Measuring the Effectiveness of Retrieval Techniques in Software Engineering

Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram

Department of Computer Science, University of Kentucky,

{hayes,dekhtyar}@cs.uky.edu, skart2@uky.edu

No Institute Given

Abstract. Mining *textual* artifacts is important for a large array of software engineering tasks: software reuse, software maintenance, software quality assurance, to name a few. Much of the work on mining software repositories has tended to "exclude" such non-structured artifacts. At the same time, we find these items to be rich in semantic information and feel that mining techniques should treat text as software and address their efficient mining. We investigate the application of information retrieval (IR) techniques to the tracing of textual elements in the software repository.

Some textual mining activities are very critical (e.g., tracing artifacts to assure satisfaction of safety requirements) and require analyst participation. We describe our approach to eliciting and processing analyst feedback for the tracing of textual elements of a repository. We then present a study that shows that standard IR methods combined with analyst feedback outperform IR methods alone in terms of coverage (recall - did we find all the relevant links?) and signal-to-noise ratio (precision - were the links we found relevant?).

With the analyst "in the loop," it is necessary to ensure that the tracing software possesses quality from the perspective of the analyst. We examined standard measures for evaluating IR methods and found that they do not always suffice for examining a tool from the analyst's perspective. To address this, we developed a set of secondary measures for evaluating the tracing software. We show, by counter-examples from two projects, that standard measures alone do not provide the detail necessary for adequately evaluating mining tools from the analyst's perspective.

Keywords:software repositories, data mining, software artifacts, requirements tracing, secondary measures, analyst feedback, technique evaluation, Information Retrieval.

1 Introduction

The mining of software repositories has become an important area of research in software engineering. Our ability to retrieve relevant, timely information on developed and evolving software impacts numerous areas: software reuse, software maintenance, software quality, as well as developer productivity, to name a few. Effi cient, effective techniques for mining software repositories are needed.

Much of the work on mining software repositories has tended to ”exclude” non-structured artifacts, such as textual requirements or bug reports. We find these items to be rich in semantic information and feel that MSR techniques should treat text as software and address their efficient mining [11]. To date, we have found that information retrieval (IR) methods can be used to support the processing of textual software artifacts. Specifically, these methods can be used to facilitate the tracing of *textual* software artifacts to each other (such as tracing design elements to requirements). We have found that we can generate candidate links in an automated fashion faster than humans; we can retrieve more true links than humans; and we can allow the analyst to participate in the process in a limited way and realize vast results improvements [19, 18].

At the same time, evaluation of the success of mining techniques in Software Engingeering is a more complex task. Typically, performance of IR methods is measured by *recall*, the percentage of correct links retrieved by the method, and *precision*, the percentage of correct links retrieved by the method.

Note that evaluating mining techniques in Software Engineering is a very different problem than in the IR field. The IR community asks for a human opinion on the relevance of returned links and uses this as the final answer. But we have an additional hurdle in our evaluation. Our goal is to evaluate how close the results of the analyst inspection of an automated tool output resemble the *theoretical true trace* (which shows what artifact elements are relevant to each other). We evaluate the final result of the software and analyst working together.

Some textual mining activities are very critical, e.g., tracing various levels of artifacts to each other in order to assure satisfaction of safety requirements. Such activities require analyst participation, to check, “certify” the correctness of, and (if needed) correct the choices made by the automated procedure. The traditional IR approach to human involvement is a relevance feedback loop: the human analyst examines some of the answers, decides whether they are correct, conveys the information to the software, that then uses its information to recompute its relevance judgments. Recall and precision remain as the measures of performance for relevance feedback.

When automated mining tools are analyzed from the analyst's perspective, however, precision and recall often are insuffi cient. In such settings, we have to evaluate the quality of the **final results**, produced *after* the analyst has examined the tool output(s). The biggest unknown here is *what the analyst does (will do)* with the tool output. This question can be broken into two components: (i) evaluation of the quality of the tool output from the analyst's perspective and (ii) study and evaluation of the work of analysts with different output data. For simplicity, we call the former part the *objective evaluation* and the latter part the *subjective evaluation* of the mining process. In this paper, we concentrate on addressing the question of *objective evaluation*, leaving *subjective evaluation* for future study[1].

To show that precision and recall are insuffi cient from the analyst's perspective, we present a (hypothetical) situation. Consider an analyst using two automated tracing tools A and B to search for existing software modules that assist with error logging (to facilitate software reuse, for example). 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% and precision of 50% (very good results). Result list A displays all of 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 (see Figure 1). Recall and precision indicate that these two tools have identical quality levels from the developer's perspective. However, from the perspective of the analyst, 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. 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 candidate result lists, ensuring, for example, that list B will be deemed superior to list A.

The contribution of this paper is two-fold: we describe the application of IR methods with relevance feedback to the problems of traceability of textual artifacts; and we demonstrate that secondary measures assist us in evaluating results from the analyst's perspective and we study

[1] Subjective evaluation requires a large human-subjects study, which is currently in the planning stage.

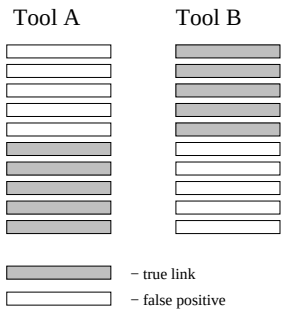


Fig. 1. Precision and recall are not enough for evaluating the results from the analyst's perspective.

how secondary measures *affect the evaluation* of different methods (and in fact can show different results than primary measures alone).

The paper is organized as follows. Section 2 presents an overview of the use of IR techniques and feedback in MSR. We introduce analyst perspective in Section 3. Evaluation of such techniques using primary and secondary measures is discussed in Section 4. Validation of the usefulness of analyst feedback and secondary measures is presented in Section 5. Section 7 presents related work, and Section 8 discusses conclusions and future work.

2 Information Retrieval Techniques in MSR

The two most common tasks when mining textual artifacts in software engineering are tracing and classification. The latter is used when information from a specific textual artifact (e.g., requirements document or bug report collection) needs to be broken into specific predetermined groups (e.g., types of bugs). The former task facilitates comparison of parts of textual artifacts to each other.

The case study for this paper comes from the tracing of textual artifacts. In the rest of this section, we describe some standard Information Retrieval techniques we have used for automated tracing. In a tracing task, the subject of study is a pair of textual artifacts H and D. Each artifact is broken into simple elements (sub-components): $H = h_1, \ldots, h_m$, $D = d_1, \ldots, d_n$. We describe

below two traditional Information Retrieval methods, tf-idf [5] and latent semantic indexing (LSI) [13] and the use of a simple thesaurus. After that, we introduce the methodology for a relevance feedback loop [5].

2.1 Information Retrieval

Tf-Idf Let $V = \{k_1, \ldots, k_N\}$ be the vocabulary (list of keywords) found in H and D. A vector model of an element d (h) is a vector $d = (w_1, \ldots, w_N)$ ($h = (q_1, \ldots, q_N)$) of keyword weights, where w_i (q_i) is computed as $w_i = tf_i(d) \cdot idf_i$ ($q_i = tf_i(h) \cdot idf_i$). Here $tf_i(d)$ ($tf_i(h)$) is the so-called *term frequency*: the frequency of keyword k_i in the element d (h), and idf_i, called inverse document frequency, is computed as $idf_i = \log_2 \left(\frac{n+m}{df_i}\right)$, where df_i is the number of elements from the set $\{h_1, \ldots h_m\} \cup \{d_1, \ldots d_n\}$ in which keyword k_i occurs. Given vectors $d = (w_1, \ldots, w_N)$ and $h = (q_1, \ldots, q_N)$), the similarity between d and h is defi ned as the cosine of the angle between the vectors:

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

By computing $sim(d, h)$ for each pair of elements from D and H we can construct ranked lists of *similar elements* from the two artifacts.

Tf-idf + Simple Thesaurus We can extend the tf-idf method by using a simple thesaurus of terms and keyphrases [18]. A simple thesaurus T is a set of triples $\langle t, t', \alpha \rangle$, where t and t' are matching thesaurus terms (keywords or phrases) and α is the similarity coeffi cient between them. For example, a simple thesaurus can contain a triple (“error”,“fault”,0.9). The vector model is augmented to account for thesaurus matches as follows. First, all thesaurus terms that are not keywords (i.e., thesaurus terms that consist of more than one keyword) are added as separate keywords to the document collection vocabulary. Given a thesaurus $T = \{\langle k_i, k_j, \alpha_{ij} \rangle\}$, and artifact elements d and h, the similarity between them is computed as (assuming that the size of the vocabulary enhanced with thesaurus terms is M):

$$sim(d, h) = \frac{\sum_{i=i}^{M} w_i \cdot q_i + \sum_{\langle k_i, k_j, \alpha_{ij} \rangle \in T} \alpha_{ij} (w_i \cdot q_j + w_j \cdot q_i)}{\sqrt{\sum_{i=1}^{N} w_i^2 \cdot \sum_{i=1}^{N} q_i^2}}.$$

Latent Semantic Indexing The element-by-term matrix consisting of element vectors $d_1, \ldots, d_n$ and $h_1, \ldots h_m$ as columns, may have large dimensionality due to a large number of keywords found in the artifacts. The true similarities in the elements may be obscured by synonomy, polysemy and other language use artifacts. Latent Semantic Indexing (LSI) [13] reduces the effects of language peculiarities by reducing the dimensionality of the element-by-term matrix. The latter is done by replacing it with a matrix of orthogonal components obtained as a result of Singular Value Decomposition (SVD) [14] of the original matrix X with dimensions $N \times m + n$:

$$X = TSD^T.$$

Here, S is a diagonal matrix of eigenvalues of $X^T X$ (also called singular values), and T and D are $M \times rank(X)$ and $rank(X) \times N$ matrices with orthonormal columns. Dimensionality reduction is achieved by replacing S with a matrix S_k, for some $k < rank(X)$, which consists of the first k diagonal elements of S. Matrix $X' = TS_kD^T$ is then used in place of X.

2.2 Incorporating Relevance Feedback

Relevance feedback is a technique to utilize a user's input to improve the performance of the retrieval algorithms. In a nutshell, the relevance feedback procedure works as follows. Given an element h, the analyst observes a list $D'_h = (d'_1, \ldots, d'_k)$ of elements ranked in the order of decreasing relevance score $sim(h, d'_i)$. The analyst examines some of the elements d'_i and decides whether or not they are relevant and confides her decision to the software. The new information obtained from the analyst leads to recomputation of the vector of keyword weights representing h, after which the IR method used to produce relevance scores is rerun with the new vector h' to produce a new ranked list of potential matches. Such a procedure can continue until the analyst is satisfied.

For tf-idf-based retrieval procedures, the traditional way of incorporating analyst feedback into the computation is the Standard Rochio method [5]. Let D'_h contain two subsets: D'_r of size R of elements deemed *relevant* by the analyst and D'_{irr} of size S of elements deemed *irrelevant* (note

that D'_h may contain many more *unvisited* elements). The new vector of keyword weights h' for element h is constructed as follows:

$$h' = \alpha h + \frac{\beta}{R} \sum_{d' \in D'_r} d' - \frac{\gamma}{S} \sum_{d'' \in D'_{irr}} d'',$$

where α, β, γ are parameters that determine the importance of taking into account the original vector (α), positive information (β), and negative information (γ).

3 Evaluation from the Analyst's perspective

The importance of perspective has been aptly demonstrated in the area of reading techniques for improving the quality of textual artifacts (in works such as that of Basili [6]). Perspective plays a role in the mining of textual software artifacts as well. Too often, the only perspective is that of developer (that is, the developer of the mining tool). We have found that the analyst's perspective is also valuable and in fact may encompass very different needs and interests than that of the developer. Below we examine the importance of perspective and the objective and subjective aspects of evaluating mining tools from the analyst's perspective.

3.1 The importance of perspective

As mentioned in the Introduction, perspective is often not captured by primary measures. By perspective, we mean the specific interests and needs of a given stakeholder or role (much as it is defined for perspective-based reading [6], for example). Examples of perspectives for a mining technique include: technique developer, manager of the technique developer, analyst or software developer using a technique, tester of a technique, manager of analysts using a technique, etc. The developer is interested in meeting high-level requirements, such as performance requirements, functional requirements. The manager of the developer may be more concerned with ensuring that the product is developed on schedule and under budget than whether or not all high level requirements are met.

To properly evaluate the effectiveness of a mining technique, we must take perspective into account. To illustrate, imagine a mining tool that has been written by the developer to execute very quickly. The completed product delivers results in mili-seconds, very satisfying results from the developer's perspective. However, the results are not very accurate (items are not retrieved that should be, items are retrieved that should not be). From the perspective of the analyst, the product is not useful at all. The analyst would rather have more accurate results, even if it takes the tool longer to run.

While the view expressed above is somewhat simplistic, it does illustrate how perspective affects evaluation of the quality of the method and/or software implementing it. In reality, the difference between the analyst and developer perspective is more nuanced. Both perspectives include *accuracy of the results* as one of the goals. However, *accuracy* is treated differently by analysts and developers. The developer concentrates on the accuracy of the tool/method itself. For the analyst, the most important thing is the accuracy of the *final result*, achieved after the analyst has inspected the output of the tool.

We illustrate the analyst perspective for a mining tool on the following example. Consider a tracing tool and the tasks that the analyst must perform. Tracing consists of document parsing, candidate link generation, candidate link evaluation, and traceability analysis. To illustrate, consider textual requirements in a Software Requirement Specification being traced to elements in the lower level Software Design Specification. Generally, after the documents have been parsed and elements have been extracted from the two document levels, a matching algorithm of some sort will be applied to build lists of low-level elements that may potentially satisfy a given high-level element. These are called candidate links.

In the process called candidate link evaluation, the analyst reviews the candidate links and determines which are actual, or true links, and which are not links (false-positives, bad links). To achieve this, the analyst typically examines visually the text of the elements, determines the meanings of the elements, compares the meanings and makes the decision based on whether (s)he believes that the meanings are sufficiently close. This determination is based on human judgment

and bears all the advantages and disadvantages that are associated with that. After tracing is complete, the analyst generates reports of the high level elements that do not have children and the low level elements that do not have parents (traceability analysis).

We examine each step of the process after parsing and posit what would represent desired behavior of the tracing tool from the analyst's perspective. We base our analysis on our experience plus that of analysts who have worked for us in industry. For candidate link generation, the analyst would like for all the true links to be found by the tool and no false-positives to be found by the tool. This corresponds to an idealistic goal of 100% recall and 100% precision for the tool. This goal is consistent with the goal of the developer of the software. Given that these numbers are not realistically achievable, both developers and analysts must have a "backup plan." For developers the accuracy of the output is assured by *high precision* and *high recall*. Analysts, however, *must go further*.

First, given a choice, the analyst would likely prefer to have all the true links found even if some false-positives are also returned. That reduces the analyst's need to use other tools to perform interactive searches and look for the missing true links. We will refer to this as *"recall"-oriented accuracy* (accurate results are returned, with emphasis on recall).

Second, for candidate link evaluation, the analyst would like to be able to trust the relevance weights associated with each candidate link. That is, the analyst's preference is that *elements with high relevance scores are true links whereas elements with low scores are false-positives*. We will refer to this as *useful relevances*. As discussed in the example in the Introduction (see Figure 1), precision and recall are inadequate in assessement of the internal structure of the candidate lists. At the same time, this very structure, as well as *its progress over time* becomes an important aspect of the analyst's perspective.

3.2 On objective versus subjective assessment

With the analyst "in the loop," there are now additional degrees of freedom that can impact the software results. Typically, a software system can be seen to be only as good as its inputs. That

is, a system is essentially "garbage in - garbage out." A tracing tool is no exception. If an analyst provides us with feedback that is 100% accurate, the tool can achieve its best possible results. If an analyst provides feedback that is not accurate, the tool cannot be expected to produce the desired results.

Note that our main concern is the quality of the FINAL mining results. That is to say, even if a particular mining method provides poor initial results, yet works in such a way as to drastically improve results given analyst feedback, it still may be a very viable method. On the other hand, a method that provides excellent initial results but, upon analyst feedback, actually produces worse results (e.g., throws away true links and keeps false-positives), is not valuable.

The evaluation of the analyst's work with mining software thus boils down to checking the following two hypotheses:

- **subjective evaluation:** if "good outputs" are produced by the mining tool, the analyst will, in turn, produce "good" final results;
- **objective evaluation:** the mining tool is *capable of* producing "good outputs."

We note, that to fully understand what "good outputs" mean in this context, we must first study what analysts are *likely to do* with the output of mining tools. We are currently preparing to conduct the subjective evaluation of the text mining process. It will involve producing outputs with different characteristics (precision, recall, secondary measures described in Section 4) and letting the analyst interact with the software and produce final results, that will then be evaluated.

At the same time, accuracy of the mining tool output is an important component of "goodness." Indeed, analyst's work with the output is "garbage in — garbage out," so in the presence of inaccurate results, we should not expect the final results to show significant improvement. In this paper, we concentrate on the definition of "goodness of output" that incorporates a more complex description of accuracy than a simple precision-recall pair of measures.

4 Evaluation of Techniques

This Section will describe primary and secondary measures for evaluating mining techniques, specifically tracing techniques.

4.1 Primary Measures

As mentioned in Section 1, IR techniques have traditionally been evaluated using recall and precision. We define these terms, using a tracing activity to illustrate. Consider again, as in Section 2, a pair of textual artifacts H and D, broken into simple elements/subcomponents, $H = h_1, \ldots, h_m$, $D = d_1, \ldots, d_n$. Given an element $h \in H$, let $D'_h = (d'_1, \ldots, d'_k)$ be the (ranked) list of all elements $d \in D$ such that $sim(h, d) > 0$ according to the method under evaluation. Let D_h be a set of all elements $d \in D$ that match h, $|D_h| = R$ and let $|D_h \cap D'_h| = r$.

Recall, the measure of the percentage of true matches found, is defined as

$$recall = \frac{r}{R}.$$

Precision, the measure of the percentage of true matches in D'_h, is defined as

$$precision = \frac{r}{k}.$$

Given the entire list $h_1, \ldots, h_m$ of elements of H, the recall and precision of the method as applied to the artifact pair (D, H) are

$$recall = \frac{\sum_{i=1}^{m} |D_{h_i} \cap D'_{h_i}|}{\sum_{i=1}^{m} |D_{h_i}|};$$

$$precision = \frac{\sum_{i=1}^{m} |D_{h_i} \cap D'_{h_i}|}{\sum_{i=1}^{m} |D'_{h_i}|}.$$

4.2 Secondary Measures

As argued above, there are many situations, however, where recall and precision do not provide sufficiently accurate information about the actual structure of the candidate lists. To address this

need, we have developed secondary measures for evaluating IR techniques as applied to MSR from the analyst's perspective. These can be applied to techniques used for a variety of purposes, as discussed in Section 2. Our measures help assess the quality of returned candidate lists. The measures help to 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.

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

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 and a false positive 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 – 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.

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

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

Lag. *DiffAR* and *DiffMR* look at the quantitative difference between the similarity scores of true matches and false positives. Note that $L = \cup_{h\in H} L_h$, 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 really 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.

Definition 1. *Let d be an element of a textual artifact D and h - 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).

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 side-by-side with precision, in order to determine whether the candidate lists returned by a text mining tool are if 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 by manually going through the list generated by an automated method rather than manually comparing each pair of elements. The smaller the selectivity, the better are the savings for the analyst.

5 Validation of Feedback and Secondary Measures

The authors performed two studies: one to address the effectiveness of feedback for tracing textual artifacts, and one to assess the usefulness of secondary measures. In [20], we have outlined a framework for evaluation of requirements tracing experiments. In the sections below, we present our experimental results, cast within this framework.

5.1 Assessing the Effectiveness of Analyst Feedback in Tracing

Motivation The motivation was to improve traceability link recovery between hierarchical levels of textual requirements documents from the analyst's perspective.

Purpose The purpose of the experiment was to evaluate whether or not the application of feedback combined with IR techniques resulted in improved requirements tracing, as measured using recall and precision.

Object We studied requirements tracing algorithms.

Hypothesis

- *Null Hypothesis 1:* The results of using requirements tracing algorithms (tf-idf, LSI, tf-idf+Thesaurus, LSI+Thesaurus) enhanced by analyst feedback as measured by recall and precision will not vary from the results of these methods without the feedback on the same dataset/project.
- *Alternative Hypothesis 1:* The results of using requirements tracing algorithms (tf-idf, LSI, tf-idf+Thesaurus, LSI+Thesaurus) enhanced by analyst feedback as measured by recall and precision will improve upon the results of these methods without the feedback on the same dataset/project.

Perspective The experiments have been conducted from the Analyst perspective.

Domain, Scope, Importance, and Experimental Design The scope was single project where one project (from the program domain) of quality of life importance was traced. The object of study importance was quality of life. The independent variables were traceability algorithm (tf-idf, tf-idf+Thesaurus, LSI, LSI+Thesaurus), feedback behavior (Top 1 through Top 4)(see Section 6), and use of filtering (See Section 6.).

Measurement and Product The recall and precision measures are formally defined, validated measures from the information retrieval field. The measures were collected in an automated fashion and are ratio. The products were two levels of documentation from two NASA software projects.

Preparation No pilot study was performed. An example of one of the artifact elements is shown in Figure 2.

Execution, Analysis and Interpretation See Section 6.

5.2 Assessing Secondary Measures for Evaluating Quality from the Analyst Perspective

The authors looked at the use of the same IR methods as above for automated tracing with an emphasis on the effectiveness of secondary measures for evaluating quality from the analyst's perspective.

Motivation The motivation was to assess the effectiveness of secondary measures (DiffAR, Lag, Selectivity) in evaluating the improvement of traceability link recovery between hierarchical levels of textual requirements documents from the analyst's perspective.

Purpose The purpose of the experiment was to determine if secondary measures improve/alter the assessement of the quality of the result lists produced by automated IR methods.

Object We studied two categories of objects: secondary measures and requirements tracing algorithms.

Hypothesis

- *Null Hypothesis 2:* The assessment of the quality of the candidate lists produced by automated IR methods as characterized by primary measures (see Section 5.1) *does not change* when secondary measures (DiffAR, Lag, Selectivity) are evaluated.
- *Alternative Hypothesis 2:* In some cases, the assessment of the quality of the candidate lists produced by automated IR methods as characterized by primary measures (see Section 5.1) *does change* when secondary measures (DiffAR, Lag, Selectivity) are evaluated.

Perspective The experiments have been conducted from the Analyst perspective.

Domain, Scope, Importance, and Experimental Design The scope was single project where one project (from the program domain) of quality of life importance was traced. The object of study importance was quality of life. The independent variables were traceability algorithm (tf-idf, tf-idf+Thesaurus, LSI, LSI+Thesaurus), feedback behavior (Top 1 through Top 4)(see Section 6), and use of filtering (See Section 6.).

Measurement and product Lag, selectivity, and DiffAR were defined in [19] and in Section 4. DiffMR is defined in Section 4. The metrics were collected in an automated fashion and are ratio. The products were two levels of documentation from two NASA software projects.

Preparation No pilot study was performed. An example of one of the artifact elements is shown in Figure 2.

" The DPU-1553 CSC shall address hardware modules as defined in document 1400, Company X Specification for the Company X Communication/Memory Module."
CM-1 [27]

Fig. 2. Sample requirements for CM-1 dataset.

Execution, Analysis and Interpretation Note that a counter-example will suffice to reject the null hypothesis. That is, if instances can be found where the assessement of the output quality based solely on primary measures is altered by considering primary measures *in conjunction with secondary measures*, the null hypothesis will be rejected in favor of the alternative hypothesis. We also note that the change in the quality effected by the consideration of the secondary measures can be both positive and negative. Detailed analysis is found in Section 6.

6 Analysis and Interpretation of Experimental Results

The results of the two experiments are presented below.

6.1 Assessing the Effectiveness of Analyst Feedback in Tracing

Execution of Experiments Two NASA datasets were used, open source Moderate Resolution Imaging Spectroradiometer (MODIS) [25], [28] and CM-1 [27][2]. The MODIS dataset contains 19 high level and 49 low-level requirements with 41 true links. The CM-1 dataset consists of a complete requirements (high-level) document and a complete design (low-level) document for a NASA scientific instrument. The project is written in C with approximately 20 KSLOC. It was made available by the Metrics Data Program (MDP) [27]. The text of the documents has been altered by NASA prior to public release in order to hide the identity of the instrument. The CM-1 dataset contains 235 high-level requirements and 220 design elements with 361 true links.

High and low-level elements were parsed from each artifact. The elements were then subjected to stemming and stopword removal. The resulting information was passed to the specific IR method for creation of vectors of term weights. Next, the selected IR technique was used to generate candidate links between the artifact levels. 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

[2] The MODIS dataset is available on the Software Engineering Empirical Website (SEEWEB)[30]

links in the list, and specified whether each examined link was a true link or a false positive. This information, encoded in XML, 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 true trace) to the results obtained by RETRO (returned candidate link lists) for every iteration. 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^α 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.

Results and Analysis Rejection of null hypothesis 1 in favor of alternative hypothesis 1 was supported by the full results obtained from a large experimental study that is detailed in [21]. Because the ultimate goal of this paper is to compare the assessements of results made using only primary measures and those made using both primary and secondary measures, we show only one specific case for each dataset. These cases contribute directly to our rejection of the first null hypothesis in favor of the alternative hypothesis.

Figures 3.(a) and 6.(b) show recall vs. precision trajectories of some selected methods for MODIS and CM-1 datasets, respectively. In both graphs, the leftmost starting point of the trajectory corresponds to iteration 0, the result of the selected method without applying analyst feedback. Each successive point represents analyst feedback for the next iteration. The endpoint of the trajectory represents the recall and precision at the end of the 8th iteration.

Figure 3 compares the trajectories of tf-idf and tf-idf+thesaurus methods on the MODIS dataset. Both methods were used in conjunction with Top 2 feedback strategy and all candidate lists have been filtered with the filter value $\alpha = 0.05$. As seen in the graph in Figure 3.(a), recall increases from roughly 0.5 to 0.8 for the non-thesaurus version of tf-idf and from approximately 0.78 to

approximately 0.96 for the thesaurus version of tf-idf. Precision increases from approximately 0.08 to 0.6 and from 0.1 to approximately 0.42, respectively.

The recall values are very high, especially for the thesaurus case, and precision increases significantly during the feedback process. A head-to-head comparison of the two methods suggests that tf-idf + thesaurus method is somewhat superior to simple tf-idf, as recall is a more important characteristic of accuracy for tracing than precision.

Figure 6.(b) compares the trajectories of tf-idf method with Top 2 feedback as applied to CM-1 dataset (with filter set at 0.1) and MODIS dataset (no filtering). These two specific cases have been selected because, despite certain dissimilarity in the values of independent variables (dataset, filter use), the trajectories themselves are rather similar. In fact, the trajectories coincide in their last part. Here, recall increases from 0.75 to 0.85 for MODIS and from 0.76 to 0.85 for CM-1. Precision increases from 0.08 to 0.18 and from 0.11 to approximately 0.18, respectively.

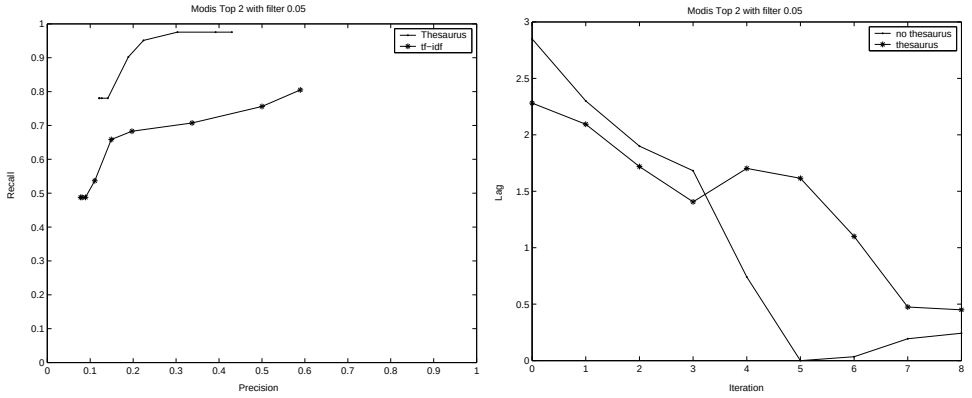
The recall values again are high. Precision is somewhat low. But, the increase in precision from around 10% (about one true link in 10) to 18% (about 1 true link in any 5-6) is significant.

6.2 Assessing Secondary Measures for Evaluating Quality from the Analyst Perspective

Execution of Experiments The experiments were conducted as described in Section 6.1, but our emphasis was on the secondary measures collected.

Results and Analysis Several counter-examples are presented below in support of rejecting the second null hypothesis in favor of the alternative hypothesis. Each counter-example demonstrates situations in our experiments where the secondary measures significantly affect the assessment of the quality of the results from the analyst's perspective.

Example 1. Consider the graph in Figure 3.(a) again. As stated above, based on precision and recall data alone, it appears that the tf-idf+thesaurus technique is better in terms of recall and would be selected over simple tf-idf method.



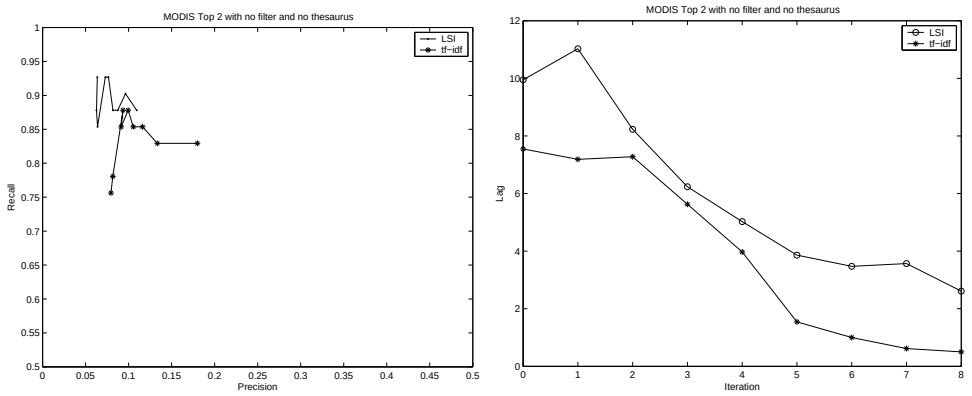
(a) (b)

Fig. 3. (a)Recall and precision and (b) Lag for MODIS dataset, Top 2 feedback, filter 0.05, Thesaurus versus No thesaurus.

Next, we look at the secondary measure of Lag for the same scenario (depicted in Figure 3.(b)). 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) 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 are formed by (almost) exclusively relevant matches sooner.

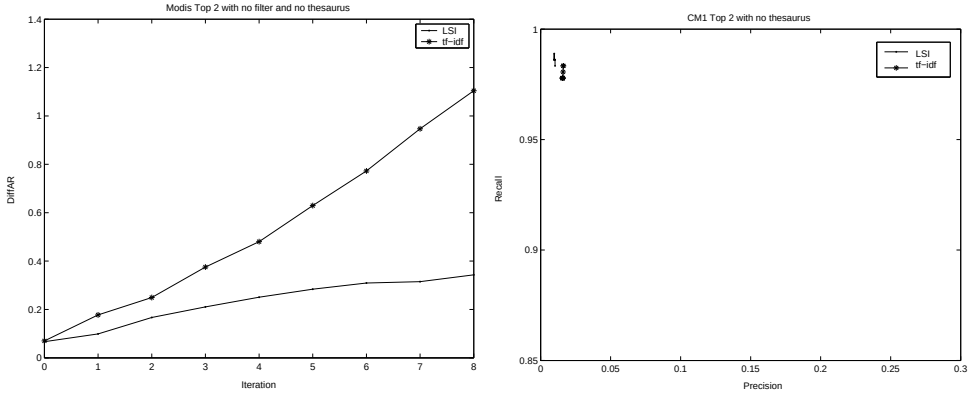
Example 2. The graph in Figure 4.(a) compares the recall vs.precision trajectories obtained in our experiments for LSI and tf-idf methods using Top 2 feedback and no filtering on the MODIS dataset. The trajectories are close to each other, with LSI showing somewhat better recall, while tf-idf eventually moving towards better precision (over 20%). 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 not in this case.

Now, we look at the secondary measure of Lag for the same scenario (shown in Figure 4.(b)). 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



(a) (b)

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

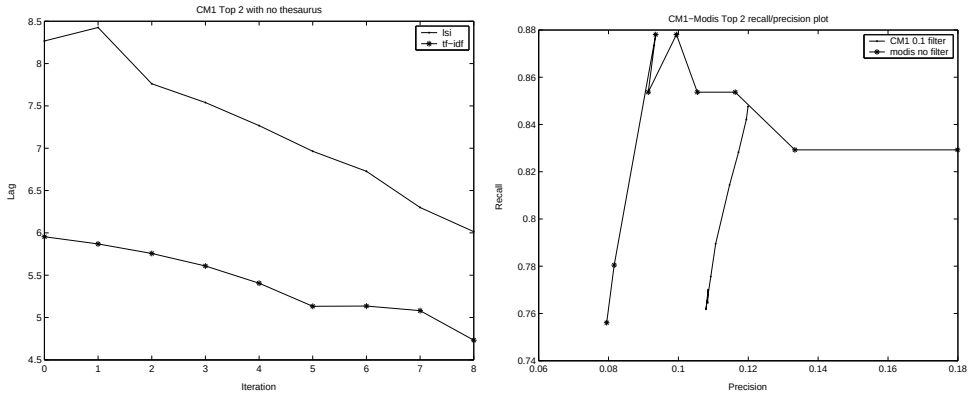


(a) (b)

Fig. 5. (a) DiffAR for MODIS dataset, Top 2 feedback, no filter, LSI versus tf-idf. (b) Recall and precision for CM-1 dataset, Top 2 feedback, no filter, LSI versus tf-idf.

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 recieves even more support upon examination of the DiffAR trends shown in Figure 5.(a): DiffAR for tf-idf shows a huge improvement over DiffAR for LSI. Based on the secondary measures, an analyst would prefer tf-idf, even though recall is somewhat lower than for LSI.



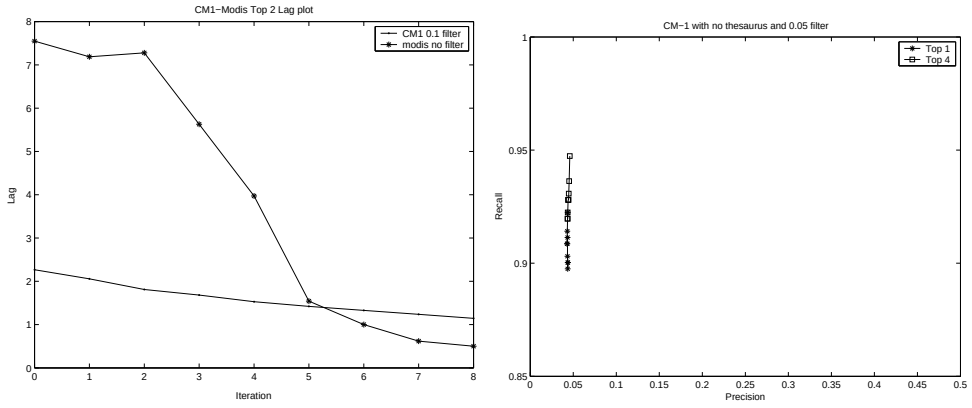
(a) (b)

Fig. 6. (a) Lag for CM-1 dataset, Top 2 feedback, no filter, LSI versus tf-idf. (b) Recall and precision, tf-idf, Top 2 feedback, CM-1 filter 0.1 versus MODIS no filter.

Example 3. The graph in Figure 5.(b) compares the recall vs. precision trajectories for LSI and tf-idf methods on the CM-1 dataset using Top 2 feedback and no filtering. The graphs show almost not change in precision (and precision itself is unacceptably low) and only slight change in recall for both methods (this run actually supports the first null hypothesis. However, [21] shows how filtering improves precision significantly without hurting recall much). 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 Figure 6.(a). 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 1-2 range), it is clear that tf-idf outperformed LSI, thus providing us with a clear tiebreaker. Again, consideration of a secondary measure changes the scenario assessment.



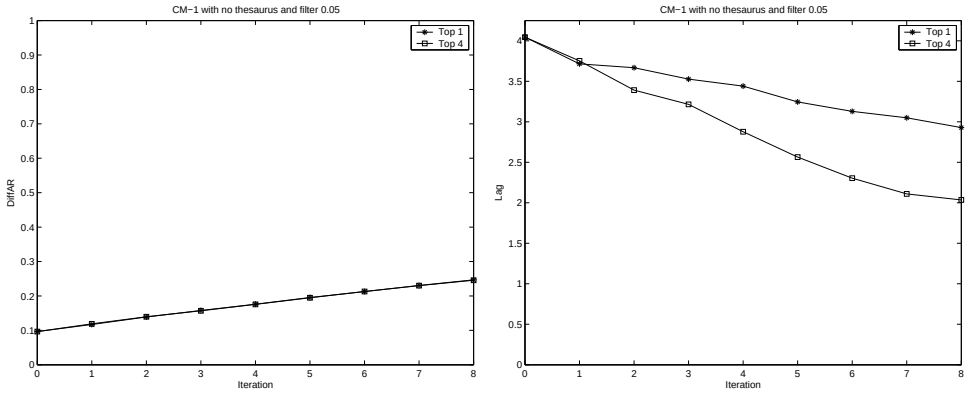
(a) (b)

Fig. 7. (a) Lag, tf-idf, Top 2 feedback, CM-1 filter 0.1 versus MODIS no filter. (b) Recall and precision for CM-1 dataset, filter 0.05, tf-idf, Top 1 versus Top 4 feedback.

Example 4. As discussed above, the graph in Figure 6.(b) compares the recall vs. precision trajectories for tf-idf method with Top 2 feedback on the MODIS (no filtering) and CM-1 (filter set to 0.1) datasets. The trajectories are similar and, in fact, on the last few iterations they follow the same path. It is hard to conclude that one of the two runs is significantly more successful than the other.

Next, we look at the secondary measure of Lag for the same scenario (in Figure 7.(a)). 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 about 1.1.

How do we interpret this? Arguments can be made in favor of stating that tf-idf worked better on CM-1 (relatively low Lag throughout), as well as on MODIS (final Lag around 0.5 vs. 1.1). While the decision of where tf-idf has proven to be more efficient is subject to specific criteria of the analyst, because Lag shows completely different behavior in the two cases discussed here, including it in the observations allows the analyst to actually make an informed decision. When only precision and recall are considered, it is impossible to point to any distinctly different behavior of methods.



(a) (b)

Fig. 8. (a) DiffAR and (b) Lag for CM-1 dataset, filter 0.05, tf-idf, Top 1 versus Top 4 feedback.

Example 5. The graph in Figure 7.(b) shows the recall vs. precision trajectories obtained for tf-idf method on the CM-1 dataset with Top 1 and Top 4 feedback strategies respectively [3]. From this graph, it appears that Top 4 is better. While precision is stable at around 5%, the run that used Top 4 feedback achieves marginally better recall.

Next, we look at the secondary measure of DiffAR for the same scenario (in Figure 8.(a)). Here, we see absolutely no difference between the two feedback behaviors. The secondary measure Lag,

[3] For practical purposes, neither Top 1 nor Top 4 feedback strategies had been deemed good [21]. The former strategy does not let the analyst examine enough data, while the latter requires the analyst to look at too much data on each step.

however, shows that Top 4 has an improvement of 1 link over Top 1 (see Figure 8.(b)). Such results can help us convince an analyst that if they will provide more feedback on each candidate link list, the quality of the lists will improve faster.

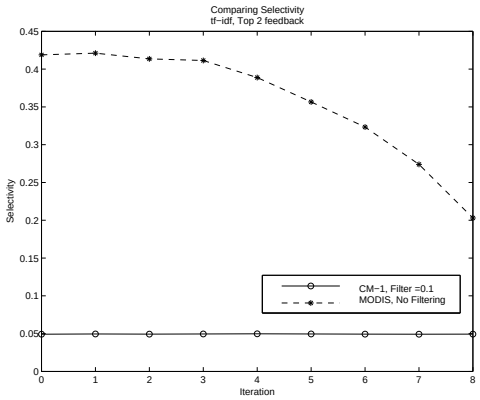


Fig. 9. Selectivity: tf-idf, Top 2 feedback, CM-1 filter 0.1 versus MODIS no filter.

Example 6. In Example 4, we have shown that while the quality of results of tf-idf on two similar runs (one on MODIS dataset with no filtering and one on CM-1 with filter at 0.1) appears similar when only precision and recall are used, the use of Lag provides ground for differentiating between these results (although the interpretation of which is better is open to analyst's specific preferences).

Consider now the graph in Figure 9, comparing the selectivity achieved in these two runs. As seen from this graph, the selectivity of tf-idf on CM-1 keeps steady at just under 5%, while for MODIS it is higher - decreasing from about 42% to 20%. Thus, the same performance in terms of precision and recall means *much greater savings* for the analyst as compared to the manual labor needed to perform the tracing without the automated tools for the CM-1 dataset.

Overall Conclusion The examples shown above illustrate some of the situations we encountered during the tracing experiments when 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. These examples allow us to reject the second null hypothesis in favor of the alternative hypothesis.

7 Related Work

A detailed look at IR techniques for software analysis and requirements tracing can be found in [18] and [21]. This discussion focuses on secondary measures for evaluating techniques from the analyst's perspective and analyst feedback in IR methods.

The IR community does have research on the effectiveness of feedback evaluation [22, 23], but these do not examine secondary measures. As explained in the Introduction, this is to be expected as that community is concerned with evaluating the output from the IR method, not the final answer after analyst feedback has been taken into account.

However, a number of other fields are using secondary measures in their evaluations. In the area of performance assessment, Le, et. al. [24] 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. [40], modified existing multi-agent technologies to provide distributed control for a real-time environment. They found it necessary to use additional secondary measures because hard scheduling deadline (in seconds) did not provide an appropriate grainsize. Haritsa, et. al. [16], 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.

In the area of user satisfaction, Riegelsberger, et. al. [37], conducted a study on trust of e-commerce sites. They used post-order service quality level as a secondary measure. Albinsson and Zhais [2] experimented with two techniques for touch screen interaction, Cross-Keys and

Precision-Handle. The dependent measures were movement time and error rate. Secondary measures concerning comfort were collected based on an International Standards Organization (ISO) questionnaire. Another piece of work on computer human interfaces also examined a new input paradigm. Pearson and Weiser [31] investigated a foot-operated curser-positioning device called a planar slide mole. Use of this device was compared to use of a mouse with the main measure (dependent variable) being time between correct target selection and the secondary measure being error rate.

Hayes, Dekhtyar, and Osborne [18] 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, Dekhtyar, Sundaram, and Howard [19] 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. Dekhtyar, Hayes, and Menzies argued that text is software in [11], illustrating their point with a requirements tracing study. They concluded that mining natural language is not too difficult and that software repositories should routinely be augmented with all the natural language text used to develop that software [11]. Antoniol, Canfra, Delicia, and Merlot [4] 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. Marcus and Maletic [26] 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.

With an emphasis on analyst feedback, Hayes, Dekhtyar, Sundaram, and Howard [19], using a small dataset, found that tf-idf with analyst feedback outperformed tf-idf without analyst feedback, in terms of recall and precision. They also found that tf-idf plus thesaurus with analyst feedback outperformed tf-idf plus thesaurus without analyst feedback, in terms of recall and precision.

8 Conclusions and Future Work

In this paper, we applied IR methods with relevance feedback to the problem of tracing textual artifacts. We demonstrated that, in certain cases, secondary measures affect significantly the analyst's perception of the quality of results. Thus, the secondary measures we studied prove to be an important asset in our quest to evaluate different methods for mining textual repositories in Software Engineering settings. We found support for relevance feedback in tracing, recall and precision increased over tracing without feedback. We also 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:

(i) study of the work of analysts in requirements tracing; and
(ii) study of the applicability of secondary measures to other mining activities in 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.

9 Acknowledgements

Our work is funded by NASA under grant NAG5-11732. We thank Tim Menzies, Stephanie Ferguson, and Ken McGill. We thank Ganapathy Chidambaram for his assistance on LSI. We also thank Sarah Howard and James Osborne who worked on early versions of the software used for the evaluation.

References

1. Abrahams, M. and Barkley, J., RTL Verification Strategies, in *Proc. IEEE WESCON/98*, 15 - 17 September 1998, pp. 130-134.
2. Albinsson, P. and Zhai, S. "High Precision Touch Screen Interaction," (HI Letters, Vol. 5, Issue 1, pp. 105-114, Ft. Lauderdale, FL, April 2003.
3. Anezin, D., "Process and Methods for Requirements Tracing (Software Development Life Cycle)," Dissertation, George Mason University, 1994.
4. Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., and Merlo, E. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, Volume 28, No. 10, October 2002, 970-983.
5. Baeza-Yates, R. and Ribeiro-Neto, B. Modern Information Retrieval, *Addison-Wesley*, 1999.
6. Lab Package for the Empirical Investigation of Perspective-Based Reading, authors: Victor Basili - Scott Green - Oliver Laitenberger - Filippo Lanubile - Forrest Shull - Sivert Sorumgard - Marvin Zelkowitz, http://www.cs.umd.edu/projects/SoftEng/ESEG/manual/pbr_package/manual.html.
7. Bohner, S., A Graph Traceability Approach for Software Change Impact Analysis, *Dissertation, George Mason University*, 1995.
8. Brouse, P., "A Process for Use of Multimedia Information in Requirements Identification and Traceability," Dissertation, George Mason University, 1992.
9. Casotto, A.. Run-time requirement tracing, Proceedings of the IEEE/ACM International Conference on Computer-aided Design, Santa Clara, CA, 1993.
10. Cleland-Huang, J., Chang, C.K., Sethi, G., Javvaji, K.; Hu, H., Xia, J. (2002) Automating speculative queries through event-based requirements traceability. Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02), Essex, Germany, 9-13 September, 2002, pages: 289- 296.
11. Dekhtyar, A., Hayes, J. Huffman, and Menzies, T. "Text is Software Too," Proceedings of the International Workshop on Mining of Software Repositories (MSR) 2004, Edinborough, Scotland, May 2004, pp. 22 - 27.
12. Telelogic product DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm
13. Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. Indexing by latent semantic analysis, *Journal of the Society for Information Science*, Vol.41(6), pp. 391-407, 1990.
14. Forsythe, G.E., Malcolm, M.A. and Moler, C.B., *Computer Methods for Mathematical Computations*, Englewood Cliffs, NJ, Prentice Hall, 1977.
15. Google search engine. http://www.google.com/.
16. Haritsa, J., Carey, M., and Livny, M. On Being Optimistic about Real-Time Constraints, *Proc. of ACM SIGACT-SIGMOD-SIGART Symposium on Principals of Database Systems*, April 1990, pp. 331-340.
17. Hayes, J. Huffman. Risk reduction through requirements tracing. In *Proc., Software Quality Week*, 1990, San Francisco, California, May 1990.
18. Hayes, J. Huffman, Dekhtyar, A., Osbourne, J. Improving Requirements Tracing via Information Retrieval, in *Proceedings of the International Conference on Requirements Engineering (RE)*, Monterey, California, September 2003, pp. 151 - 161.
19. Hayes, J. Huffman, Dekhtyar, A., Sundaram K.S., Howard S., Helping Analysts Trace Requirements: An Objective Look, in *Proceedings of the International Conference on Requirements Engineering (RE)*, Kyoto, Japan, September 2004.
20. Hayes J. Huffman, Dekhtyar A., A Framework for Comparing Requirements Tracing Experiments, *University of Kentucky Technical Report* TR 408-04, June 2004.
21. Hayes J. Huffman, Dekhtyar A., Sundaram, S., and Chidambaram G. On Effectiveness of User Feedback-based Information Retrieval Methods for Requirements Tracing, *University of Kentucky Technical Report*, TR 423-04, October 2004.
22. Jurgen Koenemann, Nicholas J. Belkin: A Case for Interaction: A Study of Interactive Information Retrieval Behavior and Effectiveness. CHI 1996: 205-212
23. Jurgen Koenemann: Supporting Interactive Information Retrieval Through Relevance Feedback. CHI Conference Companion 1996: 49-50
24. Le, L., Aikat, J., Jeffay, K., and Smith, F. "The Effects of Active Queue Management on Web Performance," SIGCOMM, August 2003, pp. 265-274.
25. Level 1A (L1A) and Geolocation Processing Software Requirements Specification, *SDST-059A, GSFC SBRS*, September 11, 1997.
26. Marcus, A., Maletic, J. Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing, in *Proceedings of the Twenty-Fifth International Conference on Software Engineering*, 2003, Portland, Oregon, 3 - 10 May 2003, pp. 125 - 135.

27. MDP Website, CM-1 Project, http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1.
28. MODIS Science Data Processing Software Requirements Specification Version 2, *SDST-089, GSFC SBRS*, November 10, 1997.
29. Mundie, T. and Hallsworth, F. Requirements analysis using SuperTrace PC. In *Proceedings of the American Society of Mechanical Engineers (ASME) for the Computers in Engineering Symposium at the Energy & Environmental Expo*, 1995, Houston, Texas.
30. Offutt, J., Yang, Y., Hayes, J. Huffman, "SEEWeb: Making Experimental Artifacts Available," Workshop on Empirical Research in Software Testing (WERST), in SIGNOTES, Volume 29, Issue 5, September 2004, Boston, MA, July 2004.
31. Pearson, G. and Weiser, M. "Exploratory Evaluation of a Planar Foot-Operated Cursor-Positioning Device," CHI 1988, pp. 13-22.
32. Pierce, R. A requirements tracing tool, Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, 1978.
33. Ramesh, B., Factors Influencing Requirements Traceability Practice, *Communications of the ACM*, December 1998, Volume 41, No. 12, pp. 37-44.
34. Ramesh, B.; Jarke, M. Toward reference models for requirements traceability; IEEE Transactions on Software Engineering, Volume 27, Issue 1, January 2001, page(s): 58 -93.
35. Rational RequisitePro, http://www.rational.com/products /reqpro/index.jsp
36. Holagent Corporation product RDD-100, http://www.holagent.com/new/products/modules.html.
37. Riegelsberger, J., Sasse, M., and McCarthy, J. "Shiny Happy People Building Trust? Photos on e-Commerce Websites and Consumer Trust," CHI 2003, April 2003, pp. 121-130.
38. Savvidis, I. "A Multistrategy Framework for Analyzing System Requirements (Software Development)," Dissertation, George Mason University, 1995.
39. Tsumaki, T. and Morisawa, Y. "A Framework of Requirements Tracing using UML," Proceedings of the Seventh Asia-Pacific Software Engineering Conference 2000, 5 - 8 December 2000, pp. 206 - 213.
40. Vincent, R., Horling, B., Lesser, V. and Wagner, T. "Implementing Soft Real-Time Agent Control," in Proceedings of the International Conference on Autonomous Agents, May 2001, pp. 355 - 364.
41. Watkins, R. and Neal, M. "Why and How of Requirements Tracing," IEEE Software, Volume 11, Issue 4, July 1994, pp. 104-106.