Assessing traceability of software engineering artifacts

Senthil Karthikeyan Sundaram · Jane Huffman Hayes · Alex Dekhtyar · E. Ashlee Holbrook

Abstract The generation of traceability links or traceability matrices is vital to many software engineering activities. It is also person-power intensive, time-consuming, error-prone, and lacks tool support. The activities that require traceability information include, but are not limited to, risk analysis, impact analysis, criticality assessment, test coverage analysis, and verification and validation of software systems. Information Retrieval (IR) techniques have been shown to assist with the automated generation of traceability links by reducing the time it takes to generate the traceability mapping. Researchers have applied techniques such as Latent Semantic Indexing (LSI), vector space retrieval, and probabilistic IR and have enjoyed some success. This paper concentrates on examining issues not previously widely studied in the context of traceability: the importance of the vocabulary base used for tracing and the evaluation and assessment of traceability mappings and methods using secondary measures. We examine these areas and perform empirical studies to understand the importance of each to the traceability of software engineering artifacts.

Keywords Requirements traceability · Automated tracing · Candidate link generation · Vocabulary base · Secondary measures

1 Introduction

The importance of traceability of textual artifacts generated during the software development lifecycle has been well established in recent years [38]. The top–down traceability of documents from a software project document hierarchy (requirements-to-design, design-to-code, design-to-test cases, etc.) provides assurance that all required features (and only they) have been implemented and properly tested. The easiest and most efficient way to ensure traceability is to prepare traces at the time of creation of project artifacts, e.g., record the requirements-to-design traceability matrix (RTM) while preparing the design document. Often, however, this is not done (or not to the proper level of detail), leaving requirements tracing to post-development (e.g., to the Independent Verification and Validation (IV&V)[1] analyst or to a tester or stakeholder who finds that they need the RTM in order to perform their work). Tracing from scratch, or even validating an existing RTM, is a tedious, error-prone, and time-consuming process typically performed by analysts with minimal tool assistance. Traditional methods for tracing two documents to each other include manual

[1] IV&V is performed by a party other than the software system developer and is the process of ensuring that the software process is followed as well as ensuring that the developed software performs as expected.

keyword assignment and keyword searches in word processors or spreadsheets [17].

Our work on the traceability problem has been motivated by the state-of-the-art in Independent Verification and Validation (IV&V) of software systems. IV&V is a mandatory stage in the process of development and deployment of most mission- or safety-critical software systems [18, 29]. Third-party analysts receive a full collection of artifacts for a software product and perform a set of verification and validation tasks prior to the final deployment of the software. Among the IV&V tasks performed, *establishing after-the-fact traceability mapping* (otherwise known as the Requirements Traceability Matrix or RTM) between the product requirements and design (as well as other artifacts, such as test cases) has traditionally been one of the most tedious and time-consuming tasks.

In our previous work [15, 16, 20], we studied approaches to automating the requirements tracing process by using Information Retrieval (IR) methods to propose candidate links between a pair of project artifacts as well as ways to incorporate automated traceability into the IV&V process [14, 29].

Incorporation of automated tracing in an IV&V process presents a number of challenges, chief among which is the still-remaining need for the IV&V analyst to validate and sign off on any RTM generated within the IV&V process [18]. The software tools we have built [14] are designed to work in concert with a human analyst. The software delivers the candidate traceability matrix to the analyst who can observe, validate, and correct the results, and provide feedback to trigger re-computation of certain portions of the candidate RTM. We have learned that automated methods can produce reasonably accurate results in much less time than it takes a human analyst to perform similar work [20].

Tracing software engineering artifacts to each other is similar to traditional information retrieval tasks such as web search. Artifacts are broken into individual elements (e.g., requirements, design elements, test cases). Elements of one artifact are treated as documents in a document collection, while elements of the other are treated as search queries. At the same time, software artifacts pose a number of unique challenges as well. IR methods are designed for very large collections of reasonably large documents. On typical IR scale, a requirements specification, viewed as a collection of individual requirements, is quite small even for larger software projects. Additionally, elements from software artifacts are significantly smaller in size than typical documents in IR. A third issue is the fact that the corpus (i.e., collection of all words) used in creating the software artifacts is small and does not obey typical word distribution in English.

It is fair to qualify the state-of-the-art in automated tracing as follows. Multiple groups [1, 4, 20, 28] have achieved proof-of-concept success with the use of IR methods for tracing. These methods provide *reasonable answers quickly.* However, *no method is yet capable of generating complete and correct RTMs* from pairs of textual artifacts. IR methods tend to capture true links in the RTM very well, but typically at the price of a high rate of false positive detection.

At this point, there are three possible directions in which research on and practical adoption of automated traceability methodology for tracing can proceed. *First,* new automated methods can be developed with emphasis on higher quality output. *Second,* existing automated processes can be improved to produce better quality candidate RTMs. *Third,* the work of human analysts with the results of automated tracing methods can be studied with an eye on improving the human–computer interaction and the quality of the RTM revised by the human analyst. Human analysts are already a significant part of the traceability loop in Independent Verification & Validation processes such as tracing, testing, and change impact assessment [18, 29]. By introducing (and/or increasing) the role of human analysts to other processes, it may be possible to achieve better traceability and achieve it faster than the current state-of-industry.

The first direction is being actively explored [5, 6, 30]. However, the results so far suggest that use of new IR methods provides at best marginal improvement over the traditional retrieval methods. While this aspect of today's research on traceability was not the focus of this paper, some of the work conducted by our research group and reported here illustrates that point (see Sect. 3 for description of methods used and Sect. 4 for results).

The third direction, introduced in [18] and discussed at TEFSE'07 [21] and TEFSE'09 [43], is an emerging research area. We describe the importance and the role of human analysts within the IV&V process in Sect. 2, in the context of an overall overview of traceability research and practices today.

The second direction is the focus of this paper. The general idea behind this approach is straightforward. IR methods produce a candidate RTM. This RTM is not quite correct—e.g., it can contain multiple false positive links. We can use various analyses on the data (the textual artifacts and the candidate RTM built for them) to attempt to improve upon the original candidate RTM. Putting humans in the tracing loop is one such way. But the goal of the second direction is to minimize the work of humans by (1) getting the most out of the IR method(s) used for RTM recovery and (2) improving upon the candidate RTMs supplied by the IR methods.

In this paper, we present two sets of experiments that address issues (1) and (2) from above. Our first experiment was designed to understand the differences in performance of three IR methods, *vector space retrieval, vector space retrieval with thesaurus,* and *latent semantic indexing,* when run under two different sets of conditions. In a typical IR system, the document collection is available up-front. It is analyzed, and the results of the analysis are stored. When a query comes, it is analyzed and compared to the representations of individual documents in the collection. In such a system, the *vocabulary* or *the corpus of terms* used to match documents to queries *comes exclusively from the document collection.* In the case of RTM recovery, both artifacts are known up-front. We can analyze the "queries" (elements of the high-level artifact) and the "documents" (elements of the low-level artifact) together. Thus, we have a choice: we can follow the tradition of IR systems and use only the *vocabulary of the low-level artifact* as the corpus of terms, or we can use the *joint vocabulary of both documents.* The first experiment in the paper uses two regular datasets, one small and one of medium size, plus a third dataset which consists of 22 pairs of artifacts to see how the four IR methods listed above fare in RTM recovery when executed with each of the two possible *vocabulary bases.*

To understand how we can improve on candidate RTMs *after* they have been reported by IR methods, we need to be able to properly evaluate them. Our second set of experiments addresses the issue of proper assessment of the quality of candidate RTMs. Traditionally, Information Retrieval uses *recall,* the percentage of true links retrieved, *precision,* the percentage retrieved links that are true, and their harmonic mean called *the f-measure* to evaluate the quality of retrieval. However, it is possible to have two candidate RTMs that have the same precision and recall but are different in terms of their quality. The motivation for our work comes from the example in Fig. 1. An IV&V analyst is using an automated tracing tool to search for the design elements that help satisfy a given high-level requirement. Two tracing tools, A and B, provide a similar user interface but differ in the methods that compute the candidate RTM. Each tool provides sorted results to the analyst's query, result list A (from tool A) and B (from tool B). Both result lists have recall of 100% (all correct links are found) and precision of 50% (half of the retrieved links are false positives): very good results. Result list A displays all the false positives in the top portion of the list, while result list B displays all the true links at the top of the list and false positives at the bottom of the list. Recall and precision indicate that these two tools have identical quality levels from the developer's perspective. However, from the perspective of the analyst who must review the results and make final selections, result list B (and thus tool B) is far superior since it requires less perusing of invalid results. Also, the analyst may have a more positive perception of the tool as the results seem more trustworthy or believable [16, 20]. As recall and precision cannot distinguish between tools A and B, more measures have to be introduced. Such *secondary measures* should be responsible for capturing the "internals" of the result lists (often called candidate lists), ensuring, for example, that list B will be deemed superior to list A.

We have found that secondary measures can be used to capture these characteristics and to help evaluate the quality of a returned list from the analyst's perspective. It is important to have a complete and correct picture of the effectiveness of IR methods and tools for requirements-related software engineering tasks. Otherwise, we may incorrectly reject good techniques or incorrectly accept poor techniques. Toward that end, we have developed several secondary measures and have examined their ability to evaluate returned lists. We illustrate the power of secondary measures with a number of examples where primary measures show one picture, but secondary measures tell a different story.

The paper is organized as follows. Section 2 provides a general overview of the state of research and practice in the area of software traceability, describes the software engineering processes that are affected and can be improved by better and faster traceability, and discusses the role of human analysts in the traceability loop. Section 3 explains the research approach and discusses vocabulary bases and secondary measures in detail. Section 4 details the vocabulary base experiments and the results obtained. Section 5 discusses the experimental

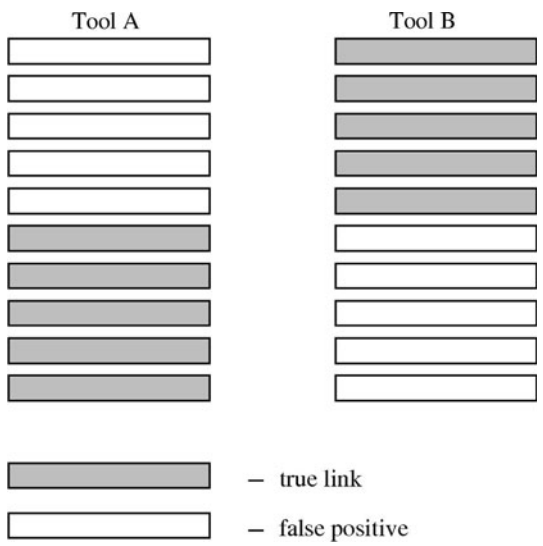

Fig. 1 Precision and recall do not suffice for evaluating the results from the analyst's perspective

design and the results obtained for the secondary measure study. Sections 4 and 5 include subsections on the corresponding related work. Finally, Sect. 6 presents the conclusions and future work.

2 Traceability in a nutshell: state-of-the-art and challenges

In this section, we provide a brief description of the state-of-the-art in traceability research in order to put the work described in this paper in broader context.

Spanoudakis and Zisman [38] define software traceability as the "ability to relate artefacts created during the development of a software system to describe the system from different perspectives." This purposefully broad definition yields a wide range of possible forms, processes, and reasons for studying software traceability.

2.1 Motivations for traceability [38]

There are two distinctly different contexts which give rise to the study of traceability: (1) *process compliance and product improvement* and (2) *software understanding and reuse*. In the former case, traceability work is performed as part of an ongoing software development process. Its results are applicable to the software project at hand. In the latter case, traceability work is performed on completed project data, and its results do not contribute directly to the product improvement but rather are used in the product and process analysis.

2.2 Links between software artifact elements

The key to traceability is the notion of a *link* between the elements (e.g., requirements and design elements) of two software artifacts. In some settings [17], *links* in the context of traceability are represented as or understood as navigable *hyperlinks* explicitly incorporated in one or both artifacts during the appropriate stage (requirements elicitation, design, testing, reengineering) of the software product lifecycle. This approach is illustrated by the RETH (Requirements Engineering Through Hypertext) system [9, 24] that uses a semi-automated process to engage the software engineer in the process reengineering of a textual artifact to incorporate hyperlinks between different elements. Such a process, in the case of the RETH system, lead to perfect precision.

In contrast to this understanding of the notion of link, our work assumes that a traceability link is a relationship between two elements of two (different) artifacts. This assumption is independent of the representation of the relationship. The representation or storing of the relationship, be it hyperlink, database index entry, pointer, etc., is an important area of further research in the traceability community. Generally speaking, the traceability research community has yet to focus on the aspects of link representation. We discuss prior work on traceability with this in mind.

2.3 Traceability in the software lifecycle [38]

Within the software development process, traceability plays an important role in change impact analysis, change management, testing, and verification and validation (V&V).[2] Traceability of artifacts can usually be achieved in one of two ways: by creating and maintaining traceability information as a by-product of development or by performing after-the-fact tracing of necessary artifacts as a dedicated part of the process (e.g., change impact analysis or V&V). In general, while maintaining traceability as a by-product of development is desirable, it is also time-consuming and is rarely done to the necessary level of granularity. At the same time, development of mission-critical projects (e.g., mission-critical software produced by NASA [29]) is subject to government regulations requiring Independent Verification & Validation (IV&V), part of which *is* validation, and, when necessary, *recovery* of traceability information across the entire hierarchy of project artifacts.

2.4 Traceability analysis [38]

Three types of traceability analysis can be distinguished: *manual, semi-automatic*, and *automatic*. Manual analysis of traceability has the analyst responsible for the search for and final decisions on links between artifacts. Automatic traceability analysis is performed by special-purpose tracing software responsible for the searching for and retrieval of links and the final traceability matrix. A plethora of semi-automatic approaches have been posited including (a) rule-based approaches [10, 11, 27, 37] which generate links based on user-defined rules, used to match portions of different artifacts; (b) process-driven approaches [8, 34] which capture traceability information by using special-purpose software to monitor software development; and (c) collaborative approaches [18, 43], which involve automated traceability tools guiding the search for traceability links and human analysts rendering final decisions on the candidate links.

[2] V&V is the same as IV&V but does not have to be performed by a third party, it can be performed by the developer.

2.5 State-of-the-art in traceability

While the study of traceability has been attracting more and more attention in recent years, the reality «on the ground» is sobering. In industry, traceability analysis most often is performed after-the-fact and manually. Analysts find little traceability information created in parallel with the artifacts, and when they do, this information is often unreliable and needs to be validated. Most traditional requirements management tools [22, 41] require manual generation of traceability links. Other approaches taken by analysts involve the use of word processing/spreadsheet software, manual keyword assignment, and use of text search functions to find candidate matches. In general, tracing processes are time-consuming and tedious and thus tend to be error-prone, as the analyst gets tired of the activity.

2.6 Traceability in IV&V

Our group has worked on automating traceability for the IV&V process. IV&V is an expensive process, as it involves third-party analysts. It is used for verification and validation of software when the necessity to guarantee proper operation outweighs the costs associated with hiring the third party to perform V&V. An example of such a situation is the IV&V analysis of all mission-critical and safety-critical software deployed by NASA on its manned and unmanned flight programs [29].

Traceability is one of the most time-consuming activities in the IV&V process. Even if traceability information is present in the artifacts, IV&V analysts still must validate the RTMs provided to them, which commonly involves recreating the trace from scratch. This is typically done using one of the manual techniques described above.

While automated tracing methods can be used to *supplant* human IV&V analysts, they cannot be used to *replace* them. IV&V analysts must certify the correctness of the software system and/or discover latent defects. As such, IV&V analysts bear critical responsibilities, not associated with the work of analysts in other traceability-related contexts (e.g., reverse engineering of existing software). IV&V analysts *must* inspect the results of any automated traceability analysis and certify or correct them. A comprehensive discussion of the issue of developing tracing software for IV&V is out of scope for this paper but has been detailed elsewhere [18].

3 Research approach

This section presents the research approach for the two experiments in this paper.

3.1 Vocabulary base

In a typical IR setting, we have a collection of documents and a user information need expressed as a natural language query (or just as a sequence of keywords). The task of an IR method is to retrieve, for the collection, the documents deemed *relevant* to the query. Different IR methods define the notion of relevance in different terms and use different means to encode the content of the documents in the collection. When considered in the context of requirements tracing, IR methods can be applied as follows. The low-level artifact (a design document, for example) is treated as a "document collection," with each low-level element viewed as an individual "document" in this collection. Each high-level element (say from a requirements document) is treated as the request to find low-level elements relevant to it. The low-level elements returned by an IR method, for each of the high-level requirements, form the candidate RTM.

Our work to date has concentrated on the study of three categories of IR methods: vector space retrieval (see Sect. 3.1.1) [15, 16, 20], latent semantic indexing (LSI) (see Sect. 3.1.2) [13, 40], and so-called keyword extraction methods [19].

3.1.1 Vector space retrieval

Standard Vector space model [3] can be defined as follows. Let $V = \{k_1,\ldots,k_N\}$ be the vocabulary of a given document collection. Then, a vector model of a document d is a vector $(w_1,\ldots,w_N)$ of keyword weights, where w_i is computed as:

$$w_i = tf_i(d) \cdot idf_i.$$

Here, $tf_i(d)$ is the so-called term frequency: the (usually normalized) frequency of keyword k_i in the document d, and idf_i, called inverse document frequency is computed as:

$$idf = \log_2\left(\frac{n}{df_i}\right) \quad \left($$

where n is the number of documents in the document collection, and df_i is the number of documents in which keyword k_i occurs. Given a document vector $d = (w_1,\ldots,w_N)$ and a similarly computed query vector $q = (q_1,\ldots,q_N)$, the similarity between d and q is defined as the cosine of the angle between the vectors:

$$\mathrm{sim}(d,q) = \cos(d,q) = \frac{\sum_{i=1}^{N} w_i \cdot q_i}{\sqrt{\sum_{i=1}^{N} w_i^2 \cdot \sum_{i=1}^{N} q_i^2}}.$$

The above weighting scheme is called the *tf-idf* weighting scheme. In addition to *tf-idf*, we used two other weighting

schemes, namely the Okapi weighting scheme [36] and LTU [23].

In the Okapi scheme, the keyword weight is calculated as follows:

$$w_i = \frac{tf_i(d)}{0.5 + 1.5\frac{dl}{\text{avg\_dl}} + tf} \log\left(\frac{n - df_i + 0.5}{df_i + 0.5}\right)$$

where dl is the length of the document under consideration, avg_dl is the average length of the documents in the document collection, n is the total number of documents, $tf_i(d)$ is the term frequency of the ith term in document d and df_i is the document frequency of the ith term in the document collection.

In the Linear Threshold Unit (LTU) weighting scheme, the keyword weight is calculated as follows:

$$w_i = \frac{(\log(tf_i(d)) + 1)\log\left(\frac{n}{df_i}\right)}{0.8 + 0.2\frac{dl}{\text{avg\_dl}}}.$$

Use of thesaurus. We have used vector space retrieval both with and without an artifact-specific thesaurus—a list of synonyms, homonyms, and abbreviations that allows us to match terms such as "fault" in the high-level document to terms such as "error" in the low-level document. Given a thesaurus $T = \{\langle k_i, k_j, \alpha_{ij}\rangle\}$, where k_i and k_j are matching thesaurus keywords and α_{ij} is the similarity coefficient between them, then the similarity between d and q can be calculated as follows:

$$\text{sim}(d,q) = \cos(d,q) = \frac{\sum_{i=1}^{N} w_i \cdot q_i \sum_{\langle ki,kj,\alpha ij\rangle \in T} \alpha ij(wi \cdot qj + wj \cdot qi)}{\sqrt{\sum_{i=1}^{N} w_i^2 \cdot \sum_{i=1}^{N} q_i^2}}.$$

Vocabulary base. As stated above, all vector space retrieval methods represent both documents and queries as vectors of weights over the space $V = \{k_1,\ldots,k_N\}$ of keyword weights. In traditional IR settings, document collections are large, stable, and known up-front, while queries are generated dynamically. In such situations, V is the list of all keywords found in the document collection. Any query terms not found in the document collection will be ignored.

When performing traceability tasks, both high-level (queries) and low-level (documents) artifacts are known up-front,[3] and, in fact, both artifacts are usually processed at the same time and side-by-side. This raises the question of what *is* the "proper" vocabulary base in our case. Is it better to stick with the traditional approach of using only keywords from the low-level artifact, or will using the combined vocabulary of both documents yield improved results?

We note here that the influence of the vocabulary base on the performance of the vector space retrieval is quite subtle. In both cases (with the exception of thesauri-based retrieval), a direct keyword match can occur only if the keyword is found in both the high-level and the low-level documents. The presence of extra terms in the vocabulary will affect the relative importance of terms, but will not add new matches or negate existing ones.

3.1.2 Latent semantic indexing

Small datasets, as found in software engineering traceability applications, allow us to use latent semantic indexing (LSI) [7]. LSI is a dimension reduction technique based on singular value decomposition (SVD) of the term-by-document matrix that can be constructed by putting the tf-idf vectors of all documents in a single matrix. SVD transforms the original matrix into a product of two orthogonal matrices and a diagonal matrix of eigenvalues. By considering only the top k eigenvalues, we can obtain an approximation of the original matrix by a smaller matrix. Rows of the matrix can be compared to each other using the cosine similarity described above. For example, if L is a document-by-term weight matrix of dimension $A \times B$, its SVD is written as $L = TSD'$, where T is a matrix with orthogonal rows, D' is a matrix with orthogonal columns, and S is a diagonal matrix of eigenvalues of L. We can trim the list of eigenvalues of L from rank(L) to a smaller number k and obtain a decomposition $L_k = TS_kD'$, where S_k is the diagonal matrix of size $k \times k$ with the k largest eigenvalues of L on the diagonal. Rows of the matrix TS_k^2D can be compared to each other using the cosine similarity as defined above. Use of the matrix TS_k^2S instead of the original matrix L reduces the dimensionality of the document vectors from B to k [7].

Vocabulary base. As with vector space retrieval, we consider two ways to build the reduced-dimensionality matrix:

- from the low-level artifact only. This is the standard approach to building the reduced matrix. Singular-valued decomposition (SVD) is applied to the element-by-keyword matrix consisting only of the low-level artifact vectors, and
- from both low-level and high-level artifacts. Just as above, we can leverage the fact that we know our queries in advance (see footnote 3). Instead of generating query vectors after performing LSI on the low-level element vectors, we can add high-level element vectors to the element-by-keyword matrix on which SVD is performed.

[3] See discussions of the process for IV&V "after-the-fact" tracing in Sects. 1 and 2.

Once SVD is complete and the new number of dimensions is selected, we extract the query vectors from the reduced matrix directly and compare them to the reduced vectors for the low-level documents.

Generally speaking, vocabulary base has a more pronounced effect on the behavior of LSI retrieval, as it affects the dimensionality of the starting matrix for the LSI process and thus may significantly alter the reduced matrix.

3.1.3 Methods for building the corpus

The IR methods described above typically start by building a corpus that contains all of the terms or words found in the artifacts that will be traced. As described above, the corpus can be built using both of the artifacts, or it can be built using just one of the artifacts. This distinction has not been studied by researchers, and we address it here. Our study will examine the impact of vocabulary base on the accuracy of the IR methods for tracing and is discussed in Sect. 4.

3.2 Secondary measures

As mentioned in the sect. 1 and illustrated in Fig. 1, there are many situations where recall and precision do not provide sufficiently accurate information about the structure of the candidate lists. To address this need, we have developed secondary measures for evaluating IR techniques as applied to software engineering artifact tracing from the analyst's perspective. These can be applied to techniques used for a variety of purposes, but we focus on tracing in this paper. Our measures help assess the quality of returned candidate lists and help select a ''best list'' when recall and precision are about the same for all lists. This is useful in comparing different IR techniques (where each returns a list), different levels of analyst feedback etc.

Before defining the secondary measures, it is useful to examine the primary IR measures of recall, precision, and f-measure. Recall is defined as the ratio of the number of links returned by the IR method to the total number of possible links.

$$\text{Recall} = \frac{\text{Number of matches found by the IR method}}{\text{Total number of possible matches}}$$

Precision is computed as the fraction of the relevant documents in the list of all documents returned by the IR method.

$$\text{Precision} = \frac{\text{Number of true links found}}{\text{Total number of links returned by the IR method}}$$

An ideal IR or traceability method should produce as high precision and recall as possible. *F*-measure is a harmonic mean of precision and recall:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

It can be seen that achieving high precision and high recall is a balancing act. The above-mentioned formula puts equal preference to both recall and precision. The β parameter is introduced in the above formula to tilt the balance one way or the other. The parameter β can be altered to set the desirable significance for either recall or precision:

$$F_\beta = \frac{(\beta + 1) \cdot \text{precision} \cdot \text{recall}}{\text{recall} + \beta \cdot \text{precision}}$$

If $\beta > 1$, the recall will be valued more than precision, and if $\beta < 1$, the precision will be valued more than recall.

Relevance feedback takes advantage of the analyst input to improve the performance of the retrieval algorithms. Specifically, we use Standard Rochio method [3] to perform analyst feedback in our work. In the case of the vector space model, the relevance feedback technique adjusts the keyword weights of the query vector based on the information provided by the analyst. The new query vector q_{new} is computed as follows:

$$q_{new} = \alpha q + \left(\frac{\beta}{r}\sum_{d_j \in D_r} d_j\right)\left(-\frac{\lambda}{s}\sum_{d_k \in D_{irr}} d_k\right)\left($$

As shown in the above formula, the factor corresponding to the document vectors identified as relevant can potentially increase the recall, and the factor corresponding to the document vectors identified as irrelevant can potentially increase the precision. The constants α, β, γ in the formulas above can be adjusted in order to emphasize positive or negative feedback as well as the importance of the original query vector. D_r is the set of documents deemed as relevant, and D_{irr} is the set of documents deemed as irrelevant, as deemed by the user. Again, the similarity is recomputed with the query vector q_{new}.

The proposed measures are presented below. The first three measures deal with the quality of the individual returned lists.

3.2.1 DiffAR

DiffAR is designed to evaluate the internal structure of candidate link lists. Informally, *DiffAR* is the difference between the average similarity of a relevant match (an item in the list that is truly relevant to the query) and a false positive (an item in the list that is not relevant) in

the list of candidates returned by an automated tool. More formally, we define *DiffAR* as follows. Given textual artifacts $H = (h_1,\ldots,h_m)$ and $D = (d_1,\ldots,d_n)$, let $L = \{(d, h) | sim(d, h)\}$ be the set of all candidate matches returned by some IR method. L consists of two types of candidates: true matches and false positives. Let L_T be the set of true matches and L_F be the set of false positives of L. Then, *DiffAR* is defined as:

$$DiffAR = \frac{\sum_{(d,h)\in L_T} sim(d,h)}{|L_T|} - \frac{\sum_{(d',h')\in L_F} sim(d',h')}{L_F}.$$

In general, the higher the value of *DiffAR*, the more distinct true matches become in the candidate lists.

3.2.2 DiffMR

Measures that rely on averages are known to be sensitive to extreme values. *DiffMR* is a version of the *DiffAR* measure that relies on medians rather than averages:

$$DiffMR = med_{(d,h)\in L_T}(sim(d,h)) - med_{(d',h')\in L_F}(sim(d',h')).$$

3.2.3 Lag

DiffAR and *DiffMR* look at the quantitative difference between the similarity scores of true matches and false positives. Note that $L = \bigcup_{h\in H} L_h$, where $L_h = \{d, h) | sim(d,h) > 0\}$, i.e., L is constructed out of candidate lists for each element $h \in H$. But, it is possible that for some $h \in H$, a similarity of 0.3 is very high, whereas for some other $h' \in H$, it is rather low, and such nuances are missed in the computation of *DiffAR* and *DiffMR*. *Lag* is the measure designed to address this potential problem.

Let d be an element of a textual artifact D and h be an element of another textual artifact H. Let (h, d) be a true match returned by an IR method in the list L_h of candidate links for h. The Lag of the link (h, d), denoted Lag(h, d), is the number of false positive links (h, d') that have higher similarity scores than (h, d). Informally, the *Lag* of a true match is the number of false positives above it in the list of candidate matches. The overall *Lag* of a list of candidate matches L is the average *Lag* of a match:

$$Lag = \frac{\sum_{(h,d)\in L} Lag(h,d)}{|L|}.$$

Lag specifies, on average, how many false positives are found in the candidate lists above true links. The lower it is, the higher is the separation between true matches and false positives (note here that if a false positive has the highest relevance in a list of candidate links, it contributes 1 to the *Lag* of each true link in the same list).

3.2.4 Selectivity

The final secondary measure we describe here is *selectivity*. Unlike previously described *DiffAR*, *DiffMR*, and *Lag*, *selectivity* does not look into the internal structure of the list of candidates. Rather, it can be used in lieu of precision in order to determine whether the candidate lists returned by a text mining tool are of acceptable sizes.

In general, when an analyst has to perform a subcomponent matching (tracing) task manually, there are $n \times m$ potential candidate matches to be checked: each component of artifact H needs to be compared to each component of artifact D. As mentioned above, an automated mining method produces a list L of candidate matches. Selectivity of the method is defined as:

$$selectivity = \frac{|L|}{m \cdot n}.$$

Selectivity measures the savings incurred by the analyst when manually going through the list generated by an automated method rather than manually comparing each pair of elements. The smaller the selectivity, the better the savings for the analyst.

Selectivity is not an exact measure of effort savings, because it assumes that the analyst will be correcting only type I errors (errors of commission) found in the candidate RTM. It needs to be considered in concert with *recall*. The higher the recall, the fewer *errors of omission* the analyst needs to fix, the better *selectivity* approximates effort savings.

4 Vocabulary base study

In this study, we compare the results of using *tf-idf* term weighting for two different vocabulary bases: low-level artifact and both low-level and high-level artifacts. We have considered three different datasets.

4.1 Datasets

The datasets used are described below.

4.2 MODIS

The NASA Moderate Resolution Spectrometer (MODIS) dataset [26, 33] is a small dataset created from the full specification (high- and low-level requirements documents) for the MODIS space instrument software. This dataset contains 19 high-level requirements, 49 low-level requirements, and a validated RTM containing 41 links that we refer to as the "answer set." The answer set was manually constructed by the authors and checked by a

number of senior analysts with significant tracing experience [33].

4.3 CM-1

The dataset consists of a complete requirement and a complete design document for a NASA space instrument. We manually extracted individual requirements (235) and design elements (220) from the documents. We consider the forward tracing task, from requirements to design elements. The answer set, containing 361 links, was constructed by a team of graduate students and junior analysts and was reviewed by a senior analyst and the authors.

4.4 Waterloo

The dataset consists of 22 projects completed by students in the graduate-level Software Engineering course taught by Dan Berry at the Department of Computer Science, University of Waterloo. The students were given the task of designing a voice-over-IP management software system. For the purpose of this study, we have used two documents from each of the 22 projects: the requirements specification and the use cases description. The requirements specification contained the same core functionality of the system for all groups as well as a number of personalized requirements. The use cases have been designed by students to match the functional requirements found in the specification. An example requirement from the requirement specification of project 1 (of the 22) follows: "F35 The caller will hear a dial tone before placing a call." An example use case from project 1 that addresses this requirement (italics added for emphasis) is shown next:

> UC36 Number: UC36
> Name: Make Call
> Authors: N.B., C.P., S.W., C.A.
> Event: Caller wishes to call callee. Callee's phone number as input.
> Actors: Caller
> Overview: This use case captures the process by which the caller places a call to the callee. The caller picks up the phone, *receives a dial tone*, and then proceeds to dial 4 digits to make the call.
> If the caller's account is cancelled, then they do not receive a dial tone. If the caller's account is suspended, then they receive a dial tone, but cannot call anyone but the administrator.
> If the callee is busy, then the caller receives a busy signal. If the callee is not busy, then the caller hears the ringing signal until the callee picks up the phone, or until 5 min pass and the system drops the connection.

We chose to use the 22 (out of a total of 36) groups whose project submissions included an RTM in softcopy format for the functional requirements-to-use cases trace. We have spot-checked the submitted RTMs, but otherwise used them without change. Each requirements document contained anywhere between 17 and 80 functional requirements, with an average of 48 requirements per document (we ignored non-functional requirements in this experiment as they were not included in the RTMs submitted by students). Each use cases document also contained between 5 and 30 use cases, with about 17 use cases per document. The answer sets contained between 19 and 143 links, with an average of about 57 links per answer set. It was not uncommon for functional requirements in this dataset to go unsatisfied by the use cases.

4.5 Results

We have implemented all the IR methods mentioned in Sect. 2 as a part of a requirements tracing tool called REquirements TRacing On target (RETRO) [16]. Each of the methods is enhanced with user feedback processing. We simulated the analyst vetting the candidate link list and indicating if links are true or false. If the simulation indicates TRUE, the keywords found in that document are given increased value in the query vector (the opposite occurs for FALSE). The IR method is then re-executed using the re-weighted query vectors. For Vector space retrieval, as well as for both keyword extraction approaches (with $x = 30, 50$), we use Standard Rochio Feedback [3] with equal weight assigned to the original query vector, positive feedback, and negative feedback. For LSI, we use the feedback technique from Deerwester et al. [7]. Each experiment is run for eight iterations. On iteration 0, the chosen IR method is run to build the initial list of candidate links. On each subsequent iteration, we simulate the examination of the top two (not yet visited) elements in the list for each high-level element. These elements are checked against the (known to the simulation) actual RTM, and correct feedback is provided.[4] The results of running each experiment were collected and analyzed against

[4] We note that our simulation of analyst feedback concentrates on capturing the effects of the provided feedback, rather than the process. It is hard to imagine an actual analyst working with the system providing feedback in such a regular manner. Research into the actual analyst interaction with traceability tools is another part of our research agenda [18] and is currently underway. It is outside the scope of this paper, however.

existing answer sets. In this paper, we show only the results from iteration 0 (prior to feedback) and iteration 8.

In Tables 1, 2, 3 and 4, we summarize the obtained results for the vector space model with tf-idf weighting, Okapi weighting, and LTU weighting, as well as for LSI for two different vocabulary bases (low-level artifact only and low-level plus high-level artifact), for iteration 0 and iteration 8, respectively. In each table, we show the results obtained for the three datasets: recall, precision, and f2. For example, we can see from Table 1, iteration 0 that vector space with tf-idf weighting for the CM-1 dataset yielded recall of 0.98, precision of 0.02, and f2 of 0.07 for the low-level artifact only and recall of 0.98, precision of 0.02, and f2 of 0.08 for a vocabulary base of low-level plus high-level artifact.

Results for CM-1 and MODIS are shown outright. Note that the LSI table adds a column for the number of dimensions used. For example, we see that LSI with 10 dimensions for the MODIS dataset yielded recall of 0.95, precision of 0.05, and f2 of 0.22 for the low-level artifact only and recall of 0.93, precision of 0.06, and f2 of 0.25 for a vocabulary base of low-level plus high-level artifact. For the Waterloo dataset, we show four results (moved to "Appendix A" for readability, Tables 7, 8 and 9): the average, the median, the maximum, and the minimum of each measure. Note that the measures are computed independently—e.g., maximum recall and maximum precision can be reached on different cases in the dataset. For example, in Table 7, we see that vector space with tf-idf weighting for the first artifact pair of the Waterloo dataset (Waterloo_1) yielded recall of 0.872, precision of 0.09, and f2 of 0.318 for the low-level artifact only and recall of 0.769, precision of 0.411, and f2 of 0.788 for a vocabulary base of low-level plus high-level artifact.

From the tables, we observe the following. All methods exhibit high recall at iteration 0. For the CM-1 dataset, recall remains generally stable, while for the Waterloo dataset recall tended to remain the same for about 40% of the cases, improve for about 50%, and drop slightly for about 10% of the cases. Precision on smaller datasets (MODIS, Waterloo) tends to be between 5 and 15% for individual cases (with the exception of a couple of outliers in the Waterloo dataset) at iteration 0, improving by 8–10% by iteration 8. However, precision is around 2% for CM-1 at iteration 0 and does not improve.

The Waterloo dataset consists of 22 artifact pairs. This afforded us the ability to run statistical analyses of the results obtained when testing a variety of IR methods on this dataset. Table 5 shows the results of the statistical analysis for three methods: vector space retrieval using TF-IDF and LTU weightings and LSI. As mentioned above, Appendix A contains the full set of tables showing the recall, precision, and the f2 measure for all pairs of artifacts tested in the experiments. TF-IDF and LTU methods were applied to all 22 artifact pairs. LSI was applied to 11 of 22 artifact pairs which contained sufficient number of requirements and test cases to warrant the use of LSI (i.e., to allow for non-trivial dimensionality reduction). The LSI method was run for 5, 10, and 15 dimensions for low-level artifact vocabulary base and for 10, 25, and 40 dimension for low-level + high-level vocabulary base. There are two cases of missing data: on two artifact pairs, 15-dimensional matrix for the low-level artifact could not be obtained (due to a small original number of keywords), these rows were removed from consideration.

Table 1 Vocabulary base: MODIS and CM-1 datasets, vector space retrieval using TF-IDF

Vocabulary base	Low			Low + high		
Dataset	Recall	Precision	F2	Recall	Precision	F2
Iteration 0						
CM-1	0.98	0.02	0.07	0.98	0.02	0.08
Modis	0.76	0.08	0.28	0.83	0.50	0.73
Iterarion 8						
CM-1	0.98	0.02	0.08	0.98	0.02	0.08
Modis	0.88	0.18	0.50	0.83	0.50	0.73

Table 2 Vocabulary base: MODIS and CM-1 datasets, vector space retrieval using Okapi

Vocabulary base	Low			Low + high		
Dataset	Recall	Precision	F2	Recall	Precision	F2
Iteration 0						
CM-1	0.98	0.02	0.08	0.98	0.02	0.08
Modis	0.76	0.08	0.28	0.76	0.08	0.28
Iteration 8						
CM-1	0.99	0.02	0.08	0.98	0.02	0.08
Modis	0.88	0.16	0.46	0.80	0.54	0.73

Table 3 Vocabulary base: MODIS and CM-1 datasets, vector space retrieval using LTU

Vocabulary base	Low			Low + high		
Dataset	Recall	Precision	F2	Recall	Precision	F2
Iteration 0						
CM-1	0.98	0.02	0.07	0.98	0.02	0.07
Modis	0.76	0.08	0.28	0.76	0.08	0.28
Iteration 8						
CM-1	0.98	0.02	0.08	0.98	0.02	0.08
Modis	0.88	0.17	0.48	0.85	0.48	0.74

Table 4 Vocabulary base: MODIS and CM-1 datasets, latent semantic indexing

Vocabulary base	Low				Low + high			
Dataset	#Dim	Recall	Precision	F2	#Dim	Recall	Precision	F2
Iteration 0								
CM1	100	0.99	0.01	0.04	200	0.99	0.01	0.04
	200	0.99	0.01	0.04	25	0.99	0.01	0.05
	25	1.00	0.01	0.04	400	0.99	0.01	0.04
Modis	10	0.95	0.05	0.22	10	0.93	0.06	0.25
	25	0.88	0.06	0.23	30	0.76	0.05	0.20
	40	0.85	0.06	0.22	60	0.88	0.06	0.22
Iteration 8								
CM1	100	0.99	0.01	0.05	200	0.99	0.01	0.05
	200	0.99	0.01	0.05	25	0.99	0.01	0.05
	25	0.98	0.01	0.04	400	0.99	0.01	0.05
Modis	10	0.80	0.16	0.45	10	0.85	0.11	0.36
	25	0.80	0.11	0.36	30	0.88	0.12	0.39
	40	0.80	0.14	0.42	60	0.78	0.15	0.42

Table 5 Vocabulary base: Waterloo dataset. Results of paired *t*-test analysis for F2 measures (low-level artifact vocabulary base versus low-level + high-level artifact vocabulary base)

Method, iteration	*N*	Means			SE	*t*-value	*p*-value	95% CI	Pearson
		Low	Low + high	Difference					
TF-IDF, iter. 0	22	0.5263	0.4933	0.033	0.0557	0.59	0.56	(−0.0828, 0.1488)	−0.288
TF-IDF, iter. 8	22	0.4974	0.4506	0.0468	0.0487	0.96	0.348	(−0.0546, 0.1481)	0.009
LTU, iter. 0	22	0.5164	0.5101	0.0063	0.0535	0.12	0.907	(−0.1049, 0.1175)	−0.165
LTU, iter. 8	22	0.4919	0.4843	0.0076	0.0504	0.15	0.882	(−0.0973, 0.1124)	−0.128
LSI, all data, iter. 0	**31**	**0.365**	**0.4098**	**−0.04474**	**0.00826**	**−5.41**	**0.0001**	**(−0.06162, −0.02787)**	**0.939**
LSI, all data, iter. 8	**31**	**0.3428**	**0.3833**	**−0.04047**	**0.00785**	**−5.16**	**0.0001**	**(−0.05650, −0.02444)**	**0.949**
LSI, 10 dim, iter. 0	11	0.3904	0.367	0.0235	0.0111	2.12	0.06	(−0.0012, 0.0481)	**0.976**
LSI, 10 dim, iter. 8	**11**	**0.3208**	**0.352**	**−0.0312**	**0.0104**	**−3.01**	**0.013**	**(−0.0543, −0.0081)**	**0.96**

Table 5 shows the results of two statistical tests performed on the f2 measures computed for the test runs: the paired t-test and the Pearson correlation. The first column lists the method and iteration, the second column lists the number of artifact pairs considered, the third column provides the means for the low artifact vocabulary base, the fourth column provides the means for the low + high artifact vocabulary base, and the fifth column presents the difference (if negative, low only was better than low + high artifact vocabulary base; if positive, the opposite is true). The sixth column of Table 5 presents the standard error, followed by the *t*-value, the eighth column has the *p*-value (alpha = 0.05), the ninth column shows the 95% confidence interval, and finally the last column provides Pearson's correlation. Vector space model with LTU weighting on iteration 0 on 22 artifact pairs had a mean f2 measure of 0.5164 for low only, 0.5101 for low + high vocabulary base, a positive difference of 0.0063 (so low + high was better), a standard error of 0.0535, a *t*-value of 0.12, a *p*-value of 0.907, a confidence interval of (−0.1049, 0.1175), and Pearson's was −0.165. Where the difference between the means of f2 measure of methods was deemed significant by the paired *t*-test, the text is in bold-face. We also applied bold-face to the high-correlation cases (all the LSI cases).

We examined the LSI method with small matrices, medium-sized matrices, and large matrices. In our case, for low-level artifact only this translated to 5, 10, and 15 dimensions. For low + high-level, there were 10, 25, and 40 dimensions (due to an increase in the number of terms in the corpus). Table 5 has some entries labeled «all data». This means that small matrices have been compared to each other (5 dimensions to 10 dimensions as shown in

Table 6 Vocabulary base: pairings of test cases for Waterloo "all data" rows in Table 5

Low-level artifact (Dimensions)	Low-level + high-level artifact (Dimensions)
5	10
10	25
15	40

Table 6), medium to medium, and large to large (15 dimensions to 40 dimensions). A straight comparison of 10 dimensions to 10 dimensions was also made as seen in Tables 5 and 9: «LSI, 10 dim» tests paired the f2 measures reported by both low-level and low-level + high-level artifact vocabulary base for 10 dimensions.

As can be seen from Table 5, the results can be grouped into three distinct categories. There are results that show no significance and no correlation (not bolded). Specifically, the vector space runs with tf-idf and LTU weighting fall into this category. This is interesting because it tells us that using the low-level plus high-level artifact for the vocabulary base does something "different" than when just using the low-level artifact. In fact, this can be observed in the full experimental results found in "Appendix A": for example, of 22 cases reported in Table 7, 11 showed improvement in f2-measure going from low to low + high vocabulary base, while the other 11 showed improvement going in the opposite direction, with differences in the values of f2-measures exceeding 0.1 in most cases. Further investigation is warranted to discover, examine, and characterize "different."

Next are the methods that have statistical significance and high correlation. This describes three of the four LSI runs. Only LSI dimension 10 iteration 0 does not fall into this category. The three LSI runs all indicate that the low-level artifact yields a better mean for f2 than with both low- and high-level artifacts (difference is negative for all three).

The third and final category covers the method runs that showed no significance and high correlation. Only the LSI run left out above falls here. Note that the LSI run of interest (for dimension 10 iteration 0) shows that the low-level + high-level yields a better f2 mean than low-level only, though not statistically significant.

The use of both high- and low-level artifacts for vocabulary base could bring significant payoff, but was risky, as overall it tended to decrease the accuracy of the results (lowered f2 measure, for example). For the MODIS dataset, using both artifacts lead to a significantly better overall result at iteration 8, as evidenced by the f-measure. However, within the Waterloo dataset, there was a wide range of diversity in the results that used both vocabulary bases.

4.6 Vocabulary base: related work

In software engineering, we have the luxury of having the queries ahead of time. This allows us to decide if we want to use the queries to assist in building the corpus or not. In information retrieval, the corpus is built just using the document collection because queries are not known in advance. However, researchers have tried to improve the queries and retrieval results based on the query logs and histories. Baeza-Yates et al. [2] propose a method that, given a query, will recommend similar queries that have been issued in the past. Using this information combined with the user behavior for those past results, the new query results can be improved.

In the information retrieval world, researchers have tried to mine the domain vocabulary to improve the query. Srinivasan et al. [39] used a combination of rough sets and fuzzy sets to create a framework to mine the vocabulary. They also examined the problem of co-coordinating multiple views of the vocabulary.

5 Secondary measures study

As mentioned in Sect. 2, human analysts must be an integral part of the traceability assessment process in the IV&V setting. Our final study concentrated on the importance of secondary measures to assisting in evaluating traceability methods from an analyst's perspective.

Our study included the following steps. High- and low-level elements were parsed from each artifact. The elements were then subjected to stemming [35] and stop word removal. The resulting information was passed to the specific IR method for creation of vectors of term weights.

Next, we simulated the work of an IV&V analyst aided by the selected IR technique. Initially, the IR technique was used to generate candidate links between the artifact levels. Then, perfect analyst feedback was simulated. We examined four different *feedback strategies*: Top 1, Top 2, Top 3, and Top 4. Using strategy *Top i*, the feedback simulator examined (for each high-level requirement) the top *i* *unexamined* candidate links in the list, looked for them in the true RTM or answer set for the dataset, and specified whether each examined link was a true link or a false positive. We chose to emulate *perfect* feedback because no software can be expected to reasonably recover from human judgment errors. At the same time, we want to investigate which IR methods are most receptive to *correct* user feedback.

The information collected via the process described above was encoded in XML and was passed to the feedback processor, which updated the query vectors and passed control back to the IR method for the next iteration.

We ran eight iterations for each IR technique. Our analysis tool was used to compare the actual results (the answer set or true RTM) to the results obtained by the IR method (returned candidate link lists) for every iteration. The IR methods were implemented in our tool REquirements TRacing On target (RETRO) [14]. The resulting information was used to calculate primary and secondary measures for evaluation. Measures were then plotted to assist in analysis.

At each iteration of the tracing process, in addition to considering the full list of candidates returned by a specific method (i.e., the list of (d, h) pairs with $sim(d, h) > 0$), we also consider *filtered lists*. Given a filter value $\alpha > 0$, the filtered list L_h^a consists of all links (d, h) such that $sim(d, h) \geq \alpha$. In our experiments, α was taken to be equal to 0.05, 0.1, 0.15, 0.2, and 0.25.

5.1 Examples

We present a number of examples discovered during the analysis of the results from our experiments that illustrate the importance of the secondary measures. In all examples, we compare two separate runs of the experiment side-by-side. We then show that the picture painted by the precision and recall numbers (in terms of which run produced "better" results) needs to be altered, as demonstrated by the secondary measures. In some examples, secondary measures serve as "tie-breakers," allowing us to choose the better of the two runs. In other examples, the information provided by the secondary measures bridges the gap between our assessments of the two runs.

The examples contain two types of graphs. We use recall-versus-precision trajectories to plot the behavior of the primary measures over the course of the feedback process. Each point represents a (precision, recall) pair after some iteration (0, 1,…,8) of the feedback loop. The lines connect the neighboring iterations. For secondary measures, we simply plot the value of the measure at each iteration.

5.1.1 Example 1

The graph in Fig. 2 shows the recall-versus-precision trajectories for tf-idf method with and without a thesaurus running on MODIS dataset with Top 2 feedback and filtered at $\alpha = 0.05$. Based on primary measures alone, it appears that the tf-idf + thesaurus technique is better in terms of recall and would be selected over the simple tf-idf method.

Next, we look at the secondary measure of Lag for the same scenario (Fig. 3). The graph shows that in iterations four through six, the non-thesaurus technique achieves total separation between true links and false positives (Lag of 0)

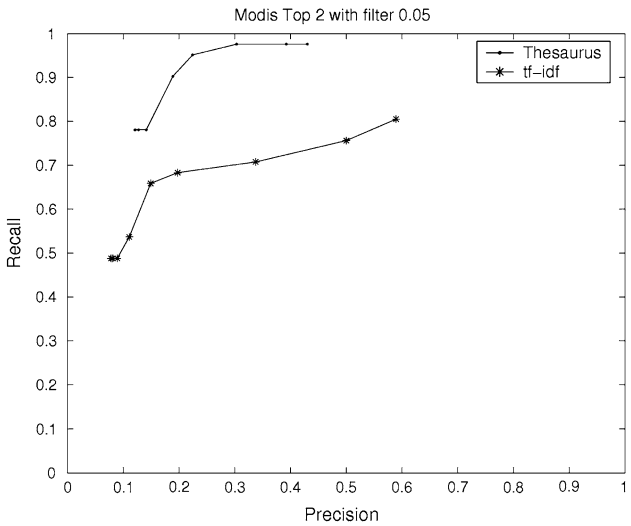

Fig. 2 Recall and precision for MODIS dataset, Top 2 feedback, filter 0.05, tf-idf plus Thesaurus versus tf-idf (No thesaurus)

much sooner than the thesaurus technique. From an analyst's perspective, the non-thesaurus method may be preferable even at a reduction in recall, because the top portions of all candidate link lists will contain (almost) exclusively relevant matches sooner (in less iterations).

5.1.2 Example 2

The graph in Fig. 4 compares the recall-versus-precision trajectories obtained in our experiments for LSI and tf-idf methods using Top 2 feedback and no filtering on the MODIS dataset. The trajectories are close to each other, with LSI showing somewhat better recall, while tf-idf

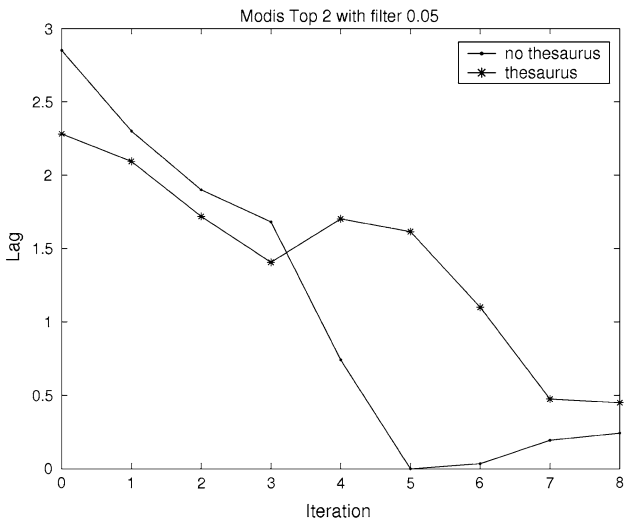

Fig. 3 Lag for MODIS dataset, Top 2 feedback, filter 0.05, Thesaurus versus No thesaurus

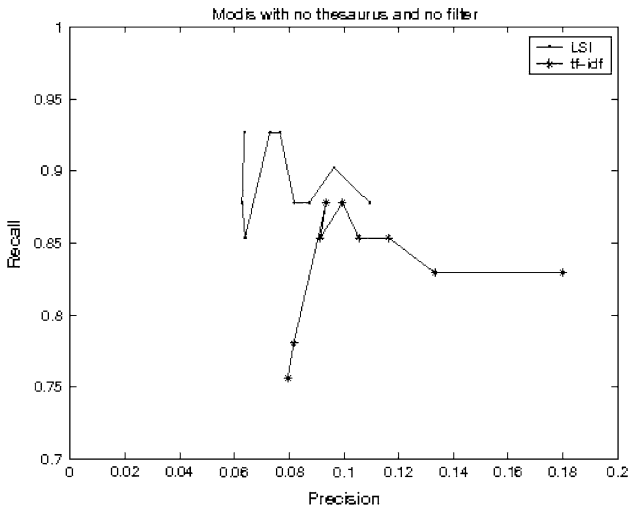

Fig. 4 Recall and precision for MODIS dataset, Top 2 feedback, no filter, LSI versus tf-idf

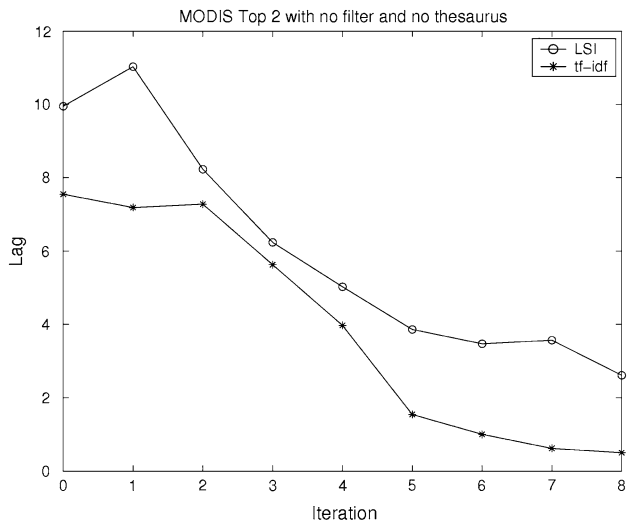

Fig. 5 Lag for MODIS dataset, Top 2 feedback, no filter, LSI versus tf-idf

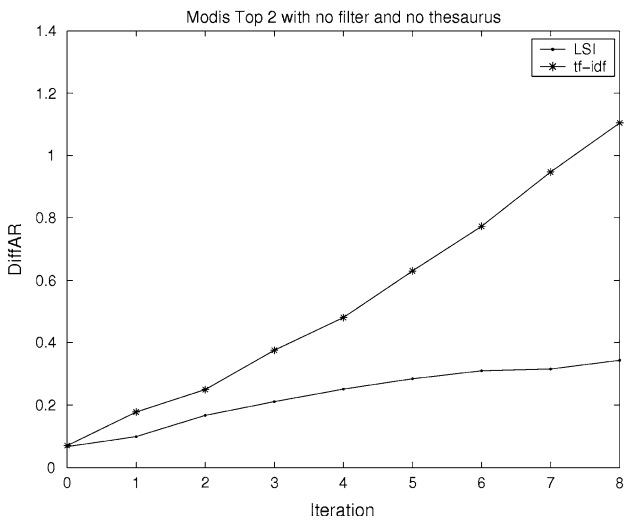

Fig. 6 DiffAR for MODIS dataset, Top 2 feedback, no filter, LSI versus tfi-df

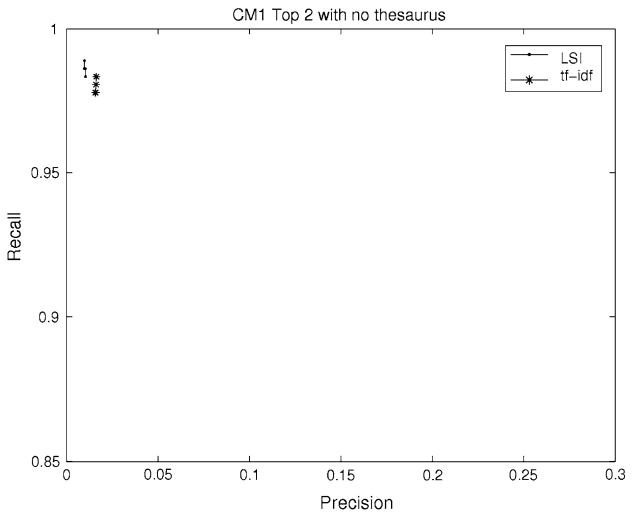

Fig. 7 Recall and precision for CM1 dataset, Top 2 feedback, no filter, LSI versus tf-idf

eventually moves toward better precision (over 18%). It is not very clear which technique is better. Unlike Example 1, where precision of both methods was quite high, the precision for LSI is significantly lower in this example.

Now, we look at the secondary measure of Lag for the same scenario, shown in Fig. 5. Both methods show similar trends in reducing Lag. However, we see that tf-idf reduces Lag to a much lower number (less than 1), while the Lag for LSI remains above 2 after iteration 8. This suggests that tf-idf is much more successful in separating the true links from false positives in candidate link lists during the feedback process.

This supposition receives even more support upon examination of the DiffAR trends shown in Fig. 6: DiffAR for tf-idf shows a huge improvement over DiffAR for LSI.[5] Based on the secondary measures, an analyst would prefer tf-idf, even though recall is somewhat lower than for LSI.

5.1.3 *Example 3*

The graph in Fig. 7 compares the recall-versus-precision trajectories for LSI and tf-idf methods on the CM-1 dataset using Top 2 feedback and no filtering. The graph shows almost no change in precision (and precision itself is unacceptably low) and only a slight change in recall for

[5] In this graph, DiffAR grows to a number above 1. This is because during the feedback process, the similarity between two requirements can exceed 1, as well as become negative.

both methods (these runs show that IR methods are not always very effective by themselves). However, in prior work [20], we include graphs showing that, in these runs, filtering improves precision significantly without hurting recall significantly. It is not clear that one technique outperforms the other in any significant way.

Next, we look at the secondary measure of Lag for the same scenario, in Fig. 8. For LSI, Lag drops from 8.3 to just above 6. But for tf-idf, Lag drops from 6 to 4.5. While both Lags start and end fairly large (our preference is for Lag to fall down to the 1–2 range), it is clear that tf-idf outperformed LSI, thus providing us with a clear tie-breaker. Again, consideration of a secondary measure changes the scenario assessment.

5.1.4 Example 4

Our last example looks at two runs over different datasets. The graph in Fig. 9 compares the recall-versus-precision trajectories for tf-idf method with Top 2 feedback on the MODIS (no filtering) and CM-1 (filter set to 0.1) datasets. We note that these two runs are very different—one is on a small dataset and another is on a large dataset. However, the precision-versus-recall trajectories of the runs look similar; in fact, they follow the same path on the last few iterations.

Next, we look at the secondary measure of Lag for the same scenario (in Fig. 10). There is a clear difference in the behavior of Lag for the MODIS and CM-1 cases. Lag for MODIS starts very high, at 7.5, and does not drop to an acceptable range until iteration 5. Once there, however, it outperforms the Lag of CM-1, which over the course of 8 iterations shows slow but steady decline from about 2.2 to 1.1.

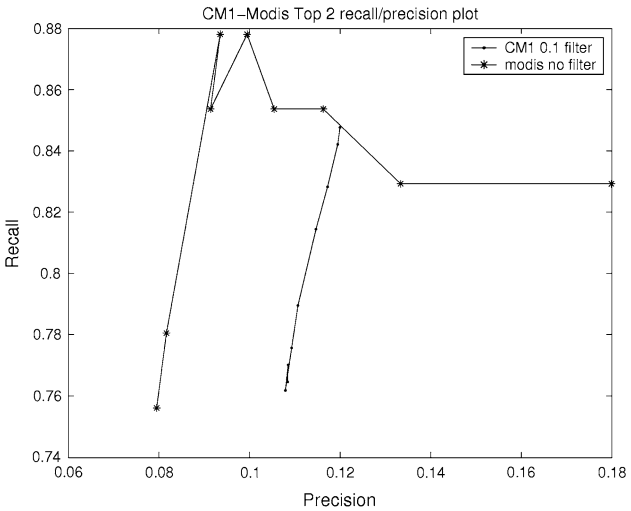

Fig. 9 Recall and precision, tf-idf, Top 2 feedback, CM1 filter 0.1 versus MODIS no filter

Selectivity, plotted in Fig. 11, also highlights the differences between these two runs. It shows a much better selectivity for the CM-1 dataset, which remains steady throughout the iterations. At the same time, selectivity for the MODIS run starts at around 0.41 and steadily improves to just over 0.2. How do we interpret this? Comparing these two runs for the purpose of determining which one was better is not very meaningful, as they relate to different datasets. However, we may notice that for all the differences between the runs, precision and recall do not distinguish between them, while our secondary measures, Lag and Selectivity, uncover the differences.

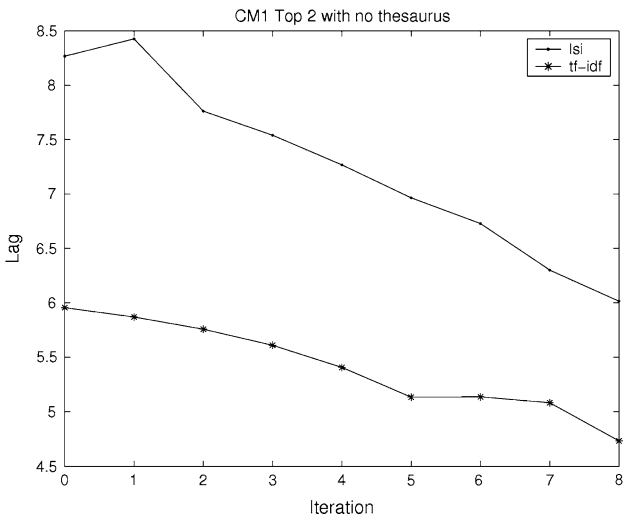

Fig. 8 Lag for CM1 dataset, Top 2 feedback, no filter, LSI versus tf-idf

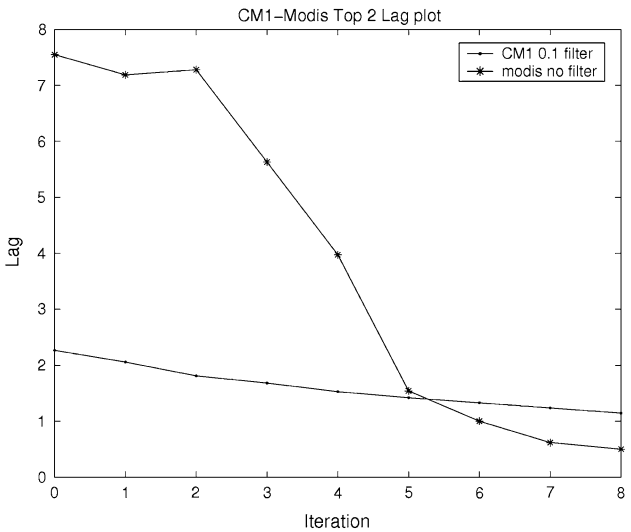

Fig. 10 Lag, tf-idf, Top 2 feedback, CM1 filter 0.1 versus MODIS no filter

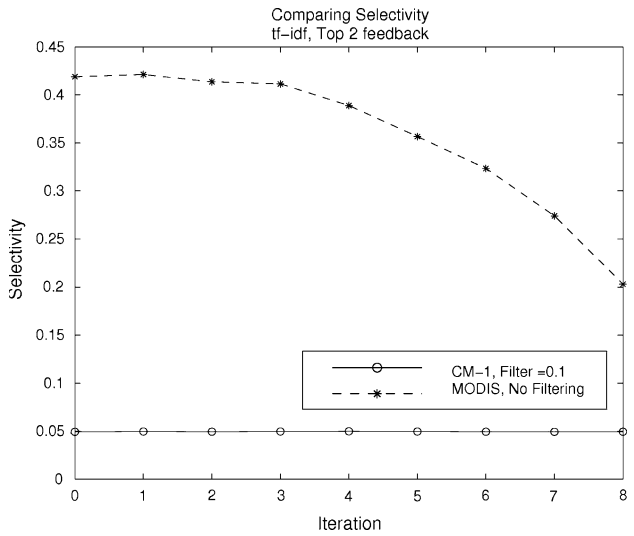

Fig. 11 Selectivity: tf-idf, Top 2 feedback, CM1 filter 0.1 versus MODIS no filter

5.2 Conclusions

The examples shown above illustrate some of the situations we encountered during our tracing experiments where the use of secondary measures either changed our perceptions about the results outright or provided us with the ability to distinguish between the quality of otherwise similar test runs. In person-power intensive requirements tasks such as tracing, we need reliable ways to assess a technique's effectiveness from the analyst's perspective. We feel that these examples provide support for and evidence of the ability of secondary measures to assist with such assessments.

Additional comments can be made concerning one of the issues with the state-of-the-art in automating traceability work that has been illustrated in the examples described above: the low precision of the candidate RTMs obtained in experiments. Indeed, in many experiments described in our work [20] and the work of other research groups [1, 4, 28], candidate RTMs generated by automated methods had rather high recall (80% and above), but low, by IR standards, precision (in single or low double digits). This, and similar situations in the use of data mining techniques to build predictive models in Software Engineering [31], has led to a vigorous discussion [31, 32, 44] on the topic.

Zhang and Zhang [44] argue that predictive models with low precision are useless in software engineering. Menzies et al. [32], writing in response to Zhang and Zhang [44], argue that low precision alone is not sufficient to declare failure.

As discussed in Sect. 3.2.4, *selectivity* can be used together with *recall* to evaluate the quality of a candidate RTM much in the same way that *precision* and *recall* are used in IR. Indeed, given a specific candidate RTM,[6] *a human analyst needs to examine it in its entirety regardless of how high the precision.* Therefore, in the presence of high *recall,* the size of the candidate RTM is a good enough estimator of the human effort needed to complete the RTM generation task, and, perhaps, an even better estimate than *precision*.

In general, we would like to see higher precision candidate RTMs (at fixed high *recall* levels) as this means that the size of the candidate RTM decreases. However, as long as the savings from generating a candidate RTM using an automated method as measured by *selectivity* are significant, we maintain that such automated methods have practical uses. For further discussion of this topic, we refer the reader to Zhang and Zhang [44] for the critique of low-precision methods and to Menzies et al. [32] for defense of such methods.

5.3 Secondary measures: related work

A number of other fields are using secondary measures in their evaluations. In the area of performance assessment, Le et al. [25] examined the effects of active queue management on response time experienced by web users. They found it necessary to use packet loss rates and link utilization as secondary measures to the primary measure of user perceived response time. Vincent et al. [42] modified existing multi-agent technologies to provide distributed control for a real-time environment. They found it necessary to use additional secondary measures because the measure hard scheduling deadline (in seconds) did not provide an appropriate grain size. Haritsa et al. [12] examined the parameters of a real-time database system that have significant impact on the performance of concurrency control algorithms. Their primary measure examines hard deadlines. A secondary measure, used when considering soft deadlines, measures the average time by which transactions miss their deadlines.

Hayes et al. [15] were able to achieve recall of 85% at precision of 40% on a small dataset using tf-idf + thesaurus, no secondary measures were collected. Hayes et al. [16] found the secondary measures of Lag, DiffAR, and selectivity to be useful in evaluating the effectiveness of tf-idf and tf-idf plus thesaurus with user feedback on a small dataset. Antoniol et al. [1] examined traceability of requirements to code using two IR techniques (tf-idf and probabilistic IR). They measured recall plus precision, achieving 100% recall at 13.8% precision for one dataset.

[6] We assume here that the RTM was produced as part of the IV&V process and thus, as stated in Sects. 1 and 2, requires validation by a human analyst.

Marcus and Maletic [28] applied latent semantic indexing (LSI) to the same datasets as Antoniol et al., again using recall plus precision to evaluate. They achieved 93.5% recall plus 54% precision for one dataset. Note that neither of these works used secondary measures.

6 Conclusions and future work

We undertook an examination of two areas that can be used to enhance traceability for software engineering. Each of the studies is discussed below.

6.1 Vocabulary base

In general, it was found that accuracy decreased when the vocabulary base was generated from both the high- and low-level artifact and that much stabler results were achieved when the vocabulary base considered only the low-level artifact. In the case of the tf-idf method with no filter, the difference in the vocabulary base did not seem to impact the results. However, the recall and precision values were slightly better for some of the filter values when both the documents and the queries were used to build the vocabulary base. This means that the new vocabulary base did not identify any new true links.

These results lend preliminary evidence that it may not be beneficial to use both artifacts as the vocabulary base. The results are not conclusive, however. This is not particularly surprising as we only examined three datasets. We plan to examine this idea again when additional datasets are available for experimentation. Also, we plan to investigate the "different" results for low + high vocabulary base for vector space with tf-idf and LTU weighting—we want to understand why the low + high behaved differently than low artifact only.

6.2 Secondary measures

We applied IR methods with relevance feedback to the problem of tracing textual artifacts and demonstrated that, in certain cases, secondary measures significantly affect the analyst's perception of the quality of the results. Thus, the secondary measures we studied prove to be an important asset in our quest to evaluate different methods for tracing requirements. We found support for the use of secondary measures. Specifically, we found numerous examples where the assessment of the results of a trace given primary measures was very different from the result assessment using secondary measures.

While the results of the study are encouraging, they also show clear avenues for improvement. Among them we identify the following:

1. Study of the work of analysts in requirements tracing, and
2. Study of the applicability of secondary measures to other retrieval activities in requirements and software engineering.

We note that the current study was an objective evaluation of the quality of results produced by the IR and relevance feedback algorithms. In practice, however, it will be up to human analysts to supply relevance feedback, and as such, it is impossible to envision analysts to be 100% correct in their decisions. Therefore, in order to make a tracing tool useful for analysts, we need to study how they tend to work with the candidate link lists produced by the software.

Acknowledgments Our work is funded by NASA under grant NAG5-11732 and partially funded by the National Science Foundation under NSF grant CCF-0811140. We thank Stephanie Ferguson, Ken McGill, and Tim Menzies. We thank Olga Dekhtyar for her assistance with statistical analysis. We thank Dan Berry and his students at the University of Waterloo. We also thank Sarah Howard, Sravanthi Vadlamudi, and James Osborne who worked on early versions of the software used for the evaluation. We also thank the anonymous reviewers of this paper for their valuable suggestions during the revision process.

Appendix A: full results from the Waterloo dataset

For the sake of completeness, we chose to include in this Appendix the full results of our experiments run on the Waterloo dataset. As mentioned above, we studied 22 pairs of artifacts (functional requirements, test cases). Each pair came with an RTM, which was used as the answer set when analyzing the results of the IR methods. The 22 pairs of artifacts are labeled Waterloo_1,…, Waterloo_22 in the tables below. As seen from the results, there was a clear variability of results on different pairs of artifacts. Some pairs (e.g., Waterloo_16) proved to be easier to trace than others (e.g., Waterloo_6). For each table, we present the following summary information:

- The row with the best value of the *f2-measure* (column F2) achieved. This row is highlighted in bold in the table and is reported separately at the bottom.
- The maximum, minimum, average, and median value of each parameter achieved. These are reported independently (i.e., the MAX value for Precision and the MAX value for Recall reported at the bottom in the same line may come from different artifact pairs).

See Tables 7, 8 and 9

Table 7 Vocabulary base: Waterloo dataset, vector space retrieval using TF-IDF

Vocabulary base	Low			Low + high		
Artifact Pair	Recall	Precision	F2	Recall	Precision	F2
Iteration 0						
Waterloo_1	0.872	0.090	0.318	**0.769**	**0.411**	**0.788**
Waterloo_2	0.824	0.467	0.714	0.559	0.826	0.597
Waterloo_3	0.954	0.082	0.305	0.908	0.274	0.621
Waterloo_4	0.978	0.099	0.351	0.956	0.253	0.614
Waterloo_5	0.938	0.113	0.381	0.875	0.255	0.588
Waterloo_6	0.477	0.071	0.223	0.534	0.245	0.432
Waterloo_7	0.566	0.833	0.605	0.509	0.844	0.553
Waterloo_8	0.439	0.193	0.350	0.182	0.545	0.210
Waterloo_9	0.648	0.275	0.510	0.197	0.519	0.225
Waterloo_10	0.784	0.674	0.759	0.378	0.583	0.407
Waterloo_11	0.733	0.667	0.719	0.300	0.600	0.333
Waterloo_12	0.882	0.147	0.441	0.804	0.298	0.600
Waterloo_13	0.627	0.561	0.613	0.118	1.000	0.143
Waterloo_14	0.706	0.471	0.642	0.588	0.513	0.571
Waterloo_15	0.841	0.311	0.627	0.580	0.302	0.489
Waterloo_16	**0.965**	**0.517**	**0.822**	0.848	0.436	0.411
Waterloo_17	0.780	0.372	0.640	0.756	0.158	0.431
Waterloo_18	0.900	0.191	0.517	0.767	0.299	0.584
Waterloo_19	0.804	0.127	0.389	0.714	0.253	0.524
Waterloo_20	0.833	0.179	0.481	0.659	0.696	0.666
Waterloo_21	0.841	0.366	0.668	0.429	0.692	0.464
Waterloo_22	0.789	0.205	0.503	0.632	0.500	0.600
Waterloo_best_F2	**0.965**	**0.517**	**0.822**	**0.769**	**0.411**	**0.788**
Waterloo-average	**0.781**	**0.319**	**0.526**	**0.594**	**0.477**	**0.493**
Waterloo-median	**0.814**	**0.240**	**0.514**	**0.610**	**0.468**	**0.538**
Waterloo-Max	**0.978**	**0.833**	**0.822**	**0.956**	**1.000**	**0.788**
Waterloo-Min	**0.439**	**0.071**	**0.223**	**0.118**	**0.158**	**0.143**
Iteration 8						
Waterloo_1	0.872	0.077	0.284	*0.077*	*0.429*	*0.092*
Waterloo_2	0.824	0.400	0.680	0.559	0.514	0.549
Waterloo_3	0.954	0.074	0.282	0.908	0.237	0.580
Waterloo_4	0.978	0.083	0.309	0.956	0.209	0.557
Waterloo_5	0.938	0.094	0.336	**0.656**	**0.778**	**0.677**
Waterloo_6	0.477	0.067	0.215	0.534	0.192	0.394
Waterloo_7	0.547	0.707	0.573	0.509	0.750	0.544
Waterloo_8	0.439	0.179	0.340	0.182	1.000	0.217
Waterloo_9	0.648	0.263	0.501	0.197	0.500	0.224
Waterloo_10	0.784	0.518	0.711	0.378	0.700	0.417
Waterloo_11	0.733	0.688	0.724	0.300	0.818	0.344
Waterloo_12	0.863	0.133	0.412	0.794	0.245	0.549
Waterloo_13	0.627	0.451	0.582	0.118	0.857	0.142
Waterloo_14	0.706	0.462	0.638	0.588	0.667	0.602
Waterloo_15	0.841	0.279	0.600	0.580	0.290	0.483
Waterloo_16	**0.965**	**0.454**	**0.788**	0.375	0.771	0.418
Waterloo_17	0.780	0.283	0.578	0.756	0.158	0.431
Waterloo_18	0.900	0.176	0.495	0.767	0.256	0.548

Table 7 continued

Vocabulary base	Low			Low + high		
Artifact Pair	Recall	Precision	F2	Recall	Precision	F2
Waterloo_19	0.804	0.107	0.348	0.714	0.202	0.474
Waterloo_20	0.841	0.164	0.461	0.652	0.694	0.660
Waterloo_21	0.857	0.293	0.619	0.429	0.794	0.472
Waterloo_22	0.789	0.176	0.466	0.632	0.343	0.541
Waterloo_best_F2	**0.965**	**0.454**	**0.788**	**0.656**	**0.778**	**0.677**
Waterloo-average	**0.780**	**0.279**	**0.497**	**0.530**	**0.518**	**0.451**
Waterloo-median	**0.814**	**0.221**	**0.498**	**0.569**	**0.507**	**0.478**
Waterloo-Max	**0.978**	**0.707**	**0.788**	**0.956**	**1.000**	**0.677**
Waterloo-Min	**0.439**	**0.067**	**0.215**	**0.077**	**0.158**	**0.092**

Table 8 Vocabulary base: Waterloo dataset, vector space retrieval using LTU

Vocabulary base	Low			Low + high		
Artifact Pair	Recall	Precision	F2	Recall	Precision	F2
Iteration 0						
Waterloo_1	0.872	0.081	0.294	0.769	0.349	0.620
Waterloo_2	0.853	0.439	0.718	0.559	0.864	0.601
Waterloo_3	0.954	0.085	0.313	0.908	0.309	0.654
Waterloo_4	0.956	0.049	0.202	0.956	0.254	0.616
Waterloo_5	0.938	0.108	0.370	0.875	0.214	0.541
Waterloo_6	0.477	0.073	0.227	0.523	0.208	0.401
Waterloo_7	0.528	0.824	0.569	0.528	0.824	0.569
Waterloo_8	0.439	0.190	0.348	0.212	0.824	0.249
Waterloo_9	0.648	0.288	0.518	0.211	0.441	0.236
Waterloo_10	0.784	0.690	0.763	0.378	0.700	0.417
Waterloo_11	0.733	0.733	0.733	0.333	1.000	0.385
Waterloo_12	0.882	0.139	0.427	0.833	0.309	0.622
Waterloo_13	0.627	0.571	0.615	0.118	0.462	0.138
Waterloo_14	0.706	0.471	0.642	0.588	0.625	0.595
Waterloo_15	0.841	0.287	0.607	0.580	0.280	0.478
Waterloo_16	**0.972**	**0.528**	**0.832**	0.403	0.784	0.446
Waterloo_17	0.780	0.368	0.637	0.756	0.419	0.651
Waterloo_18	0.900	0.190	0.515	0.800	0.353	0.638
Waterloo_19	0.804	0.131	0.397	0.732	0.273	0.548
Waterloo_20	0.841	0.178	0.481	**0.644**	**0.714**	**0.657**
Waterloo_21	0.825	0.364	0.658	0.524	0.767	0.559
Waterloo_22	0.789	0.197	0.493	0.632	0.500	0.600
Waterloo_best_F2	**0.972**	**0.528**	**0.832**	**0.644**	**0.714**	**0.657**
Waterloo-average	**0.780**	**0.317**	**0.516**	**0.585**	**0.521**	**0.510**
Waterloo-median	**0.814**	**0.242**	**0.517**	**0.584**	**0.451**	**0.564**
Waterloo-Max	**0.972**	**0.824**	**0.832**	**0.956**	**1.000**	**0.657**
Waterloo-Min	**0.439**	**0.049**	**0.202**	**0.118**	**0.208**	**0.138**
Iteration 8						
Waterloo_1	0.872	0.072	0.270	0.769	0.333	0.610
Waterloo_2	0.853	0.358	0.668	0.559	0.559	0.559
Waterloo_3	0.954	0.078	0.294	0.908	0.257	0.602

Table 8 continued

Vocabulary base	Low			Low + high		
Artifact Pair	Recall	Precision	F2	Recall	Precision	F2
Waterloo_4	0.956	0.049	0.202	0.956	0.195	0.538
Waterloo_5	0.938	0.091	0.328	0.875	0.188	0.505
Waterloo_6	0.477	0.068	0.217	0.511	0.160	0.355
Waterloo_7	0.547	0.806	0.585	0.509	0.844	0.553
Waterloo_8	0.439	0.181	0.342	0.212	0.452	0.237
Waterloo_9	0.648	0.253	0.494	0.211	0.556	0.241
Waterloo_10	0.784	0.580	0.732	0.378	0.560	0.405
Waterloo_11	0.733	0.733	0.733	0.333	0.588	0.365
Waterloo_12	0.882	0.130	0.408	0.824	0.263	0.577
Waterloo_13	0.627	0.444	0.580	0.118	1.000	0.143
Waterloo_14	0.706	0.429	0.625	0.588	0.345	0.515
Waterloo_15	0.841	0.287	0.607	0.580	0.251	0.459
Waterloo_16	**0.965**	**0.466**	**0.795**	0.396	0.770	0.438
Waterloo_17	0.780	0.278	0.573	**0.756**	**0.443**	**0.662**
Waterloo_18	0.900	0.178	0.496	0.800	0.293	0.594
Waterloo_19	0.804	0.111	0.357	0.732	0.227	0.506
Waterloo_20	0.841	0.159	0.453	0.629	0.697	0.641
Waterloo_21	0.825	0.281	0.595	0.524	0.767	0.559
Waterloo_22	0.789	0.179	0.469	0.632	0.462	0.588
Waterloo_best_F2	**0.965**	**0.466**	**0.795**	**0.756**	**0.443**	**0.662**
Waterloo-average	**0.780**	**0.282**	**0.492**	**0.582**	**0.464**	**0.484**
Waterloo-median	**0.814**	**0.217**	**0.495**	**0.584**	**0.447**	**0.526**
Waterloo-Max	**0.965**	**0.806**	**0.795**	**0.956**	**1.000**	**0.662**
Waterloo-Min	**0.439**	**0.049**	**0.202**	**0.118**	**0.160**	**0.143**

Table 9 Vocabulary Base. Waterloo Dataset (truncated), Latent Semantic Indexing

Vocabulary base	Low				Low + high			
Artifact pair	#Dims	Recall	Precision	F2	#Dims	Recall	Precision	F2
Iteration 0								
Waterloo_1	5	0.795	0.064	0.242	10	0.872	0.069	0.261
	10	0.872	0.069	0.262	25	0.846	0.080	0.289
	15	0.846	0.078	0.286	40	0.872	0.088	0.314
Waterloo_2	5	0.824	0.149	0.432	10	0.824	0.222	0.534
	10	0.794	0.284	0.584	25	0.853	0.426	0.711
	15	N/A	N/A	N/A	40	0.824	0.438	0.700
Waterloo_3	5	0.969	0.052	0.216	10	0.985	0.063	0.250
	10	0.969	0.062	0.248	25	0.985	0.076	0.291
	15	0.954	0.065	0.257	40	0.954	0.078	0.294
Waterloo_4	5	0.733	0.053	0.207	10	0.956	0.063	0.250
	10	0.933	0.055	0.222	25	0.956	0.084	0.311
	15	0.978	0.059	0.236	40	0.978	0.095	0.343
Waterloo_5	5	0.938	0.079	0.296	10	0.938	0.077	0.288
	10	0.938	0.091	0.329	25	0.938	0.106	0.365
	15	0.938	0.096	0.341	40	0.938	0.108	0.369
Waterloo_12	5	0.912	0.091	0.325	10	0.902	0.105	0.359

Table 9 continued

Vocabulary base	Low				Low + high			
Artifact pair	#Dims	Recall	Precision	F2	#Dims	Recall	Precision	F2
	10	0.922	0.101	0.350	25	0.843	0.121	0.384
	15	0.912	0.114	0.381	40	0.922	0.138	0.431
Waterloo_16	5	0.840	0.248	0.569	10	0.917	0.261	0.610
	10	0.924	0.335	0.683	25	0.951	0.371	0.725
	15	N/A	N/A	N/A	40	0.979	0.452	0.794
Waterloo_18	5	0.833	0.095	0.326	10	1.000	0.122	0.410
	10	0.967	0.127	0.417	25	0.900	0.189	0.513
	15	0.900	0.158	0.464	40	0.900	0.193	0.659
Waterloo_19	5	0.893	0.083	0.302	10	0.857	0.081	0.293
	10	0.839	0.090	0.315	25	0.839	0.101	0.341
	15	0.911	0.098	0.343	40	0.804	0.122	0.379
Waterloo_20	5	0.879	0.134	0.417	10	0.909	0.142	0.436
	10	0.909	0.147	0.446	25	0.902	0.159	0.467
	15	0.864	0.170	0.476	40	0.902	0.173	0.490
Waterloo_22	5	0.789	0.130	0.393	10	0.842	0.103	0.345
	10	0.842	0.150	0.437	25	0.842	0.178	0.482
	15	0.789	0.214	0.514	40	0.789	0.211	0.510
Iteration 8								
Waterloo_1	5	0.769	0.054	0.210	10	0.872	0.063	0.245
	10	0.872	0.063	0.246	25	0.846	0.069	0.261
	15	0.846	0.069	0.261	40	0.872	0.076	0.282
Waterloo_2	5	0.824	0.137	0.412	10	0.824	0.197	0.504
	10	0.794	0.257	0.560	25	0.853	0.377	0.681
	15	N/A	N/A	N/A	40	0.824	0.418	0.690
Waterloo_3	5	0.969	0.051	0.209	10	0.985	0.061	0.243
	10	0.969	0.060	0.241	25	0.985	0.069	0.269
	15	0.954	0.062	0.245	40	0.954	0.071	0.273
Waterloo_4	5	0.733	0.050	0.197	10	0.978	0.061	0.243
	10	0.933	0.053	0.216	25	0.956	0.073	0.280
	15	0.978	0.056	0.227	40	0.978	0.079	0.298
Waterloo_5	5	0.938	0.075	0.285	10	0.938	0.071	0.273
	10	0.938	0.085	0.312	25	0.938	0.087	0.316
	15	0.938	0.083	0.307	40	0.938	0.092	0.330
Waterloo_12	5	0.922	0.089	0.320	10	0.902	0.099	0.344
	10	0.922	0.097	0.341	25	0.833	0.111	0.362
	15	0.902	0.107	0.363	40	0.922	0.127	0.409
Waterloo_16	5	0.847	0.228	0.550	10	0.931	0.253	0.606
	10	0.910	0.295	0.642	25	0.944	0.343	0.700
	15	N/A	N/A	N/A	40	0.958	0.380	0.735
Waterloo_18	5	0.833	0.092	0.319	10	1.000	0.114	0.391
	10	0.967	0.116	0.393	25	0.900	0.168	0.480
	15	0.900	0.140	0.431	40	0.900	0.176	0.638
Waterloo_19	5	0.911	0.078	0.289	10	0.857	0.076	0.282
	10	0.839	0.082	0.294	25	0.839	0.090	0.314
	15	0.911	0.087	0.314	40	0.804	0.100	0.334
Waterloo_20	5	0.871	0.127	0.400	10	0.886	0.134	0.417
	10	0.909	0.135	0.424	25	0.871	0.147	0.439

Table 9 continued

Vocabulary base	Low				Low + high			
Artifact pair	#Dims	Recall	Precision	F2	#Dims	Recall	Precision	F2
	15	0.864	0.147	0.437	40	0.902	0.156	0.460
Waterloo_22	5	0.789	0.103	0.338	10	0.789	0.097	0.325
	10	0.842	0.128	0.398	25	0.842	0.145	0.430
	15	0.789	0.163	0.446	40	0.789	0.167	0.452

References

1. Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E (2002) Recovering traceability links between code and documentation. IEEE Trans Software Eng 28(10):970–983
2. Baeza-Yates R, Hurtado C, Mendoza M (2004) Query recommendation using query logs in search engines. In: Proceedings, international workshop on clustering information over the web (ClustWeb)
3. Baeza-Yates R, Ribeiro-Neto B (1999) Modern information retrieval. Addison-Wesley, New York
4. Cleland-Huang J, Settimi R, Khadra OB (2005) Eugenia berezhanskaya, selvia christina: goal-centric traceability for managing non-functional requirements. ICSE, pp 362–371
5. Cleland-Huang J, Hayes JH, Domel J (2009) Model based Traceability. In: Proceedings of traceability of emerging forms of software engineering (TEFSE), an ICSE workshop
6. Dekhtyar A, Hayes JH, Sundaram S, Holbrook A, Dekhtyar O (2007) Technique integration for requirements assessment. In: Proceedings of the IEEE Int'l conference on requirements engineering
7. Deerwester S, Dumais ST, Landauer TK, Furnas GW, Harshman RA (1990) Indexing by latent semantic analysis. Journal of the Society for Information Science 41(6):391–407
8. Dömges R, Pohl K (1998) Adapting traceability environments to project-specific needs. Communications of the ACM 41:54–62
9. Ebner G, Kaindl H (2002) Tracing all around in reengineering. IEEE Software 19(3):70–77
10. Egyed A, Gruenbacher P (2002) Automatic requirements traceability: beyond the record and replay paradigm. In: Proceedings of the 17th IEEE international conference on automated software engineering (ASE), Edinburgh, UK
11. Egyed A (2003) A scenario-driven approach to trace dependency analysis. IEEE Transactions on Software Engineering 9(2):116–132
12. Haritsa J, Carey M, Livny M (1990) On being optimistic about real-time constraints, April 1990
13. Hayes JH, Dekhtyar A, Sundaram S (2005) Improving after the fact tracing and mapping to support software quality predictions. IEEE Software 22:30–37
14. Hayes JH, Dekhtyar A, Sundaram S, Holbrook A, Vadlamudi S, April A (2007) REquirements tracing on target (RETRO): improving software maintenance through traceability recovery. Innovations in Systems and Software Engineering: A NASA Journal 3(3):193–202
15. Hayes JH, Dekhtyar A, Osbourne J (2003) Improving requirements tracing via information retrieval. In: International conference on requirements engineering (RE'03). Monterey, California, pp 138–148
16. Hayes JH, Dekhtyar A, Sundaram S, Howard S (2004) Helping analysts trace requirements: an objective look. In: Proceedings of IEEE requirements engineering conference (RE). Kyoto, Japan, pp 249–261
17. Hayes JH (1990) Risk reduction through requirements tracing. In: The conference proceedings of software quality Week 1990. San Francisco, CA, pp 1–25
18. Hayes JH, Dekhtyar A, Sundaram S (2005) Humans in the traceability loop: can't live with 'em, can't live without 'em. In: Proceedings of the workshop on traceability of emerging forms of software engineering (TEFSE). Long Beach, CA, pp 20–23
19. Hayes JH, Dekhtyar A, Sundaram S (2006) Advances in dynamic generation of traceability links: two steps closer to full automation? University of Kentucky tech report, (TR 451-06)
20. Hayes JH, Dekhtyar A, Sundaram S (2006) Advancing candidate link generation for requirements tracing: the study of methods. IEEE Transactions on Software Engineering 32(1):4–19
21. Huang JC, Dekhtyar A, Hayes JH, Antoniol G, Berenbach B, Eyged A, Ferguson S, Maletic J, Zisman A (eds) (2006) Grand challenges in traceability, center of excellence for traceability technical report COET-GCT-06-01. Sept 2006. http://www.traceabilitycenter.org/files/COET-GCT-06-01-0.9.pdf
22. Integrated Chipware, RTM.www.chipware.com. Accessed 13 Jan 2010
23. Jones KS, Willet P (1997) Reading in information retrieval. Morgan Kaufmann Publishers, San Francisco
24. Kaindl H, Kramer S, Diallo PSN (1999) Semiautomatic generation of glossary links: a practical solution. In: Proc. Hypertext-1999, pp 3–12
25. Le L, Aikat J, Jeffay K, Smith F (2003) The effects of active queue management on web performance. In: Proc. SIGCOMM, pp 265–274
26. Level 1A (L1A) and Geolocation Processing Software Requirements Specification, SDST-059A, GSFC SBRS, Sept 11, 1997
27. Mäder P, Gotel O, Philippow I (2008) Rule-based maintenance of post-requirements traceability relations. In: Proceedings of the 2008 16th IEEE international requirements engineering conference (Sept 08–12, 2008)
28. Marcus A, Maletic JI (2003) Recovering documentation-to-source-code traceability links using latent semantic indexing. ICSE, pp 125–137
29. McGill K, Deadrick W, Hayes JH, Dekhtyar A (2006) Houston, we have a success story: technology transfer at the NASA IV&V facility. In: Proceedings, workshop on technology transfer in software engineering. Shaghai, China, 2006
30. McMillan C, Poshyvanyk C, Revelle M (2009) Combining textual and structural analysis of software artifacts for traceability link recovery. In: Proceedings of traceability in emerging forms of software engineering (TEFSE), an ICSE workshop, TEFSE
31. Menzies T, Greenwald J, Frank A (2007) Data mining static code attributes to learn defect predictors. IEEE Trans Software Eng 33(1):2–13
32. Menzies T, Dekhtyar A, Distefano J (2007) Jeremy Greenwald: problems with precision: a response to "comments on 'data mining static code attributes to learn defect predictors'". IEEE Trans Software Eng 33(9):637–640

33. MODIS Science Data Processing Software Requirements Specification Version 2, SDST-089, GSFC SBRS, Nov 10, 1997
34. Pohl K (1996) PRO-ART: enabling requirements pre-traceability. In: Proceedings of the 2nd IEEE international. Conference on requirements engineering (ICRE 1996)
35. Porter MF (1980) An algorithm for suffix stripping. Program 14(3):130–137
36. Robertson S, Walker S (2000) Okapi/Keenbow at TREC-8. In: Proceedings eighth text retrieval conference (TREC-8)
37. Spanoudakis G, Zisman A, Pérez-Miñana E, Krause P (2004) Rule-based generation of requirements traceability relations. Journal of Systems and Software (JSS) 72(2):105–127
38. Spanoudakis G, Zisman A (2005) Software traceability: a roadmap. In: Chang SK (ed) Advances in software engineering and knowledge engineering, vol 3: recent advances. World Scientific Publishing, ISBN: 981-256-273-7, Aug 2005
39. Srinivasan P, Ruiz ME, Kraft DH, Chen J (2001) Vocabulary mining for information retrieval: rough sets and fuzzy sets. Information Processing and Management 37:15–38
40. Sundaram S, Hayes JH, Dekhtyar A (2005) Baselines in requirements tracing. In: Workshop on predictive models of software engineering (PROMISE'05), associated with ICSE 2005, St. Louis, MO 2005
41. IBM Rational DOORS. www-01.ibm.com/software/awdtools/doors. Accessed 13 Jan 2010
42. Vincent R, Horling B, Lesser V, Wagner T (2001) Implementing soft real-time agent control. In: Proceedings of the international conference on autonomous agents. pp 355–364
43. Winkler S (2009) Trace retrieval for evolving artifacts. In: ICSE workshop on traceability in emerging forms of software engineering. pp 49–56
44. Zhang H, Zhang X (2007) Comments on 'data mining static code attributes to learn defect predictors. In: IEEE Trans. Software Eng., Sept 2007