be negative. Also, values have been normalized in such a way that the sum of probabilities for each artifact is equal to one, also after applying the smoothing filter.

From a conceptual point of view, some documents can contain more noise than others. Because of that, the similarity between two documents after applying the filter can increase, decrease, or remain approximately the same, affecting the position of the document pair in the ranked list of candidate links. Specifically:

1. the similarity increases when the amount of noise was such to compromise the textual similarity between two artifacts;
2. the similarity decreases in case the filter penalizes terms that actually contributed to the similarity between the two documents;
3. the similarity remains unchanged either because there was relatively few noise to be removed, or because, after removing the noise, the proportion of terms contributing to the similarity (i.e., overlapping between documents) remained the same.

It is important to point out that the proposed approach does not require manual effort to analyze templates and linguistic choices used by developers. In addition, it is not particularly expensive from a computational point of view. More precisely, let $M$ be the $m \times n$ term-by-document matrix, the computational complexity of the proposed filter is $O(m \cdot n)$, because it requires to average $n$ vectors of size $m$. 

4. Empirical Study Definition and Planning

The goal of the study is to analyze whether the application of a smoothing filter improves the performances of IR-based traceability recovery methods. The perspective is of a researcher, that wants to assess the effects of a smoothing filter applied to IR methods. The context of the study consists of repositories of different software artifacts from five projects, namely:

- **EasyClinic**: a software system developed by Master students at the University of Salerno (Italy);
- **e-Tour**: an electronic touristic guide developed by Master students at the University of Salerno (Italy);
- **Modis**: the open source Moderate Resolution Imaging Spectroradiometer (MODIS) developed by NASA;
- **Pine**: a free and open source, text-based e-mail client developed at the University of Washington\(^3\);
- **i-Trust**: a medical application used as a class project for Software Engineering courses at the North Carolina State University\(^4\).

Table 1 reports the number of different kinds of artifacts contained in these repositories. Other than the listed artifacts, each repository also contains the traceability matrix built and validated by the application developers. For different kinds of artifacts, the traceability matrices were developed at different stages of the development (e.g., requirement-to-code matrices were produced during the coding phase). We consider such a matrix as the “oracle” to evaluate the accuracy of the different experimented traceability recovery methods.

The choice of these five projects is triggered by several factors. The most compelling factor is to have systems with different kinds of software development artifacts available (primarily requirements/use cases, then design documents, source code, and test cases), and with traceability matrices validated by developers. At the same time, we picked five projects differing for their domain: different domains use different jargons, therefore the performance of traceability recovery and of the filter may vary.

\(^3\)http://www.washington.edu/pine
\(^4\)http://agile.csc.ncsu.edu/iTrust/wiki/doku.php?id=tracing
Table 1: Characteristics of the software repositories used in the case study.

<table>
<thead>
<tr>
<th>System</th>
<th>Artifacts</th>
<th>Kind</th>
<th>Number</th>
<th>Total Number</th>
<th>Programming Language</th>
<th>Corpus Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>EasyClinic</td>
<td>Use Cases (UC)</td>
<td>20</td>
<td>160</td>
<td>Java</td>
<td>Italian</td>
<td></td>
</tr>
<tr>
<td></td>
<td>UML Interaction Diagrams (ID)</td>
<td>63</td>
<td>100</td>
<td>Italian</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test Cases (TC)</td>
<td>47</td>
<td></td>
<td>C</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Classes (CC)</td>
<td>47</td>
<td></td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>eTour</td>
<td>Use Cases (UC)</td>
<td>58</td>
<td>174</td>
<td>Java</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code Classes (CC)</td>
<td>116</td>
<td></td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modis</td>
<td>High Level Requirements (HLR)</td>
<td>49</td>
<td>68</td>
<td>C</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Low Level Requirements (LLR)</td>
<td>49</td>
<td></td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pine</td>
<td>High Level Requirements (HLR)</td>
<td>49</td>
<td>100</td>
<td>C</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Use Cases (UC)</td>
<td>51</td>
<td></td>
<td>English</td>
<td></td>
<td></td>
</tr>
<tr>
<td>iTrust</td>
<td>Use Cases (UC)</td>
<td>34</td>
<td>80</td>
<td>Java</td>
<td>English</td>
<td></td>
</tr>
<tr>
<td></td>
<td>JSP</td>
<td>47</td>
<td></td>
<td>English</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.1. Research Questions and Study Design

The study aims at addressing the following research questions:

**RQ_1** *To what extent does the smoothing filter improve the accuracy of traceability recovery methods?* This research question aims at analyzing whether, and to what extent, the performances of IR based traceability recovery improve when the smoothing filter is incorporated in the traceability recovery process.

**RQ_2** *How effective is the smoothing filter in filtering out non-relevant words, as compared to stop word removal?* This research question aims at investigating how the noise removal capability of the smoothing filter compares with that of stop word removal, either considering a standard (e.g., English or Italian) stop word list, or a customized stop word list (i.e., considering the case in which human intervention is required to remove words belonging to artifact templates by manually adding them to a stop word list). We are interested in understanding whether the use of smoothing filters is able to let developers effectively recover traceability links without a manual intervention on the indexing process.

To answer **RQ_1** we use different IR methods (VSM, LSI, and JS) to compute the textual similarity between artifact pairs, with and without using the smoothing filter. We are interested to evaluate the effect of the smoothing filter in the context of a full IR-based traceability recovery process described in Section 2, that is, documents are processed by stop word removal and stemming before applying the filter. 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
Table 2: Tracing activities performed on the object repositories.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Tracing Activity</th>
<th># Correct</th>
<th># Possible Links</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Links (oracle)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EasyClinic</td>
<td>$A_1$</td>
<td>93</td>
<td>1,410</td>
<td>Tracing use cases (UC) onto code classes (CC)</td>
</tr>
<tr>
<td></td>
<td>$A_2$</td>
<td>69</td>
<td>940</td>
<td>Tracing UML interaction diagrams (ID) onto code classes (CC)</td>
</tr>
<tr>
<td></td>
<td>$A_3$</td>
<td>200</td>
<td>9,400</td>
<td>Tracing test cases (TC) onto code classes (CC)</td>
</tr>
<tr>
<td>eTour</td>
<td>$A_4$</td>
<td>204</td>
<td>9,628</td>
<td>Tracing use cases (UC) onto code classes (CC)</td>
</tr>
<tr>
<td>Modis</td>
<td>$A_5$</td>
<td>26</td>
<td>926</td>
<td>Tracing high-level requirements (HLR) onto low-level requirements (LLR)</td>
</tr>
<tr>
<td>Pine</td>
<td>$A_6$</td>
<td>246</td>
<td>2,499</td>
<td>Tracing high-level requirements (HLR) onto use cases (UC)</td>
</tr>
<tr>
<td>i-Trust</td>
<td>$A_7$</td>
<td>58</td>
<td>1,551</td>
<td>Tracing use cases (UC) onto Java Server Pages (JSP)</td>
</tr>
</tbody>
</table>

As one can notice from Table 2, the seven traceability recovery activities represent only a subset of all possible combinations of artifacts. For some projects (e.g., Pine or Modis) only some artifacts were available. In other cases, e.g., EasyClinic, we choose to trace all artifacts onto source code, which represent a typical traceability recovery activity performed during software maintenance. Clearly, there may be other important recovery activities worthwhile of being considered (e.g., tracing requirements onto test cases). However such cases require specific treatments out of scope for this study. In addition, Settimi et al. [34] recommend tracing artifacts belonging to subsequent phases of the software development cycle.

To address RQ$_2$, we are interested to evaluate the benefits of smoothing filters as an alternative to the stop word removal, as well as in their combination. We analyze and compare the performances of a traceability recovery process when instantiated with the following filtering variants:

- **Standard stop word removal**: a standard English or Italian stop word list is used. For source code artifacts, such a list is enriched with programming language keywords (for C and Java systems) / pre-defined
tags for JSP systems such as i-Trust. Differently from a customized list (mentioned below), such a standard list is available out-of-the-box, and does not require a manual intervention.

- **Customized stop word removal**: the standard stop word list is enriched with a customized list of terms that are specific of the project and of the kinds of artifacts. This is the same stop word removal process used for **RQ1**. Building such customized lists took, on average, about two hours for each project. During software development and evolution, such a process has to be repeated every time artifacts change or new artifacts are added, because such artifacts could bring further words (to be removed) not considered before.

- **Smoothing filter**: the smoothing filter is used instead of stop word removal.

- **Standard stop word removal + smoothing filter**: we apply, in sequence, standard stop word removal and then the smoothing filter.

- **Customized stop word removal + smoothing filter**: as the previous one, but considering the customized stop word list instead of the standard one.

In all the experiments stemming is also applied during the indexing process. As for **RQ1**, we experimented with three different IR methods, each of which is instantiated with five different filtering strategies. Each combination of IR method and filtering strategy is then applied to seven different tracing activities.

### 4.2. Analysis Method

To answer **RQ1** we compare the accuracy of VSM, LSI, and JS with and without the use of the smoothing filter. In detail, we use a tool that takes as input the ranked list of candidate links produced by a traceability recovery method (e.g., VSM) 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 traceability recovery activity (see Table 2), the classification process starts from the top of the ranked list and stops when all correct links are recovered.
Once we have obtained the list of correct links and false positives, we first compare the accuracy of the different treatments at each point of the ranked list using two well-known IR metrics, namely recall and precision. Recall measures the percentage of correct links retrieved, while precision measures the percentage of links retrieved that are correctly identified:

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

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

where correct and retrieved represent the set of correct links and the set of links retrieved by the tool, respectively.

To provide statistical support to our findings we also use statistical tests to check whether the number of false positives retrieved by one recovery method with filter (e.g., VSM\textsubscript{Filter}) is significantly lower than the number of false positives retrieved by the same recovery method without filter (e.g., VSM) where the number of correct links retrieved is the same (i.e., at the same level of recall). The dependent variable of our study is the number of false positives (FP) retrieved by the traceability recovery method computed at each correct link identified when scrolling down the ranked list. Since the number of correct links is the same when comparing different methods on the same pairs of artifact types being traced (i.e., the data is paired, and we are dealing with dependent samples), we use the Wilcoxon Rank Sum test [41] to test the following null hypothesis:

\[H_{01}: \text{The use of a smoothing filter does not significantly reduce the number of false positives retrieved by a traceability recovery method for each correct link retrieved.}\]

Basically, we pairwise compare the difference between the number of false positives one has to analyze—for each true positive link—when using the filter and when not. We use a one-tailed test as we are interested to test whether the filter 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).
Then, we estimate the magnitude of the improvement of accuracy in terms of false positives reduction when using the smoothing filter. This estimation is computed using the Cliff’s Delta \( (d) \) [42], a non-parametric effect size measure for ordinal data, which indicates the magnitude of the effect of the main treatment on the dependent variables. For dependent samples (i.e., false positive distributions), it is defined as the probability that a randomly selected member of one sample has a higher response than a randomly selected member of the second sample, minus the reverse probability. Formally, it is defined as

\[
d = Pr(\text{FP}_1 > \text{FP}_2) - Pr(\text{FP}_2 > \text{FP}_1)
\]

where \( \text{FP}_1 \) is a member of population one and \( \text{FP}_2 \) is a member of population two. A sample estimate of this parameter can be obtained by enumerating the number of occurrences of a sample one member having a higher response value than a sample two member, and the number of occurrences of the reverse. This gives the sample statistic

\[
d = \frac{|\text{FP}_1 > \text{FP}_2| - |\text{FP}_2 > \text{FP}_1|}{|\text{FP}_1||\text{FP}_2|}
\]

Cliff’s Delta ranges in the interval \([-1 \ldots 1]\). It is equal to +1 when all values of one group are higher than the values of the other group and -1 when reverse is true. Two overlapping distributions would have a Cliff’s Delta equal to zero. The effect size is considered small for \( d < 0.33 \), medium for \( 0.33 \leq d < 0.474 \) and large for \( d \geq 0.474 \) [43]. We chose the Cliff’s Delta \( d \) effect size as it is appropriate for our variables (in ratio scale) and given the different levels (small, medium, large) defined for it, it is quite easy to be interpreted. We are aware that Cliff’s \( d \) is obtained by averaging across 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 [15], and such a metric has been used in previous traceability recovery work [32].

To address RQ$_2$, we compare the recovery accuracy of the different filtering configurations described in Section 4.1 (i.e., \textit{standard stop word removal}, \textit{customized stop word removal}, \textit{smoothing filter}, \textit{standard stop word removal + smoothing filter}, and \textit{customized stop word removal + smoothing filter}). We perform a pairwise comparison of the number of false positives recovered for each correct link identified by the different filtering configurations.
using the Wilcoxon Rank sum test (as also done for RQ₁). In this case we perform a two-tailed test, as we need to investigate the difference in both direction (i.e., the first filtering configuration can be better than the second, or vice versa). We report results of the pairwise comparison (i.e., results of the Wilcoxon test as well as Cliff’s Delta effect size values). Since we perform multiple tests on the same data, we correct p-values using the Holm’s correction procedure [44]. This procedure sorts the p-values resulting from \( n \) tests in ascending order, multiplying the smallest by \( n \), the next by \( n - 1 \), and so on. The hypothesis being tested is:

\[
H_{02}: \text{The five different filtering configurations do not significantly differ in terms of the number of false positives retrieved for each correct link identified.}
\]

5. Empirical Study Results

This section reports and discusses on the results of our experiments with the goal of answering our research questions stated in Section 4.1. Raw data and working data sets are available for replication purposes⁵.

5.1. RQ₁: To what extent does the smoothing filter improve the accuracy of traceability recovery methods?

Table 3 reports the improvement of precision and the reduction of retrieved false positives obtained using the proposed filtering at different levels of recall. We report the precision (Prec) achieved and the number of false positives (FP) one has to analyze when achieving a recall of 40%, 60%, 80%, and 100%. That is, depending on the percentage of true links one wants to recover, we show how many false positives—which represent waste of effort—one has to analyze. Figure 3 shows, for some examples of traceability recovery activities (taken from EasyClinic and Pine) the precision/recall curves for different recovery methods (VSM, LSI, and JS). Note that this figure only shows six examples with the aim of visually comparing the performance of VSM, LSI and JS on an example of vertical traceability recovery (UC→CC on EasyClinic) and one example of horizontal traceability recovery (HLR→ UC on Pine). Other similar graphs are available in a longer technical report [45].

⁵http://www.distat.unimol.it/reports/filters
Table 3: Percentage of precision improvement and reduction of number of false positives at different level of recall.

<table>
<thead>
<tr>
<th>Data set</th>
<th>Traced artifacts</th>
<th>IR Method</th>
<th>Prec@Rec(20%)</th>
<th>Prec@Rec(40%)</th>
<th>Prec@Rec(60%)</th>
<th>Prec@Rec(80%)</th>
<th>Prec@Rec(100%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EasyClinic A1 (UC-CC)</td>
<td>VSM</td>
<td>+21.32%</td>
<td>-67%</td>
<td>+19.05%</td>
<td>-60%</td>
<td>+30.88%</td>
<td>-74%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>+17.09%</td>
<td>-60%</td>
<td>+19.78%</td>
<td>-64%</td>
<td>+32.90%</td>
<td>-77%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>+16.26%</td>
<td>-30%</td>
<td>+8.81%</td>
<td>-30%</td>
<td>+24.77%</td>
<td>-64%</td>
</tr>
<tr>
<td>EasyClinic A2 (ID-CC)</td>
<td>VSM</td>
<td>+17.56%</td>
<td>-67%</td>
<td>+19.68%</td>
<td>-59%</td>
<td>+21.96%</td>
<td>-62%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>+4.56%</td>
<td>-25%</td>
<td>+26.52%</td>
<td>-71%</td>
<td>+24.66%</td>
<td>-60%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>+15.69%</td>
<td>-52%</td>
<td>+16.54%</td>
<td>-52%</td>
<td>+15.74%</td>
<td>-47%</td>
</tr>
<tr>
<td>EasyClinic A3 (TC-CC)</td>
<td>VSM</td>
<td>+2.70%</td>
<td>-10%</td>
<td>-1.15%</td>
<td>+5%</td>
<td>+0.96%</td>
<td>-4%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>-</td>
<td>-</td>
<td>+5.14%</td>
<td>-30%</td>
<td>+5.23%</td>
<td>-22%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>+15.96%</td>
<td>-47%</td>
<td>+11.81%</td>
<td>-47%</td>
<td>+2.74%</td>
<td>-24%</td>
</tr>
<tr>
<td>e-Tour A4 (UC-CC)</td>
<td>VSM</td>
<td>+13.16%</td>
<td>-44%</td>
<td>+16.11%</td>
<td>-56%</td>
<td>+2.88%</td>
<td>-22%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>+6.03%</td>
<td>-25%</td>
<td>+15.15%</td>
<td>-55%</td>
<td>+2.68%</td>
<td>-24%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>+15.66%</td>
<td>-47%</td>
<td>+11.81%</td>
<td>-47%</td>
<td>+2.74%</td>
<td>-24%</td>
</tr>
<tr>
<td>Modis A5 (BLLR-LLR)</td>
<td>VSM</td>
<td>-</td>
<td>-</td>
<td>+5.14%</td>
<td>-30%</td>
<td>+3.23%</td>
<td>-22%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>-</td>
<td>-</td>
<td>+5.14%</td>
<td>-30%</td>
<td>+3.23%</td>
<td>-22%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>-</td>
<td>-</td>
<td>-3.15%</td>
<td>+19%</td>
<td>+2.89%</td>
<td>-13%</td>
</tr>
<tr>
<td>Pine A6 (BLLR-UC)</td>
<td>VSM</td>
<td>+18.70%</td>
<td>-53%</td>
<td>+15.28%</td>
<td>-50%</td>
<td>+20.67%</td>
<td>-64%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>+15.05%</td>
<td>-51%</td>
<td>+12.69%</td>
<td>-42%</td>
<td>+18.15%</td>
<td>-59%</td>
</tr>
<tr>
<td></td>
<td>JS</td>
<td>+6.06%</td>
<td>-25%</td>
<td>+10.75%</td>
<td>-31%</td>
<td>+8.51%</td>
<td>-32%</td>
</tr>
<tr>
<td>iTrust A7 (BLLR-UC)</td>
<td>VSM</td>
<td>-</td>
<td>-</td>
<td>+40.95%</td>
<td>-86%</td>
<td>+16.24%</td>
<td>-52%</td>
</tr>
<tr>
<td></td>
<td>LSI</td>
<td>+7.66%</td>
<td>-200%</td>
<td>+24.01%</td>
<td>-65%</td>
<td>+15.34%</td>
<td>-72%</td>
</tr>
</tbody>
</table>

Results indicate that the proposed smoothing filter is useful to improve the recovery accuracy of both Vector Space (VSM and LSI) and Probabilistic (JS) traceability recovery methods in all datasets. In most cases for a recall level smaller than 80%, it is possible to achieve an improvement of precision ranging between 10% and 30%. Such an improvement mirrors a considerable reduction of retrieved false positives that can be quantified between 35% and 75%. Such a result represents a substantial improvement, if we look at it from the perspective of a software engineer that is inspecting the ranked list of candidate traceability links. For example, when applying LSI with filter to trace use cases onto code classes of EasyClinic, the software engineer is able to trace 66 links (80% of recall) discarding only 38 false positives (63% of precision). Achieving the same level of recall without the use of the filter would require the software engineer discarding 96 false positives (i.e., 58 false positives more). A less evident improvement is achieved for the lowest recall percentile (i.e., between 80% and 100%). In this case, the improvement in terms of precision is around 1%. This result confirms that, when the goal is to recover all correct links, there is an upper bound for the performance improvements that is very difficult to overcome [8, 38].
Figure 3: Examples of Precision/Recall curves for different traceability recovery methods.

Figure 4 shows examples of precision/recall graphs obtained using the same IR method (i.e., LSI) for artifacts with different abstraction levels.
Figure 4: Examples of Precision/Recall curves for different traceability recovery activities.

Among all possible cases, we show one for each kind of traceability recovery activity (requirements onto use cases, use cases onto code, design onto code, and test cases onto code), considering the method (LSI) that achieves the best performance. Analyzing such precision/recall curves and the precision percentage variations in Table 3, it is also possible to observe that when tracing code classes onto test cases of EasyClinic, the proposed filter does not provide any improvement at all. Figure 5 shows an example of a test case extracted from EasyClinic where the filter prunes out (i.e., reduces their weight, thus limiting the contribution of the word to the similarity computation) most of the template words (in bold face), but also some other words contained in the test case (underlined). By definition, the filter removes high frequency words (e.g., template words) that usually occur in artifacts belonging to test cases. After such a pruning, very few words are left, and only two
<table>
<thead>
<tr>
<th>Test Case</th>
<th>Login Operator by Date: C05 login registered into SIO and 20/06/2003 incorrect password (higher number of characters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>0.02.000</td>
</tr>
<tr>
<td>Use Case</td>
<td>He performs the functions UcValOpe necessary to authenticate an Operator</td>
</tr>
<tr>
<td>Priority</td>
<td>High</td>
</tr>
<tr>
<td>Set up</td>
<td>the login Rocolo is recorded into SIO</td>
</tr>
</tbody>
</table>
| Description test | Input  
Login: Rocolo  
Password: rocolivirago                                                                 |
| Oracle    | Invalid Input: Password                                                                                               |
| Coverage  | valid classes: CE2  
invalid classes: CE9                                                                                               |

Figure 5: Example of an EasyClinic test case

of them are also contained in the classes to be traced onto such test case. For this reason, traceability recovery is *per se* very difficult here, and there is very little the filter can do to improve it. Thus, the smoothing filter is clearly not perfect, as it also prunes some probably relevant words (e.g., *Operator*, *Password*, or *SIO*) because they appear in several (about 50%) test cases.

The positive effect of the filter on each traceability link can be also graphically represented by the relation diagram shown in Figure 6 for the recovery activity $A_1$ ($UC\rightarrow CC$ in EasyClinic) and the recovery activity $A_5$ ($HLR\rightarrow LLR$ in Modis) using LSI as IR method. The diagram shows—with and without applying the smoothing filter—the precision computed every time a correct link is reached in the ranked list. 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 the filter on the ranked list of candidate links. In particular, the filter tends to increase the rank of correct links facilitating their identification. Indeed, when applying the filter, correct links are recovered with a higher precision when compared to the precision obtained without the smoothing filter. Figure 6 also indicates that the filter is not able to sensibly increase the rank of the last correct links in the ranked list (i.e., the ones shown in the bottom part of the figure).

An analysis of the last links in the ranked list reveals that such links are really challenging to recover using IR methods. This happened in cases where the artifacts to be traced share only few words (and in some cases no words), and/or have a low verbosity, thus their textual similarity is very low. For instance, the last link in Figure 6-a connects the use case *Login Patient* with the abstract class *Person*. The vocabulary overlap between these two
artifacts is lower than 10%. Figure 6-b clearly shows that, for the recovery activity $A_5$, the filter fails to produce any improvement for several links in the bottom part of the ranked list, as denoted by the many horizontal lines in the bottom part of the graph. For instance, the last link in Figure 6-b connects the HLR $SDP.2-1$ with the LLR $L1APR03-F.3.2.3-2$ of Modis. Such lower performances could be due to a limited verbosity of the Modis artifacts. Probably, in this case techniques based on query expansion [32, 33] could have helped to improve the performances.

Let us now analyze the performance improvement (if any) for the various traceability recovery activities $A_1$–$A_7$, and for different recovery methods. Table 4 reports results of the Wilcoxon test and the Cliff’s $d$ effect size, with the aim of statistically supporting results shown in Table 3, Figure 3 and Figure 4. Specifically, the Table shows Cliff’s $d$ values for all pairwise comparisons. Values where the Wilcoxon test indicated a significant difference—after correcting p-values using the Holm’s correction—are shown.
Table 4: Cliff’s $d$ for differences of the number of false positives before and after applying the filter. Values shown in bold face for comparisons where the Wilcoxon Rank Sum test indicates a significant difference. Also, we use use $S$, $M$, and $L$ to indicate small, medium and large effect sizes respectively.

<table>
<thead>
<tr>
<th>Repository</th>
<th>Traced artifacts</th>
<th>VSM</th>
<th>LSI</th>
<th>JS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Cliff’s $d$</td>
<td>Cliff’s $d$</td>
<td>Cliff’s $d$</td>
</tr>
<tr>
<td>EasyClinic</td>
<td>$A_1$ (UC→CC)</td>
<td>0.51$^L$</td>
<td>0.50$^L$</td>
<td>0.56$^L$</td>
</tr>
<tr>
<td></td>
<td>$A_2$ (ID→CC)</td>
<td>0.47$^M$</td>
<td>0.53$^L$</td>
<td>0.52$^L$</td>
</tr>
<tr>
<td></td>
<td>$A_3$ (TC→CC)</td>
<td>-0.23</td>
<td>-0.39</td>
<td>-0.47</td>
</tr>
<tr>
<td>e-Tour</td>
<td>$A_4$ (UC→CC)</td>
<td>0.47$^M$</td>
<td>0.62$^L$</td>
<td>0.52$^L$</td>
</tr>
<tr>
<td>Modis</td>
<td>$A_5$ (HLR→LLR)</td>
<td>0.51$^L$</td>
<td>0.43$^M$</td>
<td>0.45</td>
</tr>
<tr>
<td>Pine</td>
<td>$A_6$ (HLR→UC)</td>
<td>0.58$^L$</td>
<td>0.59$^L$</td>
<td>0.58$^L$</td>
</tr>
<tr>
<td>i-Trust</td>
<td>$A_7$ (HLR→CC)</td>
<td>0.21$^{S}$</td>
<td>0.36$^M$</td>
<td>0.28$^S$</td>
</tr>
</tbody>
</table>

in bold face. Specifically, results shown in Table 4 confirm findings of Table 3 and indicate that:

- the smoothing filter does not produce any significant improvement for activity $A_3$ (i.e., when tracing TC onto CC in EasyClinic);

- in all other cases, the filter introduces significant benefits, and, with few exception, the effect size is large for all traceability activities and for all recovery methods. In essence, filtering works well not only for VSM, but also for techniques such as LSI and JS that usually exhibit better performance than VSM. This is because while techniques such as LSI are good to deal with problems such as polysemy and synonymy, the smoothing filter helps to deal with noise in software artifacts due to the presence of document templates/technical words. Moreover, for JS the smoothing filter helps to remove useless words that alter the probability distribution of words in the artifacts corpus, because of their high frequency in documents belonging to the same kind of artifacts (i.e., templates/technical words). Indeed, by definition highly frequent words drastically reduce the information entropy of the documents where they appear [29];

- for the recovery activity $A_7$ (i.e., when tracing HLR onto CC in i-Trust) the effect size is small, and it is medium—at least for LSI and JS methods—for activity $A_5$ (i.e., tracing HLR onto LLR in Modis). In both cases, this is due to the low performances of the filter for links
in the bottom part of the ranked list, as it can be seen in Table 3, and also in Figure 6 -b for Modis.

We can conclude this first part of results stating that concerning **RQ1** the smoothing filter significantly improves the performances of all traceability recovery methods we investigated (VSM, LSI, JS), with an effect size that is large in most cases. In addition, the filter works well on all kinds of artifacts, except for situations where the artifacts to be traced exhibit a very low overlap in terms of common words after applying the filter, as it happens when tracing test cases onto source code. In summary, with the only exception mentioned, we can reject $H_{01}$.

### 5.2. RQ2: How effective is the smoothing filter in filtering out non-relevant words, as compared to stop word removal?

In this section we aim at answering **RQ2**, by comparing the pruning contribution of different noise removal configurations. We statistically compare the different distributions of false positives achieved by the various configurations for all the planned traceability activities ($A_1 - A_7$) and using LSI only as traceability recovery technique. This choice is due to the fact that (i) as shown for **RQ1**, the effect of the filter does not change with the IR technique, and (ii) LSI often outperforms the other techniques. We also performed the same computations using VSM and JS, obtaining similar conclusions (tables can be found in the online technical report [45]).

Table 5 reports Cliffs $d$ obtained in pairwise comparisons of the various configurations, highlighted in bold face when results of the Wilcoxon Rank Sum test$^6$ are statistically significant. By looking at the table, we can notice that:

- the customized stop word list produces, in some cases, results significantly better than the standard stop word list. This happens for the activity $A_3$ of EasyClinic (i.e., tracing test cases onto code classes) because the customized stop word list is able to filter template words out better than the filter. Also, we obtain the same results, for similar reasons, for activity $A_5$ of Modis (i.e., tracing HLR onto LLR). In both these cases the effect size is high;

---

$^6$Adjusted using the Holm’s correction.
Table 5: Comparison of different “noise removal” configurations: Cliff’s $d$ effect size (shown in bold face for comparisons where the Wilcoxon Rank sum test indicates a significant difference). Also, we use $S$, $M$, and $L$ to indicate small, medium and large effect sizes respectively.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Customized stop word list Vs. Standard stop word list</td>
<td>-0.04</td>
<td>0.13</td>
<td>1.12</td>
<td>-0.25$^d$</td>
<td>1.05$^d$</td>
<td>0.19$^d$</td>
<td>-0.25$^d$</td>
</tr>
<tr>
<td>Smoothing Filter Vs. Standard stop word list</td>
<td>0.15$^e$</td>
<td>0.08</td>
<td>-0.79$^d$</td>
<td>-0.24$^d$</td>
<td>-0.46$^d$</td>
<td>0.82$^d$</td>
<td>-0.07</td>
</tr>
<tr>
<td>Smoothing Filter Vs. Customized stop word list</td>
<td>0.15$^e$</td>
<td>0.08</td>
<td>-0.76$^d$</td>
<td>0.50$^d$</td>
<td>-0.47</td>
<td>0.85$^d$</td>
<td>0.19$^d$</td>
</tr>
<tr>
<td>Standard stop word list + Smoothing Filter Vs. Standard stop word list</td>
<td>0.53$^d$</td>
<td>0.50$^d$</td>
<td>0.29$^d$</td>
<td>-0.25$^d$</td>
<td>-0.37</td>
<td>0.60$^d$</td>
<td>0.34$^d$</td>
</tr>
<tr>
<td>Standard stop word list + Smoothing Filter Vs. Customized stop word list</td>
<td>0.54$^d$</td>
<td>0.50$^d$</td>
<td>-0.39$^d$</td>
<td>0.41$^d$</td>
<td>-0.39$^d$</td>
<td>0.61$^d$</td>
<td>0.57$^d$</td>
</tr>
<tr>
<td>Standard stop word list + Smoothing Filter Vs. Smoothing Filter</td>
<td>0.21$^s$</td>
<td>0.74$^d$</td>
<td>1.90$^d$</td>
<td>-0.15$^s$</td>
<td>0.36$^d$</td>
<td>0.33$^d$</td>
<td>0.28$^d$</td>
</tr>
<tr>
<td>Customized stop word list + Smoothing Filter Vs. Standard stop word list</td>
<td>0.51$^d$</td>
<td>0.53$^d$</td>
<td>-0.31$^d$</td>
<td>0.20$^s$</td>
<td>0.33$^d$</td>
<td>0.59$^d$</td>
<td>0.11</td>
</tr>
<tr>
<td>Customized stop word list + Smoothing Filter Vs. Customized stop word list</td>
<td>0.50$^d$</td>
<td>0.53$^d$</td>
<td>-0.39$^d$</td>
<td>0.62$^d$</td>
<td>0.43</td>
<td>0.59$^d$</td>
<td>0.36$^d$</td>
</tr>
<tr>
<td>Customized stop word list + Smoothing Filter Vs. Smoothing Filter</td>
<td>0.21$^s$</td>
<td>0.77$^d$</td>
<td>1.49$^d$</td>
<td>0.53$^d$</td>
<td>0.78$^d$</td>
<td>0.30$^d$</td>
<td>0.23$^d$</td>
</tr>
<tr>
<td>Customized stop word list + Smoothing Filter Vs. Standard stop word list + Smoothing Filter</td>
<td>0.08</td>
<td>0.21</td>
<td>-0.36$^d$</td>
<td>0.43$^d$</td>
<td>0.62$^d$</td>
<td>-0.08</td>
<td>-0.13</td>
</tr>
</tbody>
</table>

- the smoothing filter per se does not perform better than the stop word lists, as there are case where the smoothing filter performs better (e.g., activity $A_6$) and cases where it performs worse (e.g., activity $A_3$);

- the combination of standard stop word list and smoothing filter is, with few exceptions, able to outperform single filters, including the customized stop word list. In this case, for activities $A_1$, $A_2$, $A_4$, $A_6$, and $A_7$, the difference is in favor of the combination standard stop word list + smoothing filter, with a high effect size. The only cases where the difference is slightly in favor of the customized list (medium effect size), are activities $A_3$ and $A_5$, congruently with the other results obtained so far. In general, this is a very important result, because the combination of the standard stop word list and the smoothing filter (which does not require any manual intervention) is able to outperform the customized stop word list, which instead requires at least two hours of effort for each system;

- finally, the combination of filter and customized stop word list is significantly better than the combination of filter and standard stop word list only for $A_1$ (negligible effect size), $A_4$ (medium effect size), and $A_5$ (high effect size).

In summary, we can reject $H_{02}$, because we were able to show that different filtering configurations exhibit significantly different performances. Be-
sides that, we can answer \textbf{RQ}_2 stating that the smoothing filter is able to remove noise as well as a standard stop word list, and that the combination of standard stop word list + filter is able to perform slightly better than a customized stop word list.

\section*{6. Threats to Validity}

This section discusses the threats to validity that could affect our results. Threats to \textit{construct validity} concern the relationship between theory and observation. We used widely adopted metrics (precision, recall, and number of false positives) for assessing the IR technique as well as their improvement when using the smoothing filter. The accuracy of the oracle (traceability matrix) used to evaluate the tracing accuracy could also affect our results. We used traceability matrices provided by the original developers to mitigate such a threat. For the two students' projects, EasyClinic and eTour, we also validated the links during review meetings involving the original development team together with PhD students and academic researchers.

Threat to \textit{internal validity} could be due to confounding factors affecting our results. Although we show a significant improvement when using the smoothing filter, there could be other factors causing such an improvement. For example, the improvement could depend on other stages of the traceability recovery process, e.g., stop word removal or stemming, or else the words filtered out were “by chance” the one causing low traceability recovery performances. However, we have mitigated the threat by showing that the approach works even if other noise removal techniques are used, such as (i) stop word lists and functions; (ii) stemming; (iii) \textit{tf-idf} and (iv) LSI. Also, in \textbf{RQ2} we separately (and jointly) analyze the filtering performances of stop word lists and of the smoothing filter. Also, to mitigate the possibility that such results were obtained by chance, we applied the filter on seven different recovery activities from five projects.

For what concerns \textit{conclusion validity}, we support our findings by using proper statistics (the Wilcoxon non-parametric test in particular). When multiple tests were performed on the same data sets, we use the Holm’s correction to adjust p-values. Furthermore, we use an effect size measure (Cliff’s Delta) to provide evidence of the practical significance of results, other than of their statistical significance.

Threats to \textit{external validity} concern the generalization of our findings. A relevant threat is related to the repositories used in the empirical study. The
chosen repositories have the advantage of containing various kinds of artifacts (use cases, requirements, design documents, source code, test cases, JSP pages). Also, in our experiments we have considered students’ (EasyClinic, e-Tour, i-Trust), as well as industrial (Modis) 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 [24], and the same happened for Modis [37, 19] and iTrust [46]. 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.

7. Lessons Learned and Conclusion

This paper proposed the use of a smoothing filter to improve the recovery accuracy of IR-based traceability recovery approaches. The idea is inspired by digital signal processing—e.g., image or sound processing—where smoothing is used to remove the noise from a signal. In the context of software traceability recovery, smoothing helps to remove terms that do not bring relevant information about the document and that, however, could not simply be removed by means of stop word lists and functions.

The usefulness of a smoothing filter has been evaluated in a case study conducted on five software repositories, namely EasyClinic, eTour, Modis, Pine, and i-Trust. The obtained results allow us to summarize the following pieces of evidence:

- **the smoothing filter generally improves the accuracy of traceability recovery methods, based on probabilistic and vector space models (RQ$_1$).** We compared the recovery accuracy of different traceability recovery methods, namely VSM, LSI, and JS, with and without the application of the smoothing filter. The achieved results indicated an improvement of precision ranging from 10% to 30% at the same level of recall. This mirrors a notable decrement (from 35% to 75%) of false positives that need to be discarded by the software engineer indicating the potential usefulness of smoothing filters in IR-based traceability recovery processes;

- **the kind of the source and target artifacts influences the effectiveness of the smoothing filters (RQ$_1$).** We also investigated whether the char-
acteristics of the traced software artifacts, e.g., the kind of artifact (requirement, use case, design document, source code, test case), as well as its verbosity, play any role in the effectiveness of the smoothing filter. We observed that there is only one case (out of seven) where the smoothing filters seemed to be not useful at all, i.e., when tracing test cases onto use cases on the EasyClinic dataset. In this particular case we observed that after applying the filter very few words are left in the test cases, and only few of them are also contained in the classes to be traced onto such test cases. For this reason, traceability recovery is per se very difficult here, and there is very little the filter can do to improve it;

• the smoothing filter can be used in combination with lightweight stop word lists to avoid the definition of ad-hoc stop word lists (RQ2). In our study we instantiated and compared the impact on the recovery accuracy of four different stop word filtering processes: standard stop word removal, customized stop word removal, smoothing filters, and their combinations, i.e., smoothing filter plus stop word lists. The analysis of the results indicated that generally the best recovery accuracy is obtained combining smoothing filters with a customized stop word list. However, the results achieved combining smoothing filters with a standard stop word list are (i) comparable with the best results and (ii) better than the accuracy achieved with only stop word lists (including the customized stop word list). This is an important result, because the use of standard (and non-domain specific) stop word list does not always ensure good recovery accuracy. Usually, stop word lists need to be customized in order to obtain a domain-specific stop word list. However, this process is tedious and time consuming (in our study, it required about two hours for each system), since the software engineer needs to manually identify terms that are not useful to characterize the semantic of the artifacts in the specific application domain. Thus, the use of smoothing filter overcomes such a problem providing a way to automatically remove the “noise” of non-domain specific terms that are not useful to characterize the content of software artifacts.

In summary, the proposed filter introduces a substantial improvement in the results produced by the state-of-the-art IR-based traceability recovery approaches. In essence, such improvements substantially reduce the number
of candidate links a software engineer has to manually analyze to cover 80%-100% of the true positive links. As Figures 3 and 4 show, by adopting the filter it would be possible—with very few exceptions—to achieve a recall of at least 80% with a precision between 40% and 50% (or more). However, it must be clear that, regardless of such improvements, the applicability of IR-based traceability recovery still depends on the lexicon quality and consistency, and should be possibly enforced during the development and maintenance activities [40].

For future work, we plan to perform an empirical comparison of smoothing filters with other techniques that help to enhance the performances of traceability recovery (described in Section 2.2), to better understand their complementarity as well as possible interactions. Also we plan to extend the application of smoothing filters to other IR-based software engineering approaches. Specifically, areas where it would be worthwhile to apply smoothing filters include feature location [3, 4], as well as approaches for bug prediction or for refactoring exploiting the notions of conceptual cohesion and coupling [5].

Acknowledgements

We would like to thank Jane Huffman Hayes, Wei-Keat Kong, Wenbin Li, Hakim Sultanov, and Alex Wilson for providing us with the Pine dataset.

References


34


