Trustrace: Mining Software Repositories to Improve the Accuracy of Requirement Traceability Links

Nasir Ali, *Student Member*, *IEEE*, Yann-Gaël Guéhéneuc, *Senior Member*, *IEEE*, and Giuliano Antoniol, *Member*, *IEEE*

Abstract—Traceability is the only means to ensure that the source code of a system is consistent with its requirements and that all and only the specified requirements have been implemented by developers. During software maintenance and evolution, requirement traceability links become obsolete because developers do not/cannot devote effort to updating them. Yet, recovering these traceability links later is a daunting and costly task for developers. Consequently, the literature has proposed methods, techniques, and tools to recover these traceability links semi-automatically or automatically. Among the proposed techniques, the literature showed that information retrieval (IR) techniques can automatically recover traceability links between free-text requirements and source code. However, IR techniques lack accuracy (precision and recall). In this paper, we show that mining software repositories and combining mined results with IR techniques can improve the accuracy (precision and recall) of IR techniques and we propose Trustrace, a trust-based traceability recovery approach. We apply Trustrace on four medium-size open-source systems to compare the accuracy of its traceability links with those recovered using state-of-the-art IR techniques from the literature, based on the Vector Space Model and Jensen-Shannon model. The results of Trustrace are up to 22.7 percent more precise and have 7.66 percent better recall values than those of the other techniques, on average. We thus show that mining software repositories and combining the mined data with existing results from IR techniques improves the precision and recall of requirement traceability links.

Index Terms—Traceability, requirements, feature, source code, repositories, experts, trust-based model

1 Introduction

Requirement traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of ongoing refinement and iteration in any of these phases)" [1]. Traceability links between the requirements[1] of a system and its source code are helpful in reducing system comprehension effort. They are also essential to ensuring that a system's source code is consistent with its requirements and that all and only the specified requirements have been implemented by developers. Yet, during software maintenance and evolution, as developers add, remove, or modify features, requirement traceability links become obsolete because developers do not/cannot devote effort to updating them [2]. Yet, recovering these traceability links later is a daunting and costly task for developers. Consequently, the literature has proposed methods, techniques, and tools to recover these traceability links semi-automatically or automatically [3].

Requirements traceability has received much attention over the past decade in the scientific literature. Many researchers used information retrieval (IR) techniques, e.g., [2], [3], [4], to recover traceability links between high-level documents, e.g., requirements, manual pages, and design documents, and low-level documents, e.g., source code and UML diagrams [3], [4], [5], [6]. IR techniques assume that all software artifacts are/can be put in some textual format. Then, they compute the textual similarity between two software artifacts, e.g., the source code of a class and a requirement. A high textual similarity means that the two artifacts probably share several concepts [3] and that, therefore, they are likely linked to one another.

The effectiveness of IR techniques is measured using IR metrics: recall, precision, or some average of both, like the F_1 score [3], [5], [7]. For a given requirement, recall is the percentage of correct recovered links over the total number of pertinent, expected links, while precision is the percentage of correct recovered links over the total number of recovered links. High recall could be achieved by linking each requirement to all source code entities (classes, structures, methods, and functions), but precision would be close to zero. High precision could be achieved by reporting only obvious links, but recall would be close to zero. Either extreme cases are undesirable because developers then would need to manually review numerous candidate links to remove false positives and/or study the source code to recover missing links [3]. Hence, the

1. Without loss of generality with other textual documentation, in the following we use textual requirements as high-level documents and source code as low-level documents.

• *The authors are with the Département de Génie Informatique et Génie Logiciel, École Polytechnique de Montréal, C.P. 6079, succursale Centre-Ville Montréal, QC H3C 3A7, Canada.*
E-mail: {nasir.ali, yann-gael.gueheneuc}@polymtl.ca, antoniol@ieee.org.

Manuscript received 20 Mar. 2012; revised 4 July 2012; accepted 22 Oct. 2012; published online 26 Oct. 2012.
Recommended for acceptance by T. Tamai.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-03-0067.
Digital Object Identifier no. 10.1109/TSE.2012.71.

literature has proposed IR-based techniques that sought a balance between precision and recall.

During software maintenance and evolution, developers often evolve requirements and source code differently. Indeed, they often do not update requirements and requirement-traceability links with source code. Requirements and source code consequently diverge from each other, which decreases their textual similarity. Thus, IR techniques (or any combination thereof [8], [9]) typically produce links with low precision and/or recall because, due to their very nature, they depend on the textual similarity between requirements and source code [10]. Yet, while developers may not evolve requirements in synchronization with source code, they frequently update other sources of information, including CVS/SVN repositories, bug-tracking systems, mailing lists, forums, and blogs. We believe that we can exploit these other sources of information to build improved traceability-recovery approaches.

Consequently, we conjecture that: 1) We can mine software repositories, e.g., CVS/SVN repositories, to support the traceability recovery process and improve the precision and recall of IR-based traceability recovery approaches; 2) we can think of heterogeneous sources of information as experts whose opinions we can combine using a trust model to discard/rerank the traceability links provided by the IR techniques to improve accuracy; and 3) we can use an automatic, dynamic, per-link weighting technique rather than some global static weights to combine the opinions of experts to avoid the need of manually built oracles to tune weights.

Consequently, we design, implement, and evaluate Trustrace, a traceability-recovery approach between requirements and source code, which we use to support our conjectures. Trustrace uses heterogeneous sources of information to dynamically discard/rerank the traceability links reported by an IR technique. Trustrace consists of three parts:

1. *Histrace* mines software repositories to create links between requirements and source code using information from the repositories. Histrace stores all the recovered links between requirements and software repositories in dedicated sets, e.g., $Histrace_{commits}$ and $Histrace_{bugs}$, which are considered as experts whose opinions will be used to discard/rerank baseline traceability links. For example, Histrace mines CVS/SVN repository to link requirements and source code using commit messages and provide the set of expert $Histrace_{commits}$.
2. Trumo combines the requirement traceability links obtained from an IR technique and discards/reranks them using the an expert's opinions and a trust model inspired by web-trust models [11], [12], [13], [14]. It compares the similarity of the recovered links with those provided by the experts and with the number of times that the link appears in each expert's set. It is not tied to any specific IR-based traceability-recovery approach and can use any expert's opinion to adjust the ranking of recovered links. For example, the experts can be $Histrace_{commits}$ and/or $Histrace_{bugs}$, taking advantage of CVS/SVN repositories and bug-tracking systems.
3. DynWing computes and assigns weights to the experts in the trust model dynamically, i.e., on a per-link basis. As combining different experts' opinions is still an active research area [9], researchers used either static weights computed using oracles [2], [8] or other techniques, e.g., principal component analysis (PCA) [9], for each source of information. However, defining a static weight for all the links for each expert may not be feasible because oracles may not be available or might be too costly to build. Thus, dynamic-weighting techniques are promising to assign weights for each link. DynWing analyzes each expert's similarity value for each link and assigns weights according to these values.

We empirically evaluate Trustrace on four medium-size open-source systems, i.e., *jEdit v4.3*, *Pooka v2.0*, *Rhino v1.6*, and *SIP Communicator v1.0-draft*, to compare the accuracy of its recovered requirement traceability links with those of state-of-the-art IR techniques. As state-of-the-art IR techniques, we choose the Vector Space Model (VSM), a representative of the algebraic family of techniques, and Jensen-Shannon model (JSM), a representative of the probabilistic family of techniques. We use the IR measures of precision, recall, and the F_1 score. We also compare two different weighting techniques: PCA and DynWing. We thus report evidence that Trustrace improves, with statistical significance, the precision and recall of the recovered traceability links.

Hence, we found the evidence supporting our three conjectures about the benefits of 1) mining software repositories and considering the links recovered through these repositories as experts, 2) using a trust model inspired by web-trust models to combine these experts' opinions, and 3) weighting the experts' opinions dynamically for each link recovered using an IR technique. We also bring the following contribution with respect to our previous work [2]:

- We implement and use two experts, i.e., $Histrace_{commits}$ when mining CVS/SVN and $Histrace_{bugs}$ when mining bug reports.
- We propose DynWing, a dynamic weighting technique to automatically assign weights to different experts on a per-link basis.
- We compare DynWing with a PCA-based weighting technique and with a static weighting technique to analyze the potential improvement of using different weighting techniques.
- We study the impact of Trustrace on another IR technique, i.e., the Jensen-Shannon similarity model.
- We perform detailed statistical analyses of the data distribution to select an appropriate statistical test as well as to measure the effect size of Trustrace over JSM and VSM.
- We perform experiments on four medium-size open-source systems, i.e., two additional systems, *jEdit* and *Rhino*, in addition to *Pooka* and *SIP*, to analyze the impact of Trustrace.

Section 2 describes our approach, Trustrace, and our three novel techniques: Histrace, Trumo, and DynWing. Section 3 presents our empirical evaluation of Trustrace, while Section 4 reports its results. Section 5 provides a

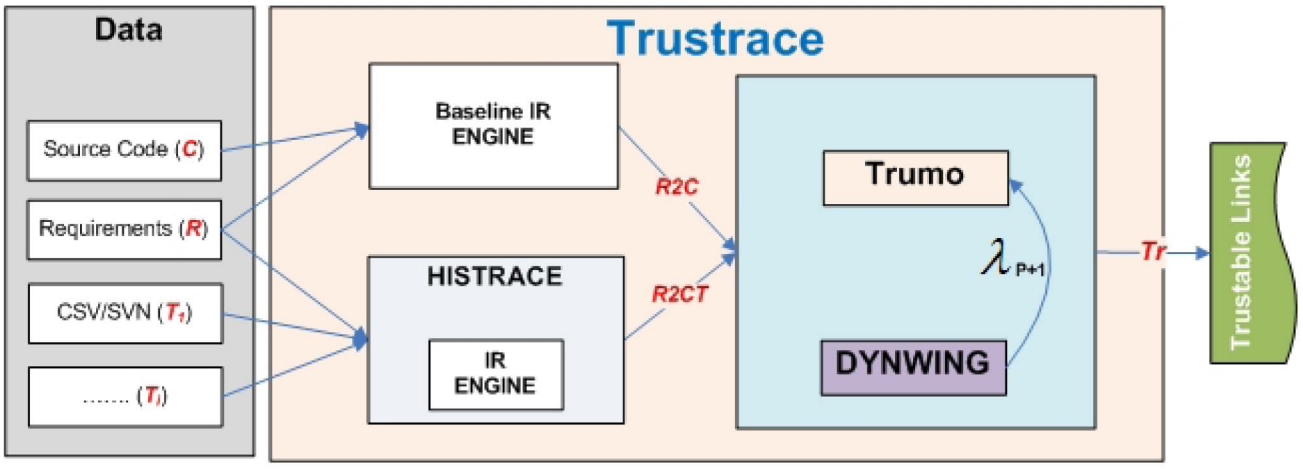

Fig. 1. Trust-based requirement traceability process.

discussion of the results and a qualitative analysis. Section 6 puts our work in perspective with respect to previous work. Finally, Section 7 concludes and sketches future work.

2 TRUSTRACE: TRUST-BASED TRACEABILITY

We now present Trustrace. Trustrace uses software repositories, e.g., CVS/SVN repositories and bug-tracking systems, as experts to trust more or less some baseline links recovered by an IR technique and, thus, to discard/rerank the links to improve the precision and recall of the IR-based techniques. Fig. 1 shows the high-level architecture of Trustrace, whose conceptual steps we detail in the following sections: Section 2.1 provides the definitions of the terms that we use to describe Trustrace; Section 2.2 describes Histrace, our traceability recovery technique based on mining software repositories; Section 2.3 describes our trust model; and Section 2.4 details our dynamic weighting technique.

2.1 In a Nutshell

In the following, without loss of generality, we target object-oriented systems and use classes as representative of source code files. We also consider classes because considering packages is likely to be too coarse grained, as a package contributes to the implementation of several requirements, while considering methods is likely to be too fine grained as a method only participates in the implementation of some requirement(s) and rarely implements them entirely. Moreover, CVS/SVN software repositories only consider files, not packages or methods.

2.1.1 Definitions

In Trustrace, we represent a traceability link as a triple {source document, target document, similarity} and we use the following notations. Let $R = \{r_1, \ldots, r_N\}$ be a set of requirements and $C = \{c_1, \ldots, c_M\}$ be a set of classes supposed to implement these requirements. Let $\mathcal{T} = \{T_1, \ldots, T_P\}$ be a collection of sets where each $T_i = \{t_1, \ldots, t_{N_i}\}$ is a set of homogeneous pieces of information, e.g., the set of all CVS/SVN commits or of all bug reports for a given system. P is the total number of experts.

Then, let us assume that, for each $T_i \in \mathcal{T}$, we define a function δ_{T_i} mapping one element of T_i into a subset of C. For example, if T_i is a set of bug reports, then, for a given bug report t_k, $\delta_{T_i}(t_k)$ returns the set of classes affected by t_k, with $\delta_{T_i}(t_k) \subseteq C$.

Let $R2C$ be the set of baseline traceability links recovered between R and C by any standard IR technique, such as VSM, and further, assume that, for each set $T_i \in \mathcal{T}$, we build a set $R2CT_{i,r_j,t_k}$ for each expert T_i as follows:

$$R2CT_{i,r_j,t_k} = \{(r_j, c_s, \sigma_i'(r_j, t_k)) | c_s \in \delta_{T_i}(t_k) \ \& \ t_k \in T_i\}.$$

Finally, let us define two functions α and ϕ. The first function, $\alpha(r_j, c_s, \sigma'(r_j, c_s))$, returns the pair of documents (or set of pairs) involved in a link, e.g., requirements and source code, i.e., (r_j, c_s). The second function, $\phi(r_j, c_s, \sigma'(r_j, c_s))$, returns the similarity score $\sigma'(r_j, c_s)$ of the link.

Given these definitions, Trustrace works as follows: It first builds the set $R2C$ from R and C using an IR technique. Then, it calls Histrace to build the sets $R2CT_{i,r_j,t_k}$ using CVS/SVN commit messages and bug reports. Second, it uses Trumo to evaluate the trustworthiness of each link using DynWing to compute the weights $\lambda_i(r_j, c_s)$ to assign to each link in the sets $R2CT_{i,r_j,t_k}$, and to rerank the similarity values of the link, using the experts' opinions $R2CT_{i,r_j,t_k}$.

2.1.2 Information Retrieval Technique

Trustrace uses IR techniques for two different purposes: 1) to create the baseline set of traceability links $R2C$, whose similarity values will be recomputed using Trumo and DynWing using the output of Histrace, and 2) to create the output sets of Histrace, $R2CT_{i,r_j,t_k}$.

IR techniques consider all the software artifacts as textual documents. They extract all the terms from the documents and compute the similarity between two documents based on the similarity of their terms and/or their distributions. With any IR technique, a high similarity value between two documents suggests a potential link between them. IR techniques take some preprocessed documents, as explained in the following, as input to build an $m \times n$ term-by-document matrix, where m is the number of all unique terms that occur in the documents and n is the number of documents in the corpus. Then, each cell of the matrix contains a value $w_{i,j}$, which represents the weight of the ith term in the jth document, i.e., the importance of the term

in the corpus. Various term weighting schemes are available to compute the weight of a term [3], [15]. Different IR techniques [3], [15], [16], [17] can be used to compute the similarity between two documents.

2.2 Histrace

Histrace creates links between the set of requirements, R and the source code, C, using the software repositories T_i, i.e., in the following, T_1 stands for CVS/SVN commit messages and T_2 for bug reports. Histrace considers the requirements' textual descriptions, CVS/SVN commit messages, bug reports, and classes as separate documents. It uses these sources of information to produce two experts, $R2CT_{1,r_j,t_k}$, which we call $Histrace_{commits}$ in the following for simplicity, and $R2CT_{2,r_j,t_k}$, which we call $Histrace_{bugs}$, which use CVS/SVN commit messages and bug reports, respectively, to create traceability links between R and C through the T_i. Below, we discuss each step of Histrace in detail.

2.2.1 Document Preprocessing

Depending on the input information source (i.e., requirements, source code, CVS/SVN commit messages, or bug reports), we perform specific preprocessing steps to remove irrelevant details from the source, (e.g., CVS/SVN commit number, source code punctuation, or language keywords), split identifiers, and, finally, normalize the resulting text using stop-word removal and stemming.

Requirements and source code: Histrace first processes source code files to extract all the identifiers and comments from each class. Histrace then uses underscore and the CamelCase convention [3] to split identifiers into terms, thus producing a separate document for each class.

Histrace then performs the following steps to normalize requirements and source code documents: 1) converting all upper case letters into lower case and removing punctuation, 2) removing all stop words (such as articles, numbers, and so on), and 3) performing word stemming using the Porter Stemmer [18], bringing back inflected forms to their morphemes.

CVS/SVN commit messages: We use Ibdoos[2] to convert CVS/SVN commit logs into a unified format and put all commit messages into a database for ease of treatment. Ibdoos provides parsers for various formats of commit logs, including CVS, Git, and SVN.

To build $Histrace_{commits}$, Histrace first extracts CVS/SVN commit logs and excludes those that 1) are tagged as "delete" because they concern classes that have been deleted from the system and thus cannot take part in any traceability link, 2) do not concern source code (e.g., if a commit only contains HTML or PNG files), 3) have messages of length shorter than or equal to one English word because such short messages would not have enough semantics to participate in creating $R2CT_{1,r_j,t_k}$.

Histrace then extracts the CVS/SVN commit messages as well as the classes that 1) are still part of the source code and 2) have been part of the commits. Histrace applies the same normalization steps on CVS/SVN commit messages as those for requirements and source code.

2. http://www.ptidej.net/download/ibdoos/.

Following our definitions, T_1 is the set of all remaining CVS/SVN commit messages and, for any $t_k \in T_1$, we have a function δ_{T_1} that returns a subset of the classes $C'_k \subset C = \{c_1, \ldots, c_M\}$ modified in the commits. Histrace uses commit messages and an IR-based technique to compute σ'_{r_i,t_k}. Using δ_{T_1}, Histrace builds $R2CT_{1,r_j,t_k}$.

For example, Histrace could use an IR technique, e.g., VSM, to link the CVS/SVN log $Logs_{1741}$ of *Pooka*, containing `NewMessageInfo.java`, to the requirement r_{21}. This means that Histrace can simply link r_{21} to `NewMessageInfo.java` and add it to the $Histrace_{commits}$ set for *Pooka*.

Bug reports: To build $Histrace_{bugs}$, Histrace extracts all the bug reports from a bug-tracking system. Usually, bug reports do not contain explicit information about the source code files that developers updated to fix a bug. All the details about the updated, deleted, or added source code files are stored in some CVS/SVN repository. Therefore, Histrace must link CVS/SVN commit messages to bug reports before being able to exploit bug reports for traceability.

Histrace uses regular expressions, i.e., a simple text matching approach but with reasonable results, to link CVS/SVN commit messages to bug reports. However, Histrace could also use more complex techniques, such as those in [19], [20]. Consequently, Histrace assumes that developers assigned to each bug a unique ID that is a sequence of digits recognizable via regular expressions. The same ID must be referred to by developers in the CVS/SVN commit messages.

Then, to link CVS/SVN commit messages to bug reports concretely, Histrace performs the following steps: 1) extracts all CVS/SVN commit messages, along with commit status and committed files, 2) extracts all the bug reports, along with time/date and textual descriptions, and 3) links each CVS/SVN commit message and bug reports using regular expressions, e.g.,

$$((b)[ug]\{0,2\}\backslash s * [id]\{0,3\}|id|fix|pr|\#) \\ [\backslash s\# =] * [?([0 - -9]\{4,6\})]?$$

which is the regular expression tuned to the naming and numbering conventions used by the developers of *Rhino*. This expression must be updated to match other naming and numbering conventions, as discussed in Section 3.6.4.

In its last step, Histrace removes false-positive links by imposing the following constraint:

$$fix(e[ds])?|bugs?|problems?|defects?patch".$$

This regular-expression constraint only keeps a CVS/SVN commit if it contains a keyword, i.e., fix(es), fixed, bug(s), problem(s), defect(s), or patch, followed by a number, i.e., if it follows naming conventions for bug numbering usual in open-source development. It thus returns the bug reports linked to CVS/SVN commit messages.

Following our definitions, T_2 is the set of all bug reports and, for any $t_k \in T_2$, we have a function δ_{T_2} that returns a subset of classes $C'_k \subset C = \{c_1, \ldots, c_M\}$ modified to fix a bug. Histrace uses the bug reports and an IR-based technique to compute σ'_{r_i,t_k}. Using δ_{T_1}, Histrace builds $R2CT_{2,r_j,t_k}$ as explain above.

For example, Histrace could use an IR technique to link the bug report bug_{434} to requirement r_{11}. Then, it could link

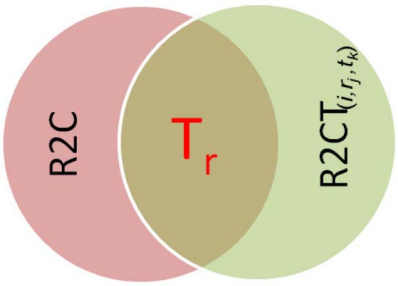

Fig. 2. Overlapping of $R2C$, $R2CT_{i,r_j,t_k}$, and Tr.

the bug report bug_{434} to the CVS/SVN log $Logs_{4912}$ in *SIP* using the appropriate regular expression. $Logs_{4912}$ contains `FirstWizardPage.java` and `DictAccountRegistrationWizard.java`. Thus, Histrace could link r_{11} to `FirstWizardPage.java` and `DictAccountRegistrationWizard.java` to build the $Histrace_{bugs}$ set for *SIP*.

2.3 Trumo

Trumo assumes that different experts $Histrace_{commits}$ (also known as $R2CT_{1,r_j,t_k}$) and $Histrace_{bugs}$ ($R2CT_{2,r_j,t_k}$) know useful information to discard or rerank the traceability links between two documents, e.g., requirements and source code classes. Trumo is thus similar to a web model of the users' trust: The more users buy from a web merchant, the higher the users' trust of this merchant [14].

By the definitions in Section 2.1, in (1), r_j is a requirement with $r_j \in R$; c_s is a class with $c_s \in \delta_{T_i}(t_k)$ because we use the sets $T_i \in \mathcal{T}$ to build a set of trustable links Tr; σ'_i is the similarity score between the requirement r_j and some class t_k such that $t_k \in T_i$ and $\alpha(R2CT_{i,r_j,t_k})$ returns a pair (r_j, t_k). With (1), Trumo uses the set of candidate links $l_{rc} = (r_j, c_s, \sigma_i(r_j, c_s))$ with $j \in [1, \ldots, N]$ and $s \in [1, \ldots, M]$ and the sets of candidate links $l_{rt} = (r_j, c_s, \sigma'_i(r_j, t_k))$ with $j \in [1, \ldots, N]$ and $k \in [1, \ldots, N_i]$ generated from each expert T_i and, for each requirement $r_j \in R$:

$$\begin{aligned} Tr = \{&(r_j, c_s, \sigma'_i(r_j, t_k))| \\ &\exists\, t_k \in T_i : (r_j, c_s) \in \alpha(R2CT_{i,r_j,t_k}) \\ &\&\ (r_j, c_s) \in \alpha(R2C)\}. \end{aligned} \tag{1}$$

Fig. 2 shows the Venn diagram of the $R2C$, $R2CT_{i,r_j,t_k}$, and Tr sets. The last constraint $(r_j, c_s) \in \alpha(R2C)$ imposes that a link be present in the baseline set $R2C$ and in any of the $R2CT_{i,r_j,t_k}$ sets. If a link does not satisfy this constraint, then Trumo discards it. Then, Trumo reranks the similarity of the remaining links in Tr as follows: Let $TC_i(r_j, c_s)$ be the restriction of Tr on (r_j, c_s) for the source T_i, i.e., the set $\{(r_j, c_s, \sigma'_i(r_j, t_k)) \in Tr\}$, then Trumo assigns to the links in $TC_i(r_j, c_s)$ a new similarity $\sigma^*_i(r_j, c_s)$ computed as

$$\sigma^*_i(r_j, c_s) = \frac{\sigma(r_j, c_s) + \sum_{l \in TC_i(r_j,c_s)} \phi(l)}{1 + |TC_i(r_j, c_s)|}, \tag{2}$$

where $\sigma(r_j, c_s)$ is the similarity between the requirement r_j and the class c_s as computed in $R2C$ and $\phi(l)$ is the similarity of the documents linked by the link l of $TC_i(r_j, c_s)$, i.e., derived from t_k, which means $\sigma'_i(r_j, t_k)$. Finally, $|TC_i(r_j, c_s)|$ is the number of elements in $TC_i(r_j, c_s)$. The higher the evidence (i.e., $\sum_{l \in TC_i(r_j,c_s)} \phi(l)$) provided by links in $TC_i(r_j, c_s)$, the higher the new similarity $\sigma^*_i(r_j, c_s)$; on the contrary, little evidence decreases $\sigma^*_i(r_j, c_s)$.

Finally, Trumo assigns weight to each expert and combines their similarity values to assign a new similarity value to each link in Tr using the ψ function:

$$\begin{aligned} \psi_{r_j,c_s}(Tr) = &\left[\sum_{i=1}^{P} \lambda_i(r_j, c_s)\sigma^*_i(r_j, c_s)\right] \\ &+ \lambda_{P+1}(r_j, c_s)\frac{|Tr(r_j, c_s)|}{\max_{n,m}|Tr(r_N, C_M)|}, \end{aligned} \tag{3}$$

where $Tr(r_j, c_s)$ is the subset of Tr restricted to the links between r_j and c_s, $\lambda_i(r_j, c_s) \in [0, 1]$ and $\lambda_1(r_j, c_s) + \lambda_2(r_j, c_s) + \cdots + \lambda_{P+1}(r_j, c_s) = 1$; recall that P is the total number of experts. With the ψ function, the more often a pair (r_j, c_s) exists in Tr, the more we can trust this link (if such a link is also present in $R2C$).

2.4 DynWing

To automatically decide the weights $\lambda_i(r_j, c_s)$ for each expert, we apply a dynamic weighting technique. Existing techniques [2], [8] to define weights use static weights for all the experts. Thus, they require oracles to decide a "good" weight or range of weights. However, with real, legacy systems, no such oracle exists, i.e., no a priori-known set of traceability links exists. Moreover, using the same static weight may not be beneficial for all the recovered links.

Therefore, we consider each link recovered by a baseline IR technique and by the different experts as an independent link and dynamically assign weights to baseline links and each expert. Choosing the right weight per link is a problem that we formulate as a maximization problem. Basically, we have different experts, i.e., CVS/SVN commits, bug reports, and others, to trust a link. Each expert has its own trust into the link. By maximizing the similarity value $\psi_{r_j,c_s}(Tr)$ (and hence determining the optimal $\lambda_i(r_j, c_s)$ values), DynWing automatically identifies the experts that are most trustworthy (highest $\lambda_i(r_j, c_s)$ values) and those that are less trustworthy (lowest $\lambda_i(r_j, c_s)$ values):

$$\max_{\lambda_1(r_j,c_s),\ldots,\lambda_{P+1}(r_j,c_s)} \{\psi_{r_j,c_s}(Tr)\}, \tag{4}$$

with the following constraints:

$$\begin{aligned} &0 \le \lambda_i(r_j, c_s) \le 1, i = 1, \cdots, P+1 \\ &\lambda_1(r_j, c_s) + \lambda_2(r_j, c_s) + \cdots + \lambda_{P+1}(r_j, c_s) = 1 \\ &\lambda_{k_1}(r_j, c_s) \ge \lambda_{k_2}(r_j, c_s) \ge \cdots \ge \lambda_{k_{P+1}}(r_j, c_s). \end{aligned}$$

Given the three previous constraints, it is possible that DynWing assigns $\lambda_i(r_j, c_s) = 1$ to a single expert i. To avoid such an assignment, a developer can define her global trust into the experts. For example, CVS/SVN commit messages may be considered by the developer more trustworthy than bug reports. Therefore, the developer may further constrain (4) by imposing

$$\lambda_{commits}(r_j, c_s) \ge \lambda_{bugs}(r_j, c_s) > 0.$$

3 Empirical Evaluation

We now report on an empirical evaluation of Trustrace with four systems to assess its accuracy in terms of precision and recall. We use two state-of-the-art IR techniques, i.e., JSM and VSM, for evaluation purposes. We use the names $Trustrace_{VSM}$ and $Trustrace_{JSM}$ to denote the IR techniques that Trustrace uses. We also compare the DynWing weighting technique of Trustrace with the PCA weighting technique [9].

We implement Trustrace and its three novel techniques in FacTrace[3] (artiFACT TRACEability). FacTrace provides several modules with activities ranging from traceability recovery to the manual verification of traceability links.

3.1 Goal

The *goal* of our empirical evaluation is to study the accuracy of Trustrace when recovering traceability links against that of a single IR technique, JSM and VSM, using requirements, source code, CVS/SVN commits, and/or bug reports as experts. The *quality focus* is the accuracy of Trustrace in terms of precision and recall [21]. It is also the improvement brought by the dynamic weighting technique, DynWing, with respect to a PCA-based technique in terms of F_1 score. The *perspective* is that of practitioners interested in recovering traceability links with greater precision and recall values than that of currently available traceability recovery IR-based techniques. It is also that of researchers interested in understanding whether or not we can support our conjectures.

3.2 Research Questions

Our research questions are:

- **RQ1:** How does the accuracy of the traceability links recovered by Trustrace compare with that of approaches based on JSM and VSM alone?
- **RQ2:** How does the accuracy of the traceability links recovered using DynWing compare to that using PCA?

To answer **RQ1**, we assess the accuracy of JSM, Trustrace, and VSM in terms of precision and recall by applying them on four systems seeking to reject the four null hypotheses:

- $\mathbf{H_{01}}$**:** There is no difference in the precision of the recovered traceability links when using Trustrace or VSM.
- $\mathbf{H_{02}}$**:** There is no difference in the precision of the recovered traceability links when using Trustrace or JSM.
- $\mathbf{H_{03}}$**:** There is no difference in the recall of the recovered traceability links when using Trustrace or VSM.
- $\mathbf{H_{04}}$**:** There is no difference in the recall of the recovered traceability links when using Trustrace or JSM.

To answer **RQ2**, we use the DynWing and PCA weighting techniques and compute the F_1 score of Trustrace to analyze which weighting technique provides better results. We try to reject the two null hypotheses:

- $\mathbf{H_{05}}$**:** DynWing does not provide automatically better $\lambda_i(r_j, c_s)$ than a PCA-based weighting technique for $Trustrace_{VSM}$.
- $\mathbf{H_{06}}$**:** DynWing does not provide automatically better $\lambda_i(r_j, c_s)$ than a PCA-based weighting technique for $Trustrace_{JSM}$.

3.3 Variables

We use precision, recall, and F_1 score as dependent variables. All measures have values in the range $[0, 1]$:

$$Precision = \frac{|\{relevant\ links\} \cap \{retrieved\ links\}|}{|\{retrieved\ links\}|},$$

$$Recall = \frac{|\{relevant\ links\} \cap \{retrieved\ links\}|}{|\{relevant\ links\}|}.$$

The F_1 score is the harmonic mean of precision and recall, which is computed as

$$F_1 = \frac{2}{\frac{1}{R} + \frac{1}{P}},$$

where R is the recall, P is the precision, and F_1 is the harmonic mean of R and P. We use the F_1 score to compare DynWing and PCA because F_1 equally weighs precision and recall. Thus, it shows which weighting technique provides the best precision and recall values.

We use the approaches, either single IR technique, i.e., JSM and VSM, or Trustrace, as independent variables. The independent variable corresponding to Trustrace also includes varying values of λ_i using the DynWing and PCA-based weighting techniques.

3.4 Objects

We select the four open-source systems, *jEdit 4.3*, *Pooka v2.0*, *Rhino 1.6*, and *SIP Communicator 1.0-draft*, because they satisfy our key criteria. First, we choose open-source systems so that other researchers can replicate our evaluation. Second, all systems are small enough so that we could recover and validate their traceability links manually in previous work [2] while still being a real-world system.

jEdit[4] is a text editor for developers written in Java. jEdit includes a syntax highlighter that supports over 130 file formats. It also allows developers to add additional file formats using XML files. It supports UTF-8 and many other encoding techniques. It has extensive code folding and text folding capabilities, as well as text wrapping that takes indentation into account. It is highly customizable and can be extended with macros written in BeanShell, Jython, JavaScript, and some other scripting languages. jEdit version 4.3 has 483 classes, measures 109 KLOC, and implements 34 requirements.

Pooka[5] is an e-mail client written in Java using the JavaMail API. It supports reading e-mails through the IMAP and POP3 protocols. Outgoing emails are sent using SMTP. It supports folder search, filters, and context-sensitive colors.

3. http://www.factrace.net.

4. http://www.jedit.org.

5. http://www.suberic.net/pooka/.

Pooka version 2.0 has 298 classes, weighs 244 KLOC, and implements 90 requirements.

Rhino[6] is an open-source JavaScript engine entirely developed in Java. *Rhino* converts JavaScript scripts into objects before interpreting them and can also compile them. It is intended to be used in server-side systems but can also be used as a debugger by making use of the *Rhino* shell. *Rhino* version 1.6 has 138 classes, measures 32 KLOC, and implements 268 requirements.

SIP Communicator[7] is an audio/video Internet phone and instant messenger that supports some of the most popular instant messaging and telephony protocols, such as AIM/ICQ, Bonjour, IRC, Jabber, MSN, RSS, SIP, Yahoo! Messenger. *SIP* Communicator version *1.0-draft* has 1,771 classes, measures 486 KLOC, and implements 82 requirements.

3.5 Oracles

We use four oracles, i.e., $\text{Oracle}_{\text{jEdit}}$, $\text{Oracle}_{\text{Pooka}}$, $\text{Oracle}_{\text{Rhino}}$, and $\text{Oracle}_{\text{SIP}}$, to compute the precision and recall values of JSM, Trustrace, and VSM when applied on our four object systems.

For *Pooka* and *SIP*, the first author and another PhD student created traceability links between the requirements of the two systems and their source code classes. They read the requirements and manually looked for classes in the source code that implement these requirements. They used Eclipse to search for the source code and stored all manually built links in FacTrace database. The second and third authors used the FacTrace voting system to accept or reject all manually built links. At no point of the process did we use any automated technique or software repository to create the oracles. This process is the same as used in our previous work [2], [22].

For *jEdit* and *Rhino*, we use the same oracles as previous researchers (Eaddy et al. [23] and Dit et al. [24]), which helps mitigating threats to the validity of our evaluation.

3.6 Preprocessing

We now detail how we gather and prepare the input data necessary to perform our empirical evaluation of Trustrace.

3.6.1 Requirements

jEdit contains 34 requirements. These requirements were manually identified and extracted from the *jEdit* source code repository [24]. In previous work [2], we used PREREQUIR [7] to recover requirements for *Pooka* and *SIP*. We recovered 90 and 82 requirements for both systems, respectively. *Rhino* contains 268 requirements that we extracted from the related ECMAScript specifications by considering each ECMAScript section as a requirement.

3.6.2 Source Code

We downloaded the source code of *jEdit v4.3*, *Pooka v2.0*, *Rhino v1.6*, and *SIP v1.0-draft* from their respective CVS/SVN repositories. We made sure that we had the correct files for each system before building traceability links by setting up the appropriate environments and by downloading the appropriate libraries. We thus could compile and run all the systems.

6. http://www.mozilla.org/rhino/.
7. http://www.jitsi.org.

```
<logentry revision="1741">
    <author>akp</author>
    <date>2008-10-18T16:14:08.529138Z</date>
    <paths>
        <path kind="" action="M">/trunk/pooka/todo</path>
        <path kind="" action="M">/trunk/pooka/src/net/
            suberic/pooka/gui/NewMessageDisplayPanel.java
        </path>
        <path kind="" action="M">/trunk/pooka/src/net/
            suberic/pooka/NewMessageInfo.java
        </path>
    </paths>
    <msg>
        fixing a bug where the cc: on reply-all doesn't
        get cleared out if the cc field is set to empty.
    </msg>
</logentry>
```

Fig. 3. Excerpt of *Pooka* SVN Log.

3.6.3 CVS/SVN Commit Messages

Fig. 3 shows an excerpt of a commit of *Pooka*. There are 3,762, 1,743, 3,261, and 8,079 SVN commits for *jEdit*, *Pooka*, *Rhino*, and *SIP*, respectively. We performed the data preprocessing steps described in Section 2.2.1 on all SVN commits with the help of FacTrace.

After performing the preprocessing steps, we obtained 2,911, 1,393, 2,508, and 5,188 SVN commits for *jEdit*, *Pooka*, *Rhino*, and *SIP*, respectively. There were many SVN commits that did not concern source code files. Also, some commit messages contained both source code files and other files. For example, revision 1604 in *Pooka* points only to HTML files except for one Java file, `FolderInternalFrame.java`. Therefore, we only kept the Java file and removed any reference to the HTML files. We stored all filtered SVN commit messages and related files in a FacTrace database.

3.6.4 Bug Reports

We cannot use *jEdit* [24] and *Pooka* bug reports because the first system does not have a publicly available bug repository and the second one has too few recorded bugs (16). *Rhino* is part of the Mozilla browser and its bug reports are available via the Mozilla Bugzilla bug tracker. We extracted all 770 bugs reported against *Rhino* and used Histrace to link them with the CVS repository as described in Section 2.2. Histrace automatically linked 457 of the bug reports to their respective commits. In the case of *SIP*, we downloaded 413 bug reports. *SIP* developers did not follow any rule while fixing bugs to link bug reports and commits. Hence, there was no bug ID in the commit messages. However, developers referenced SVN revision numbers in the bug reports' comments, e.g., bug ID 237 contains the revision ID *r*4550. We tuned the regular expression of Histrace to find the revision IDs in the descriptions of the SIP bug reports. Histrace thus extracted all the bug IDs and linked them to SVN commits. Overall, Histrace automatically linked 169 bugs reported against *SIP* to their respective commits.

3.6.5 Last Preprocessing Step

We automatically extracted all the identifiers from the *jEdit*, *Pooka*, *Rhino*, and *SIP* requirements, source code, filtered CSV/SVN commit messages, and filtered bug reports, using FacTrace. The output of this step is four corpora that we use for creating traceability links, as explained in Sections 2.2 and 2.3.

3.7 IR Techniques

To build the sets of traceability links, we use the VSM (from the algebraic family of techniques) and JSM (from the probabilistic family of techniques) techniques. Abadi et al. [15] performed experiments using different IR techniques to recover traceability links. Their results show that the Vector Space Model and the Jensen-Shannon model outperform other IR techniques. In addition, these two techniques do not depend on any parameter. Thus, we use both JSM and VSM to recover traceability links and compare their results in isolation with those of Trustrace. These techniques both essentially use term-by-document matrices. Consequently, we choose the well-known TF/IDF measure [3], [25], [26], [27] for VSM and the normalized term frequency measure [15] for JSM. These two measures and IR techniques are state-of-the-art IR techniques. In the following, we explain both techniques in details.

3.7.1 Vector Space Model

Many traceability recovery techniques use VSM as the base algorithm [3], [28], [29]. In VSM, documents are represented as vector in the space of all the terms. Different term weighting schemes can be used to construct these vectors. We use the standard TF/IDF weighting scheme [28]: A document is a vector of TF/IDF weights. TF is often called the local weight. The most frequent terms will have more weight in TF, but this by itself does not mean that they are important terms. The inverse document frequency, IDF, of a term is calculated to measure the global weight of a terms and is computed as $IDF = \log_2(\frac{|D|}{|d:t_i \in d|})$. Then, TF/IDF is defined as

$$(TF/IDF)_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \times \log_2\left(\frac{|D|}{|d : t_i \in d|}\right),$$

where $n_{i,j}$ are the occurrences of a term t_i in document d_j, $\sum_k n_{k,j}$ is the sum of the occurrences of all the terms in document d_j, $|D|$ is the total number of documents d in the corpus, and $|d : t_i \in d|$ is the number of documents in which the term t_i appears.

Once documents are represented as vectors of terms in a VSM, traceability links are created between every two documents with their own similarity value depending on each pair of documents, e.g., a requirement and a class. The similarity between two documents is measured by the positive cosine of the angle between their corresponding vectors (because the similarity between two documents cannot be negative). The ranked list of recovered links and a similarity threshold are used to divide links into a set of candidate links to be manually verified [3].

3.7.2 Jensen-Shannon Model

JSM is an IR technique proposed by Abadi et al. [15]. It is driven by a probabilistic approach and hypothesis testing technique. JSM represents each document through a probability distribution, i.e., a normalized term-by-document matrix. The probability distribution of a document is

$$p = \frac{n(w,d)}{T_d},$$

where $n(w,d)$ is the number of times a word appears in a document d and T_d is the total number of words appearing in d. The empirical distribution can be modified to take into account the term's global weight, e.g., IDF. After considering the global weight, each document distribution must be normalized. Once the documents are represented as probability distribution, JSM computes the distance between two documents' probability distribution and returns a ranked list of links. JSM ranks target documents via the "distance" of their probability distributions to that of the source documents:

$$JSM(q,d) = H\Big(\frac{p_q + p_d}{2}\Big) - \frac{H(p_q) + H(p_d)}{2},$$
$$H(p) = \sum h(p(w)),$$
$$h(x) = -x \log x,$$

where $H(p)$ is the entropy of the probability distribution p, and p_q and p_d are the probability distributions of the two documents (a "query" and a "document"), respectively. By definition, $h(0) \equiv 0$. We compute the similarity between two documents using $1 - JSM(q,d)$. The similarity values are in $[0,1]$.

3.8 Building Sets of Traceability Links

First, we use JSM and VSM to create traceability links, i.e., $R2C_{JSM}$ and $R2C_{VSM}$, between requirements and source code. Second, we apply $Histrace_{commits}$, as described in Section 2.2, to process *jEdit*, *Pooka*, *Rhino*, and *SIP* CVS/SVN commit messages (T_1), and requirements to create the traceability link set $R2CT_{1,r_j,t_k}$. We process *SIP* and *Rhino* bug reports (T_2) to create the traceability link sets $R2CT_{2,r_j,t_k}$ using $Histrace_{bug}$.

For example, we trace *Pooka* requirement "*it should have spam filter option*" to the SVN commit message "*adding prelim support for spam filters.*" SVN commit revision number 1133. Then, we recover all the source code classes related to this commit, i.e., `SpamSearchTerm.java` and `SpamFilter.java`. Finally, we create a direct traceability link between the files `SpamSearchTerm.java` and `SpamFilter.java` to the requirement "*it should have spam filter option*."

Third, we apply Trumo as described in Section 2.3 using traceability links recovered with JSM and VSM. We thus compute two sets $R2C$, $R2C_{JSM}$ and $R2C_{VSM}$, one with each IR technique. We then apply the Trumo equation via CVS/SVN commit messages and-or bug reports to discard/rerank links by computing new similarity values using (3). These values help to answer **RQ1** and to attempt rejecting our null hypotheses.

3.9 Weighting Technique

We use DynWing as presented in Section 2.4 and compare with PCA [9]. We use these two weighting techniques on the recovered traceability links of *jEdit*, *Pooka*, *Rhino*, and *SIP* to answer **RQ2** and attempt rejecting our null hypotheses.

3.9.1 Principal Component Analysis

PCA is a mathematical analysis used in a recent case study on traceability [9] to combine different IR techniques and to define weights for each technique. PCA defines a single static weight for each expert.

PCA uses an orthogonal transformation to convert a set of correlated variables into a set of values of uncorrelated

variables, called principal components. This transformation is defined in such a way that the proportion of variance for each principal component (PC) is $PC_1 > PC_2 > \ldots PC_n$. The number of PCs is less than or equal to the number of original variables.

Gethers et al. [9] compute the weights for different IR techniques using PCA as follows: 1) They use the value of the proportion of variance based on the PC with the highest correlation and 2) they normalize values to obtain weights for each technique. For example, if PC_1's proportion of variance is 71.29 percent and CVS/SVN has a correlation of 0.99, we would assign 71.29 percent to CVS/SVN. Likewise, if a bug report has a higher correlation than PC_1 when compared to other PCs, it would also receive a value of 71.29 percent. After assigning values of proportions of variances to all experts, we normalize the values so their sum equals to one.

3.10 Experimental Settings

We must choose only one setting before applying Trustrace: our global trust of the experts. This global trust helps DynWing to assign weights to each link according to our a priori trust in each expert as mentioned in Section 2.4. In our empirical evaluation, we define the global trust as

$$\lambda_{commits} \geq \lambda_{bugs} \geq \lambda_{p+1} > 0.$$

This global weight ensures that DynWing gives more weight to the $Histrace_{commits}$ expert than to the $Histrace_{bugs}$ expert. We choose to favor $Histrace_{commits}$ based on the quality of the semantic information contained in the commit messages, the bug reports, and the source code classes, as further discussed in Section 5.1.

3.11 Analysis Method

We use $\text{Oracle}_{\text{jEdit}}$, $\text{Oracle}_{\text{Pooka}}$, $\text{Oracle}_{\text{Rhino}}$, and $\text{Oracle}_{\text{SIP}}$ to compute the precision and recall values of the links recovered using JSM, Trustrace, and VSM. JSM and VSM assign a similarity value to each and every traceability link whereas Trustrace uses its model, defined in Section 2.3, to reevaluate the similarity values of the links provided by a baseline technique.

To answer **RQ1**, we perform several experiments with different threshold values on the recovered links to perform statistical tests on precision and recall values. We use a threshold t to prune the set of traceability links, keeping only links whose similarities values are greater than or equal to $t \in [0, 1]$. We use different values of t from 0.01 to 1 per steps of 0.01 to obtain different sets of traceability links with varying precision and recall values, for all approaches. We then perform paired-statistical tests to measure the improvements brought by Trustrace. In the paired-statistical tests, two chosen approaches must have the same number of data points. Therefore, we keep the same threshold t values for both approaches. For example, if VSM discards all traceability links whose textual similarity values are below the 0.7 threshold, then we also use the same 0.7 threshold for Trustrace.

Then, we assess whether the differences in precision and recall values, in function of t, are statistically significant between the JSM, Trustrace, and VSM approaches. To select an appropriate statistical test, we use the Shapiro-Wilk test to analyze the distributions of our data points. We observe that these distributions do not follow a normal distribution. Thus, we use a nonparametric test, i.e., Mann-Whitney test, to test our null hypotheses to answer **RQ1**.

An improvement might be statistically significant, but it is also important to estimate the magnitude of the difference between the accuracy levels achieved with a single IR technique and Trustrace. We use a nonparametric effect size measure for ordinal data, i.e., Cliff's d [27], to compute the magnitude of the effect of Trustrace on precision and recall as follows:

$$d = \left| \frac{\#(x_1 > x_2) - \#(x_1 < x_2)}{n_1 n_2} \right|,$$

where x_1 and x_2 are precision or recall values with JSM, Trustrace, and VSM, and n_1 and n_2 are the sizes of the sample groups. The effect size is considered small for $0.15 \leq d < 0.33$, medium for $0.33 \leq d < 0.47$, and large for $d \geq 0.47$.

To answer **RQ2**, we use PCA and DynWing to assign weights to the traceability links recovered using Trustrace. We use different values of t from 0.01 to 1 per steps of 0.01 to obtain different sets of traceability links with varying F_1 scores. We use the Mann-Whitney to reject the null hypotheses $\mathbf{H_{05}}$ and $\mathbf{H_{06}}$.

4 Results

We now present the results and answers to our two research questions.

4.1 RQ1: How Does the Accuracy of the Traceability Links Recovered by Trustrace Compare with that of Approaches Based on JSM and VSM Alone?

Fig. 4 shows the precision and recall graphs of JSM, Trustrace, and VSM. Trustrace provides better precision and recall values than the two IR techniques by themselves. Table 1 shows the average precision and recall values calculated by comparing the differences between the JSM, Trustrace, and VSM approaches. Trustrace with DynWing has better precision and recall than the other weighting techniques. The recall value for *Pooka* improves on average but without statistical significance when compared to VSM results. In the case of *SIP* with only the $Histrace_{bugs}$ expert, recall values decrease with statistical significance when compared to VSM values, as discussed in Section 5.1. There is no statistically significant decrease in recall when compared to JSM results. We explored the reason for the recall values to decrease in the case of *SIP* with $Histrace_{bugs}$. We found that only 4 percent of *SIP* SVN commits are linked to bug reports. Therefore, we did not find much evidence for many links and this lack of evidence yielded many links from the baseline set to be removed, following our constraints in (1), and, consequently, a lower recall.

We performed the statistical tests described in Section 3.11 to verify whether or not the average improvements in precision and recall are statistical significant. We have statistically significant evidence to reject $\mathbf{H_{01}}$ and $\mathbf{H_{02}}$. Table 1 shows that the p-values for the precision values are below the standard significant value, $\alpha = 0.05$. The reported figures show that, for most values of precision and recall,

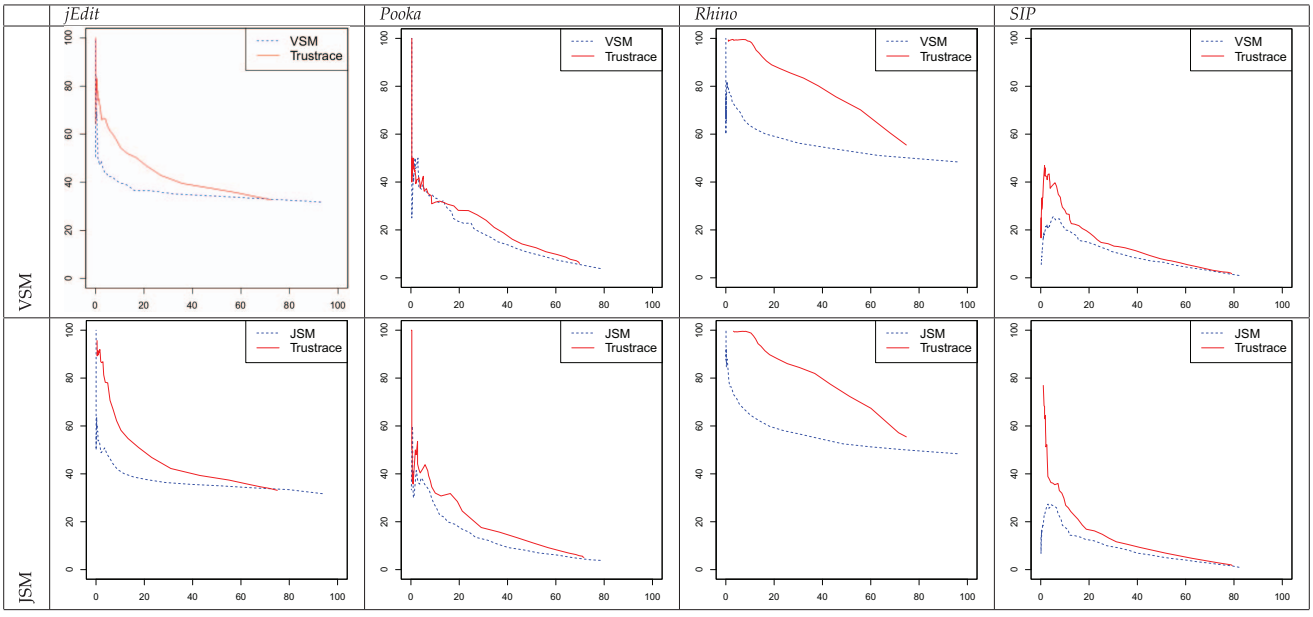

Fig. 4. Precision and recall values of JSM, Trustrace, and VSM, with the threshold t varying from 0.01 to 1 by step of 0.01. The X-axis shows recall and the Y-axis shows precision.

we can reject $\mathbf{H_{03}}$ and $\mathbf{H_{04}}$ in all but three cases (in bold in Table 1). Thus, we cannot claim to always reject $\mathbf{H_{03}}$ and $\mathbf{H_{04}}$: In two cases for VSM and one for JSM, recall values do not improve.

We use Cliff's d as introduced in Section 3.11 to measure the effect of Trustrace over single IR techniques. Table 1 shows that Trustrace has a large effect on the improvements in precision and recall values in 66 percent, medium in 19 percent, small in 9 percent of the improvements. Only in the case of *SIP*, i.e., 6 percent, with only $Histrace_{bugs}$ did Trustrace recall values decrease with one large and one small effect size. Overall, the obtained effect size values indicate a practical improvement with Trustrace.

We answer **RQ1**: How does the accuracy of the traceability links recovered by Trustrace compare with that of approaches based on JSM and VSM alone? as follows: Trustrace helps to recover more correct links than IR techniques alone. When two experts are available, Trustrace is always better. In only one case and with just a single expert due to a lack of external source of information, did recall go down.

4.2 RQ2: How Does the Accuracy of the Traceability Links Recovered Using DynWing Compare to that Using PCA?

When comparing DynWing with a PCA-based weighting technique, Fig. 5 shows that DynWing provides better results than PCA. PCA tends to provide higher precision than recall in some cases, whereas DynWing tends to find a better balance between precision and recall in all cases.

We performed a Mann-Whitney test to analyze if DynWing statistically provides better F_1 scores or not. Table 2 shows that p-values are below the standard significant value $\alpha = 0.05$ for $Trustrace_{VSM}$. Thus, we reject $\mathbf{H_{05}}$. In the case of

TABLE 1
Precision and Recall Values for *jEdit*, *Pooka*, *Rhino*, and *SIP*, Mann-Whitney Test Results, and Cliff's d Results

	Precision				Recall			
	VSM	$Trustrace_{VSM}$	p-value	Effect Size	**VSM**	$Trustrace_{VSM}$	p-value	Effect Size
jEdit	59.55	69.14	<0.01	0.59	5.81	8.31	<0.01	0.67
Pooka	42.28	52.46	<0.01	0.37	11.14	12.52	**0.99**	0.24
Rhino ($Histrace_{commits}$ *only*)	71.79	92.76	<0.01	0.92	6.82	9.52	<0.01	0.92
Rhino ($Histrace_{bugs}$ *only*)	71.79	93.73	<0.01	0.92	6.82	9.20	<0.01	0.92
Rhino	71.79	94.49	<0.01	0.84	6.82	12.33	<0.01	0.96
SIP, ($Histrace_{commits}$ *only*)	15.84	25.97	<0.01	0.61	15.61	15.79	<0.04	0.55
SIP, ($Histrace_{bugs}$ *only*)	15.84	42.97	<0.01	0.37	15.61	**11.07**	**<0.01**	**0.63**
SIP	15.84	24.28	<0.01	0.34	15.61	21.60	<0.01	0.48

	JSM	$Trustrace_{JSM}$	p-value	Effect Size	**JSM**	$Trustrace_{JSM}$	p-value	Effect Size
jEdit	52.82	71.12	<0.01	0.82	13.62	15.91	<0.01	0.88
Pooka	33.11	45.48	<0.01	0.47	13.33	16.45	<0.01	0.19
Rhino ($Histrace_{commits}$ *only*)	77.37	87.16	<0.01	0.15	15.56	16.18	<0.01	0.77
Rhino ($Histrace_{bugs}$ *only*)	77.37	90.88	<0.01	0.47	15.56	17.33	<0.01	0.59
Rhino	77.37	91.79	<0.01	0.47	15.56	18.26	<0.01	0.59
SIP ($Histrace_{commits}$ *only*)	15.83	21.29	<0.01	0.46	19.34	21.44	<0.01	0.60
SIP ($Histrace_{bugs}$ *only*)	15.83	37.94	<0.01	0.33	19.34	**14.24**	**<0.56**	**0.25**
SIP	15.83	27.67	<0.01	0.29	19.34	27.00	<0.01	0.92

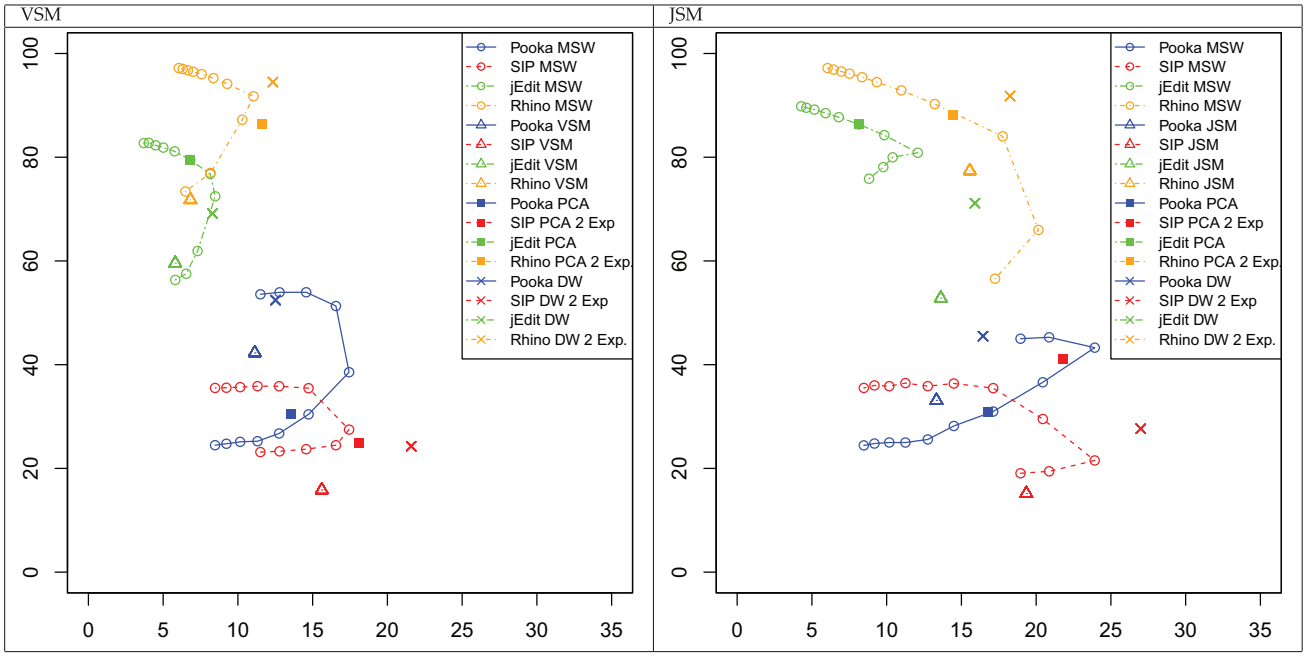

Fig. 5. Precision and recall values, with the threshold t varying from 0.01 to 1 by steps of 0.01. The X-axis shows precision values and Y-axis recall values. DW represents the DynWing results.

JSM, for *SIP*, using two experts does not yield any difference between DynWing and PCA-based weighting. Thus, we cannot reject $\mathbf{H_{06}}$ because in one case DynWing and PCA-based weighting provide the same results.

Thus, we answer **RQ2:** How does the accuracy of the traceability links recovered using DynWing compare to that using PCA? as follows: DynWing provides better weights for different experts than a PCA-based weighting technique. However, it is possible that in some cases PCA-based weighting provides the same (but not better) results as DynWing.

5 Discussion

We now provide qualitative analyses of our results and discuss observations from our empirical evaluation of Trustrace. We also return to our three conjectures.

5.1 Data Set Quality Analysis

Fig. 4 shows that Trustrace has a better accuracy, on average, when compared to JSM and VSM for *jEdit* and *Rhino* but less for *Pooka* and *SIP*. We explain the differences in improvements by the many other factors that can impact the accuracy of traceability-recovery approaches, as discussed elsewhere [10]. One of these factors is quality of source code identifiers. If there is a low similarity between the identifiers used by requirements and source code, then no matter how good an IR-based technique is, it would not yield results with high precision and recall values.

To analyze the quality of the identifiers in our datasets and measure their similarity values, we use our previous approach, Coparvo [22], which helps to identify poor semantic areas in source code. We want to compute the similarity value between the set of requirements R, all merged into a single document $R_{all} = \bigcup_j r_j$, and the set of classes C, all merged into $C_{all} = \bigcup_j c_j$. We build the normalized term-by-document matrix to avoid any effect from the document lengths. Then, we use JSM and VSM to compute the similarity between $\bigcup_j r_j$ and $\bigcup_j c_j$. The similarity between these sets shows how close two documents are in terms of semantics.

Fig. 6 indeed shows that in the case of *Pooka* and *SIP* both JSM and VSM return low similarity values. Low similarity values among different documents would result in low

TABLE 2
F_1 Values for *jEdit*, *Pooka*, *Rhino*, and *SIP*, and Mann-Whitney Test Results

	$Trustrace_{VSM}$								
	$Histrace_{commits}$			$Histrace_{bugs}$			$Histrace_{commits+bugs}$		
	DynWing F_1	PCA F_1	p-value	DynWing F_1	PCA F_1	p-value	DynWing F_1	PCA F_1	p-value
jEdit	8.56	7.50	<0.01	–	–	–	–	–	–
Pooka	10.52	7.42	<0.01	–	–	–	–	–	–
Rhino	12.18	11.61	<0.01	12.01	8.29	<0.01	36.91	10.55	<0.01
SIP	9.07	6.78	<0.01	6.57	6.51	<0.01	12.64	7.67	0.027
	$Trustrace_{JSM}$								
jEdit	9.60	8.49	<0.01	–	–	–	–	–	–
Pooka	8.02	6.60	<0.01	–	–	–	–	–	–
Rhino	13.21	12.33	<0.01	13.16	12.71	<0.01	40.72	15.18	<0.01
SIP	9.07	6.78	<0.01	8.11	6.15	0.02	11.39	7.84	0.127

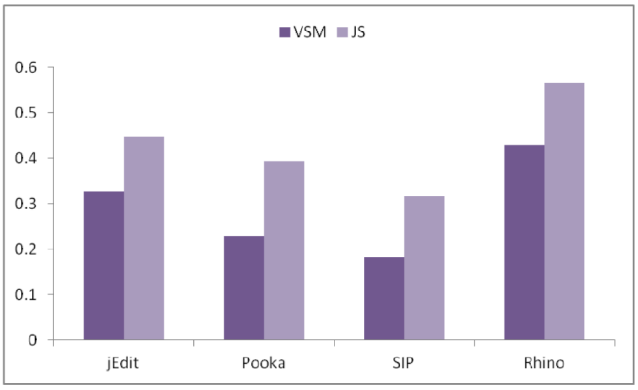

Fig. 6. Similarity between merged requirements and source code documents.

precision and recall values for the traceability links [10]. Fig. 6 shows that JSM provides better similarity values between documents than VSM. Fig. 4 shows that JSM tends to provide better precision and recall values than VSM.

A second factor from Ali et al. [10] is the knowledge of the developers who wrote the requirements. In our empirical evaluation, we recovered requirements for *Pooka* and *SIP* using Prereqir [7]. All the requirements were written by subjects who use e-mail clients and instant messengers on a regular basis, not by developers of such systems. Therefore, the subjects used nontechnical terms in their requirements, which have little semantic similarity with the source code.

jEdit is an open-source Java text editor targeting the Java programming language. Although some *jEdit* users must be Java developers, we could not find any indication that all new feature requests were written by Java developers. Moreover, it is likely that jEdit users are not all Java developers. Users can ask for new features or report bugs in its online bug-tracking system. If a new requested feature is considered important by the community, then it is included in a following version of *jEdit*. In this paper, we considered such features requested by users as genuine requirements. Yet, users used technical terms to define requirements which result in higher similarity values with the source code than *Pooka* and *SIP*.

In the case of *Rhino*, we used the ECMAScript specifications as requirements. ECMAScript specifications are detailed and written by technical writers. Thus, the similarity values between *Rhino* requirements and source code is higher than those of the other three datasets. We also observe that CVS/SVN commit messages are not always informative. Developers often did not provide relevant messages for the CVS/SVN commits or only some very generic message, e.g., "changed," "updated files," and/or "fixed bug." Yet, the CVS commit messages of *Rhino* were better than those of the other three datasets.

Thus, we can say that high semantic similarity between requirements, source code, CVS/SVN commit messages, and bug reports affect the results of any requirements traceability approach. However, Table 1 shows that even with low similarity values between documents, Trustrace improves the precision and recall values of the recovered links.

5.2 DynWing versus MSW versus PCA

Manually tuning the weights used to combine the opinions of experts allows favoring precision over recall or vice versa. However, this manual tuning requires a labeled corpus, i.e., an oracle, that is almost never available. Consequently, previous approaches [2], [8] used multiple-static weights (MSW) to define a range of weights to tune their approaches. We analyzed the links recovered when using MSW to compare DynWing with carefully manually tuned weights.

We use MSW in Trumo, then compute the precision and recall of the obtained links in comparison to our four oracles, $\text{Oracle}_{\text{jEdit}}$, Oracle_{Pooka}, $\text{Oracle}_{\text{Rhino}}$, and Oracle_{SIP}, to find the optimal λ values for (3) in the Trustrace model in Section 2.3 for *jEdit*, *Pooka*, *Rhino*, and *SIP*, respectively. We use different λ values to assess which λ value provides better results. We use $\lambda \in [0, 1]$ values with a 0.1 increment.

We observe in Fig. 5 that DynWing is close to the optimal solution that MSW provides, but still there is room for improvement in terms of precision and recall. For example, in the case of *Pooka*, with only $Histrace_{commits}$, DynWing (cross signs) provides on average 52.46 and 12.52 precision and recall values, whereas MSW provides on average 51.31 and 14.63: DynWing increased the recall value by 2.11 at the cost of a decrease in precision by 1.14. In the case of *Rhino* and *SIP*, we only show the average precision and recall of two experts in the graph for the clarity of the graph. Fig. 5 and Table 1 show that using more than one expert provides better results than the MSW weighting technique. However, Fig. 5 shows that for each system, the ranges of weights vary. Thus, we cannot identify one range of weights that would yield traceability links with reasonable precision and recall values for all the four systems. As the number of experts increases, it would become more difficult to identify the most appropriate weights or range of weights. Instead, DynWing would relieve project managers from choosing static weights.

We observe that treating each and every link independently helps to increase precision and recall. Some researchers [8], [9] performed several experiments using various weights to provide a range of weights that work well with their approaches/datasets. However, we cannot generalize such kinds of weights because every dataset is unique [10]. In addition, a range of weights is only a start to recover traceability links and, without an oracle, it is impossible to identify what weights and/or ranges of weights are suitable. Moreover, Fig. 5 shows that we cannot provide a reasonable range using MSW in the cases of *Pooka* and *SIP*: The MSW weight range for JSM is not suitable for VSM. Thus, we cannot impose the same weight range on all the datasets and IR techniques as proposed by other researchers [8], [9].

Gethers et al. [9] proposed a PCA-based weighting technique that does not require an oracle to define weights. Fig. 5 shows that the PCA-based weighting technique favors precision over recall. Sometimes it provides better results than DynWing in term of precision only. DynWing increases both precision and recall.

This comparison of DynWing with MSW also supports the answer of **RQ2** that DynWing tends to provide a better tradeoff between precision and recall. We conclude that DynWing does not require any previous knowledge of an

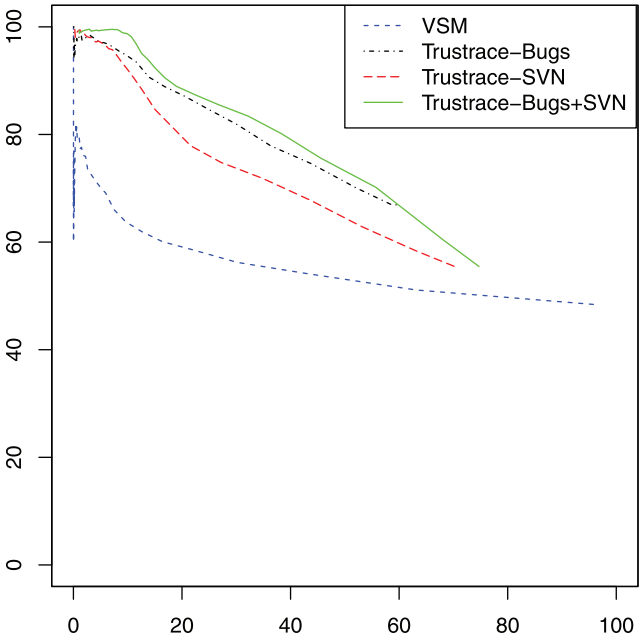

Fig. 7. Rhino precision and recall graph for $Trustrace$ using one and two experts. The X-axis shows precision values and the Y-axis recall values.

oracle and that treating each link separately and assigning weights on a per-link basis provides better precision and recall values than other weighting techniques. The achieved improvements are statistically significant and, most of the time, with large effect sizes.

5.3 Number of Experts

Fig. 4 shows the results of two experts only, in the case of *Rhino* and *SIP*, for the sake of clarity. All the detailed figures and results are available online.[8] We only include one figure of two experts as an example in this paper to show that adding more experts does impact precision and recall positively.

Fig. 7 shows that using one expert provides better results than an IR-based technique alone. As we increase the number of experts, the graph shows a clear improvement in precision and recall. Table 1 reports that, in the case of *Rhino* and *SIP*, using two experts increases the precision and recall values more than using a single expert. It supports the idea that increasing the number of experts increases the precision and recall values.

5.4 Other Observations

The empirical evaluation supports our claim that Trustrace combined with IR techniques is effective in increasing the precision and recall values of some baseline requirements' traceability links. Our novel approach performs better than JSM and VSM.

While creating $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$, we tagged some requirements as unsupported features. While we performed the empirical evaluation, we found that Histrace produced links to some unsupported features. We manually verified these links and found that the source code related to these features indeed existed. We updated our oracles accordingly, which are the ones used in the previous sections.

8. http://www.ptidej.net/download/experiments/tse12/.

For example, in *Pooka*, we declared the drag-and-drop requirement as an unsupported feature. Yet, Histrace produced some traceability links to this requirement. We manually verified these links and found that developers implemented the drag-and-drop feature partially, but named it "dnd" while commenting "this is drag and drop feature" in some related SVN commit messages. This example shows that Trustrace can indeed help developers in recovering missing links from human mistakes and the limitations of automated approaches. In the cases of *jEdit* and *Rhino*, we did not find such missing links.

This observation shows that Histrace not only helps to recover traceability links but also may help in evolving traceability links: If a developer creates traceability links and, after some years, wants to update the traceability links, then she does not need to create/verify all the links again. She could run Histrace and/or Trustrace to obtain possible missing links that she can verify.

5.5 Practical Applicability of Trustrace

Trustrace enables practitioners to automatically recover traceability links between requirements and source code. The current model of Trustrace is general in the sense that all the conceptual steps of Trustrace, shown in Fig. 1, can be changed. Indeed, Histrace could be customized to accommodate other software repositories, e.g., mailing lists. Trumo could be applied to other problems, e.g., feature location. For example, we recently customized Trumo to address bug location in combination with binary-class relationships. The most important aspect of Trustrace is that it does not require the tuning of some parameters for every dataset on which it is applied. DynWing is an automatic weighting scheme that assigns weights to the different experts at runtime. In contrast, a project manager would need to guess and assign weights to each expert.

The Trustrace model could be implemented in any software development environment. It does not require particular inputs or parameter tuning. Trustrace could mine any software repository and use them as experts to recover traceability links. A project manager could also use the output of Trustrace for other purposes than requirement traceability as well. For example, Trustrace could tell a project manager which requirements require more maintenance, in particular which requirements are causing more bugs and CVS/SVN commits.

5.6 Revisiting the Conjectures

In the introduction, we stated three conjectures regarding the use of other sources of information, of a trust model, and of a dynamic weighting technique to improve the accuracy of the requirement traceability links recovered using an IR technique.

Within the limits of the threats to the validity of the results of our empirical study, we conclude that our conjectures are true. Indeed, our empirical study shows that the requirement traceability links recovered through mining software repositories, i.e., CVS/SVN repositories and Bugzilla bug-tracking systems, can be considered as experts whose opinions can be used in a trust model to discard/rerank the links provided by an IR technique. The experts' opinions must be combined dynamically, i.e., on a per-link basis, to reap the full benefits of the trust model.

To the best of our knowledge, this paper is the first stating these conjectures and reporting on the benefits of combining software repositories to improve the accuracy of requirement traceability links. It is also the first use of a dynamic weighting technique in combination with a trust model. We expect that other software repositories could be useful during the traceability recovery process and that other models of trust and weighting techniques could improve the accuracy of the links recovered by an IR technique even further.

5.7 Threats to Validity

Several threats potentially impact the validity of our experimental results.

5.7.1 Construct Validity

Construct validity concerns the relation between theory and observations. We quantified the degree of inaccuracy of our automatic requirement traceability approach by means of a validation of the precision and recall values using manually built oracles. First, two authors created manual traceability oracles and then the third author verified their content to avoid inaccuracy in the oracles. We manually verified some of the automatically recovered links by our approach to discover any imperfection in manually built oracles. We improved the oracles after applying Trustrace and discovering missing links.

5.7.2 Internal Validity

The internal validity of a study is the extent to which a treatment's effects change the dependent variable. The internal validity of our empirical study could be threatened by our choice of the λ value: Other values could lead to different results. We mitigated this threat by studying the impact of λ on the precision and recall values of our approach in Section 4 and using MSW- and PCA-generated λ values. The global trust over an expert could also impact the validity of results. We mitigated this threat by using the same setting for all the experiments and found the same trends of improvements.

5.7.3 External Validity

The external validity of a study relates to the extent to which we can generalize its results. Our empirical study is limited to four systems, i.e., *jEdit*, *Pooka*, *Rhino*, and *SIP*. Yet, our approach is applicable to any other systems. However, we cannot claim that the same results would be achieved with other systems. Different systems with different SVN commit logs, requirements, bug descriptions, and source code may lead to different results. Yet, the four selected systems have different SVN commit logs, SVN commit messages, requirements, bug reports, and source code quality. Our choice reduces the threat to the external validity of our empirical study.

5.7.4 Conclusion Validity

Conclusion validity threats deal with the relation between the treatment and the outcome. The appropriate nonparametric test, Mann-Whitney, was performed to statistically reject the null hypotheses, which does not make any assumption on the data distribution. We also mitigated this threat by applying the Shapiro-Wilk test to verify the distribution of our data points to select an appropriate statistical test and effect size.

6 Related Work

Traceability recovery and web trust models are related to our work.

6.1 Traceability Approaches

Sherba and Anderson [30] proposed an approach, TraceM, based on techniques from open hypermedia and information integration. TraceM manages traceability links between requirements and architecture. An open hypermedia system enables the creation and viewing of relationships in heterogeneous systems. TraceM allows the creation, maintenance, and viewing of traceability relationships in tools that software professionals use on a daily basis. Maider et al. [31] proposed an approach to support automated traceability maintenance by recognizing development activities. Development activities are formally specified and changes to certain model elements trigger a LinkUpdateManager. This manager is responsible for updating traceability links that are related to the changed elements. However, the authors did not mention how traceability links are actually created and updated. Poshyvanyk et al. [8] combined a scenario-based probabilistic ranking of events and an IR technique that uses latent semantic indexing for feature location. Their empirical study shows that combining different approaches can perform better than a single IR technique.

Lucia et al. [32] proposed an approach helping developers to keep source code identifiers and comments consistent with high-level artifacts. The approach computes textual similarity between source code and related high-level artifacts, e.g., requirements. The textual similarity helps developers to improve their source code lexicon. Maletic and Collard [6] proposed an XML-based traceability query language, TQL. TQL supports queries across multiple artifacts and multiple traceability link types. TQL has primitives to allow complex query construction and execution support. Zou et al. [33] performed empirical studies to investigate Query Term Coverage, Phrasing, and Project Glossary term-based enhancement methods that are designed to improve the performance of a probabilistic automated tracing tool. The authors proposed a procedure to automatically extract critical keywords and phrases from a set of traceable artifacts to enhance the automated trace retrieval. Gethers et al. [9] proposed an integrated approach to combine orthogonal IR techniques, which have been statistically shown to produce dissimilar results. Their approach combines VSM, JSM, and relational topic modeling. Their proposed approach uses each IR technique as an expert and uses a PCA-based weighting scheme to combine them.

The precision and the recall [3] of the links recovered during traceability analyses are influenced by a variety of factors, including the conceptual distance between high-level documentation and low-level artifacts, the way in which queries are formulated, and the applied IR technique. Comparisons have been made among different IR techniques, e.g., [29] and [15], with inconclusive results. On several datasets, the vector space model and Jensen-Shannon similarity model perform favorably in comparison

TABLE 3
Related Work Summary

Approaches	External Info.	Soft. Repo.	Multiple Experts	Automated Weights	Tool Support
Trustrace	✓	✓	✓	✓	✓
[9]	✗	✗	✓	✓	✗
[30]	✗	✗	✗	✗	✓
[31]	✗	✗	✗	✗	✓
[6]	✗	✗	✗	✗	✗
[8]	✗	✗	✓	✗	✗
[35]	✓	✓	✗	✗	✗

to more complex techniques, such as latent semantic analyses [15] or latent Dirichlet allocation [17]. Yet, algebraic models, e.g., the vector space model [3], and probabilistic models, e.g., the Jensen-Shannon similarity model [15], are a reference baseline for both feature location [8], [34] and traceability recovery [3], [29].

Kagdi et al. [35] presented a heuristic-based approach to recover traceability links between software artifacts using the software system's version history. Their approach assumes that, if two or more files cochange [36] in the system history, then there is a possibility that they have a link between them. However, it is quite possible that two files are cochanging, but that they do not have any semantic relationship. It is also likely that some documents evolve outside the system's version control repository and, in such a case, their approach cannot find a link from/to these documents, e.g., requirement specifications. In addition, their approach does not analyze the contents of the CVS/SVN commit logs and files that were committed in CVS/SVN. More importantly, in co-change-based traceability [35], [36], [37], if two or more files have a link but they were not co-changed, then these approaches fail to find a link. Our proposed novel approach is not dependent on cochanges and overcomes these limitations.

Table 3 summarizes the related works on traceability recovery approaches. The column external information in Table 3 shows whether the approach uses any external information, e.g., execution traces, software repositories, human knowledge, or not. The multiple expert column shows if the current approach accommodates more than one expert opinion. The automated weights column shows if the approach is capable of assigning weights to each expert. Only [9] provides an automated support for assigning weights to multiple experts. In [9], the authors proposed a PCA-based weighting scheme for multiple experts. We compared DynWing with this PCA-based weighting scheme. Our results show that DynWing can provide better results than the PCA-based weighting scheme. To the best of our knowledge, all the above-mentioned approaches use textual similarity among various software artifacts to recover traceability links. The work presented in this paper is complementary to existing IR-based techniques because it uses current state-of-the-art techniques to create links and uses a trust model to filter out false-positive links and increase the trust over remaining links.

6.2 Web Trust Model

Our proposed novel approach is influenced by the web trust model [11], [12], [13], [14]. There are two types of trust in e-commerce: first, a customer's initial trust [14] when she interacts with a website for the first time and, second, the website's reputation trust [38] that develops over time and after repeated experiences. When customers hesitate to buy things online, they may ask their friends, family, and other buyers to make sure that they can trust a website. Many researchers investigated [11], [13], [14], [38], [39] the problem of increasing customers' trust in a website. Some researchers [11], [13] suggested that using other sources of information can increase the trust in a website.

Berg and Van [11] attempted to develop the equivalent of symbolons, for e-commerce. Their study deals with a specific type of electronic medium, the World Wide Web. In particular, it focuses on the so-called web assurance services, which provide certification of the legitimacy of websites. The authors highlight business-to-consumer commerce, but also pay attention to the challenges regarding web assurance services in the business-to-business environment.

Palmer et al. [13] presented an empirical investigation of how firms can improve customers' trust by exploring and using Trusted Third Parties (TTPs) and privacy statements. Their exploratory data show that the use of TTPs and privacy statements increase a customer's trust in a website. Wang et al. [40] presented a novel content trust learning algorithm that uses the content of a website as a trust point to distinguish trustable web contents and spam contents.

McKnight et al. [12] empirically tested the factors that may influence initial trust in a web-based company. The authors tested a trust building model for new customers of a fictitious legal advice website and found that perceived company reputation and perceived site quality both had a significant positive relationship with initial trust with the company.

Koufaris and Hampton-Sosa [14] propose a model to explain how new customers of a web-based company develop initial trust in the company after their first visit. The authors empirically tested their hypothesis using a questionnaire-based field study. The results indicate that perceived company reputation and willingness to customize products and services can significantly affect initial trust.

Our proposed approach uses traceability from requirements to source code as initial trust and then uses CVS/SVN commit logs and also uses bug reports, mailing lists, temporal information and so on, as reputation trust for a traceability link. As the reputation of a link increases, the trust in this link also increases.

7 Conclusion

The literature [32], [3], [41] showed that IR techniques are useful to recover traceability links between requirements and source code. However, IR techniques lack accuracy (precision and recall). In this paper, we conjectured that:

1) We could mine software repositories to support the traceability recovery process; 2) we could consider heterogeneous sources of information to discard/rerank the traceability links provided by an IR technique to improve its accuracy; and 3) we could use an automatic, per-link experts' weighting technique to avoid the need of manually built oracles to tune weights.

To support our conjectures, we proposed a new approach, Trustrace, to improve the precision and recall values of some baseline traceability links. Trustrace is based on mining techniques, on a trust model, and a dynamic weighting technique. Trustrace consists of three parts:

1. Histrace is a technique to create experts supporting the identification of traceability links between requirements and source code. We implemented two instances of Histrace: $Histrace_{commits}$ uses CVS/SVN commit messages and $Histrace_{bugs}$ uses bug reports to build traceability links between requirements and source code. The rationale of Histrace is that commits and bugs are tied to changed source-code entities and thus can be used to infer traceability links between requirements and source code.
2. Trumo, inspired by web-trust models [11], [12], [13], [14], improves the precision and recall values of some baseline traceability links. It uses any traceability-link recovery approach and various experts' opinions, i.e., $Histrace_{commits}$ and $Histrace_{bugs}$, to discard and/or rerank the traceability links provided by the recovery approach, thus improving their accuracy. Fig. 1 shows that Trumo indeed improves the accuracy of traceability links recovered by either JSM or VSM.
3. DynWing combines and assigns weights to the Histrace experts' opinions using a dynamic weighting technique that considers each link separately. Experts' opinions are then used by Trumo to rerank the traceability links. We compared the DynWing weighting technique with a PCA weighting technique and a multiple-static weighting technique and reported that DynWing provides better results in terms of precision and recall.

Trustrace is the first approach to integrate time information (from CVS/SVN commit logs) and bug information (from Bugzilla reports) to recover requirement traceability links. We applied Trustrace on *jEdit*, *Pooka*, *Rhino*, and *SIP* to compare its requirement-traceability links with those recovered using only the JSM or VSM techniques, in terms of precision and recall. We showed that Trustrace improves, with statistical significance, the precision and recall values of the traceability links. We also compared DynWing with a PCA-based weighting technique in terms of F_1 score. We thus showed that our trust-based approach indeed improves precision and recall and also that CVS/SVN commit messages and bug reports are useful in the traceability recovery process.

We thus conclude that our conjectures are supported: The accuracy of the traceability links between requirements and source code recovered by an IR technique are improved by 1) mining software repositories and considering the links recovered through these repositories as experts, 2) using a trust model inspired by web-trust models to combine these experts' opinions, and 3) weighting the experts' opinions on a per-link basis for each link recovered by the IR technique.

In future work, we plan to implement more instances of Histrace, in particular using e-mails and threads of discussions in forums. We will use our Trumo model in other software engineering fields, in particular, test-case prioritization using various prioritisation approaches, anti-pattern detection using users' feedback, and concept location using execution traces. We will deploy Trustrace in a development environment and perform experiments with real developers to analyze how effectively Trustrace can help developers in recovering traceability links. We also plan to use advanced matching techniques between bug reports and CVS/SVN commit messages, such as those in [19], [20].

Acknowledgments

The authors thank Dr. Bram Adams, assistant professor at the École Polytechnique de Montréal, for his invaluable feedback. This work is partly supported by the NSERC Research Chairs on Software Cost-effective Change and Evolution and on Software Patterns and Patterns of Software.

References

[1] O.C.Z. Gotel and C.W. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proc. First Int'l Conf. Requirements Eng.*, pp. 94-101, Apr. 1994.

[2] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trust-Based Requirements Traceability," *Proc. 19th IEEE Int'l Conf. Program Comprehension*, S.E. Sim and F. Ricca, eds., pp. 111-120, June 2011.

[3] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 970-983, Oct. 2002.

[4] A. Marcus and J.I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing," *Proc. 25th Int'l Conf. Software Eng.*, pp. 125-135, 2003.

[5] J.H. Hayes, A. Dekhtyar, S.K. Sundaram, and S. Howard, "Helping Analysts Trace Requirements: An Objective Look," *Proc. 12th IEEE Int'l Requirements Eng. Conf.*, pp. 249-259, 2004.

[6] J.I. Maletic and M.L. Collard, "TQL: A Query Language to Support Traceability," *Proc. ICSE Workshop Traceability in Emerging Forms of Software Eng.*, pp. 16-20, 2009.

[7] J.H. Hayes, G. Antoniol, and Y.-G. Guéhéneuc, "PREREQIR: Recovering Pre-Requirements via Cluster Analysis," *Proc. 15th Working Conf. Reverse Eng.*, pp. 165-174, Oct. 2008.

[8] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature Location Using Probabilistic Ranking of Methods Based on Execution Scenarios and Information Retrieval," *IEEE Trans. Software Eng.*, vol. 33, no. 6, pp. 420-432, June 2007.

[9] M. Gethers, R. Oliveto, D. Poshyvanyk, and A.D. Lucia, "On Integrating Orthogonal Information Retrieval Methods to Improve Traceability Recovery," *Proc. 27th IEEE Int'l Conf. Software Maintenance*, pp. 133-142, Sept. 2011.

[10] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, *Factors Impacting the Inputs of Traceability Recovery Approaches*, A. Zisman, J. Cleland-Huang, and O. Gotel, eds. Springer-Verlag, 2011.

[11] R. Berg and J.M.L. Van, "Finding Symbolons for Cyberspace: Addressing the Issues of Trust in Electronic Commerce," *Production Planning and Control*, vol. 12, pp. 514-524, 2001.

[12] D.H. McKnight, V. Choudhury, and C.K. Kacmar, "The Impact of Initial Consumer Trust on Intentions to Transact with a Web Site: A Trust Building Model," *The J. Strategic Information Systems*, vol. 11, nos. 3/4, pp. 297-323, 2002.

[13] J.W. Palmer, J.P. Bailey, and S. Faraj, "The Role of Intermediaries in the Development of Trust on the WWW: The Use and Prominence of Trusted Third Parties and Privacy Statements," *J. Computer-Mediated Comm.*, vol. 5, no. 3, 2000.

[14] M. Koufaris and W. Hampton-Sosa, "The Development of Initial Trust in an Online Company by New Customers," *Information & Management*, vol. 41, no. 3, pp. 377-397, 2004.

[15] A. Abadi, M. Nisenson, and Y. Simionovici, "A Traceability Technique for Specifications," *Proc. 16th IEEE Int'l Conf. Program Comprehension,* pp. 103-112, June 2008.

[16] A. Marcus and J.I. Maletic, "Recovering Documentation-to-Source-Code Traceability Links Using Latent Semantic Indexing," *Proc. Int'l Conf. Software Eng.,* pp. 125-135, May 2003.

[17] H. Asuncion, A. Asuncion, and R. Taylor, "Software Traceability with Topic Modeling," *Proc. 32nd ACM/IEEE Int'l Conf. Software Eng.,* vol. 1, pp. 95-104, 2010.

[18] M. Porter, "An Algorithm for Suffix Stripping," *Program: Electronic Library and Information Systems,* vol. 40, pp. 130-137, 1980.

[19] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The Missing Links: Bugs and Bug-Fix Commits," *Proc. 18th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.,* pp. 97-106, 2010.

[20] R. Wu, H. Zhang, S. Kim, and S. Cheung, "Relink: Recovering Links between Bugs and Changes," *Proc. 19th ACM SIGSOFT Symp. and 13th European Conf. Foundations of Software Eng.,* pp. 15-25, 2011.

[21] W.B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, 1992.

[22] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Requirements Traceability for Object Oriented Systems by Partitioning Source Code," *Proc. 18th Working Conf. Reverse Eng.,* pp. 45-54, Oct. 2011.

[23] M. Eaddy, T. Zimmermann, K.D. Sherwood, V. Garg, G.C. Murphy, N. Nagappan, and A.V. Aho, "Do Crosscutting Concerns Cause Defects?" *IEEE Trans. Software Eng.,* vol. 34, no. 4, pp. 497-515, July/Aug. 2008.

[24] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature Location in Source Code: A Taxonomy and Survey," *J. Software Maintenance and Evolution: Research and Practice,* 2011.

[25] J.H. Hayes and A. Dekhtyar, "A Framework for Comparing Requirements Tracing Experiments," *Int'l J. Software Eng. and Knowledge Eng.,* vol. 15, no. 5, pp. 751-782, 2005.

[26] J.H. Hayes, A. Dekhtyar, S.K. Sundaram, E.A. Holbrook, S. Vadlamudi, and A. April, "Requirements Tracing on Target (Retro): Improving Software Maintenance through Traceability Recovery," *Innovations in Systems and Software Eng.,* vol. 3, pp. 193-202, 2007.

[27] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving IR-Based Traceability Recovery Using Smoothing Filters," *Proc. 19th IEEE Int'l Conf. Program Comprehension,* pp. 21-30, June 2011.

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

[29] A.D. Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering Traceability Links in Software Artifact Management Systems Using Information Retrieval Methods," *ACM Trans. Software Eng. Methodology,* vol. 16, no. 4, article 13, 2007.

[30] S.A. Sherba and K.M. Anderson, "A Framework for Managing Traceability Relationships between Requirements and Architectures," *Proc. Second Int'l Software Requirements to Architectures Workshop, part of Int'l Conf. Software Eng.,* pp. 150-156, 2003,

[31] P. Mader, O. Gotel, and I. Philippow, "Enabling Automated Traceability Maintenance by Recognizing Development Activities Applied to Models," *Proc. 23rd IEEE/ACM Int'l Conf. Automated Software Eng.,* pp. 49-58, 2008.

[32] A.D. Lucia, M.D. Penta, and R. Oliveto, "Improving Source Code Lexicon via Traceability and Information Retrieval," *IEEE Trans. Software Eng.,* vol. 37, no. 2, pp. 205-227, Mar. 2011.

[33] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving Automated Requirements Trace Retrieval: A Study of Term-Based Enhancement Methods," *Empirical Software Eng.,* vol. 15, no. 2, pp. 119-146, 2010.

[34] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "SNIAFL: Towards a Static Noninteractive Approach to Feature Location," *ACM Trans. Software Eng. Methodology,* vol. 15, pp. 195-226, Apr. 2006.

[35] H. Kagdi, J. Maletic, and B. Sharif, "Mining Software Repositories for Traceability Links," *Proc. 15th IEEE Int'l Conf. Program Comprehension,* pp. 145-154, June 2007.

[36] H. Kagdi and J. Maletic, "Software Repositories: A Source for Traceability Links," *Proc. Int'l Workshop Traceability in Emerging Forms of Software Eng.,* pp. 32 -39, Mar. 2007.

[37] H. Kagdi, S. Yusuf, and J.I. Maletic, "Mining Sequences of Changed-Files from Version Histories," *Proc. Int'l Workshop Mining Software Repositories,* pp. 47-53, 2006.

[38] D. Artza and Y. Gil, "A Survey of Trust in Computer Science and the Semantic Web," *Web Semantics: Science, Services and Agents on the World Wide Web,* vol. 5, no. 2, pp. 58-71, 2007.

[39] T. Grandison and M. Sloman, "A Survey of Trust in Internet Applications," *IEEE Comm. Surveys Tutorials,* vol. 3, no. 4, pp. 2-16, Fourth Quarter 2000.

[40] W. Wanga, G. Zenga, and D. Tang, "Using Evidence Based Content Trust Model for Spam Detection," *Expert Systems with Applications,* vol. 37, no. 8, pp. 5599-5606, 2010.

[41] L. Dapeng, M. Andrian, P. Denys, and R. Vaclav, "Feature Location via Information Retrieval Based Filtering of a Single Scenario Execution Trace," *Proc. 22nd IEEE/ACM Int'l Conf. Automated Software Eng.,* pp. 234-243, 2007.

Nasir Ali received the MSc degree in computer science and the MBA degree from the University of Lahore and National College of Business Administration & Economics, respectively, and is currently working toward the PhD degree in the Department of Computing Engineering and Software Engineering at the École Polytechnique de Montréal. The primary focus of his PhD thesis is to develop tools and techniques to improve the quality of software artifacts' traceability. He has more than six years of industrial experience. His research interests include software maintenance and evolution, system comprehension, and empirical software engineering. He is a student member of the IEEE.

Yann-Gaël Guéhéneuc received the PhD degree in software engineering from the University of Nantes, France (under Professor Pierre Cointe's supervision) in 2003 and an Engineering Diploma from the École des Mines of Nantes in 1998. His PhD thesis was funded by Object Technology International, Inc. (now IBM OTI Labs.), where he worked in 1999 and 2000. He is a full professor in the Department of computing and software engineering at the École Polytechnique de Montréal where he leads the Ptidej team on evaluating and enhancing the quality of object-oriented systems by promoting the use of patterns, at the language-, design-, or architectural-levels. In 2009, he was awarded the NSERC Research Chair Tier II on Software Patterns and Patterns of Software. His research interests include system understanding and system quality during development and maintenance, in particular through the use and the identification of recurring patterns. He was the first to use explanation-based constraint systeming in the context of software engineering to identify occurrences of patterns. He is also interested in empirical software engineering; he uses eye-trackers to understand and to develop theories about system comprehension. He has published many papers in international conferences and journals. He has been a senior member of the IEEE since 2010.

Giuliano Antoniol received the degree in electronic engineering from the Università di Padova in 1982 and the PhD degree in electrical engineering from the École Polytechnique de Montréal in 2004. He worked in companies, research institutions, and universities. In 2005, he was awarded the Canada Research Chair Tier I in Software Change and Evolution. He has published more than 100 papers in journals and international conferences. He has served as a member of the program committees of international conferences and workshops, such as the International Conference on Software Engineering, the International Conference on Software Maintenance, the International Conference on Program Comprehension, and the International Symposium on Software Metrics. He is presently a member of the Editorial Board of the *Journal of Software Testing Verification & Reliability*, the *Journal of Information and Software Technology*, the *Journal of Empirical Software Engineering*, and the *Journal of Software Quality*. He is currently a full professor at the École Polytechnique de Montréal, where he works in the area of software evolution, software traceability, software quality, and maintenance. He is a member of the IEEE.