Advancing Candidate Link Generation for Requirements Tracing: the Study of Methods

Jane Huffman Hayes Alex Dekhtyar Senthil Karthikeyan Sundaram
Department of Computer Science, University of Kentucky,
{hayes,dekhtyar}@cs.uky.edu, skart2@uky.edu

Abstract— **This paper addresses the issues related to improving the overall quality of the dynamic candidate link generation for the requirements tracing process for Verification and Validation and Independent Verification and Validation analysts. The contribution of the paper is four-fold: we define goals for a tracing tool based on analyst responsibilities in the tracing process; we introduce several new measures for validating that the goals have been satisfied; we implement analyst feedback in the tracing process; and we present a prototype tool that we built, RETRO (REquirements TRacing On-target), to address these goals. We also present the results of a study used to assess RETRO's support of goals and goal elements that can be measured objectively.**

Index Terms— **Keywords: Requirements tracing, dynamic link generation, Verification and Validation (V&V), Independent Validation and Verification (IV&V), Information Retrieval, TF-IDF, LSI, recall, precision.**

I. INTRODUCTION

The fundamental purpose of Verification and Validation (V&V) and Independent Verification and Validation (IV&V) is to ensure that the right processes have been used to build the right system. To that end, we must verify that the approved processes and artifacts are guiding development during each lifecycle phase as well as validate that all requirements have been implemented at the end of the lifecycle. A requirements traceability matrix (RTM) is a prerequisite for both of these. Though Computer-Aided Software Engineering tools such as DOORS [52], RDD-100 [27], and Rational RequisitePro [43] can assist, we have found that often developers do not build the RTM to the proper level of detail or at all. V&V and IV&V analysts are faced with the time consuming, mind-numbing, person-power intensive, error-prone task of "after the fact" requirements tracing to build and maintain the RTM. Examples of V&V/IV&V activities that can't be undertaken without an RTM include, but are not limited to: verification that a design satisfies the requirements; verification that code satisfies a design; validation that requirements have been implemented and satisfied; criticality analysis; risk assessment; change impact analysis; system level test coverage analysis; regression test selection. V&V/IV&V can be viewed as the backbone of safety-critical, mission-critical, and Critical-Catastrophic High Risk (CCHR) systems [1]. Similarly, the RTM can be viewed as the backbone of V&V/IV&V. We focus on "after the fact" requirements tracing (hereafter referred to simply as "requirements tracing.") Note that our techniques may be applied to any pair of textual artifacts: high level to low level requirements; requirements to design; design to requirements; design to test cases; etc. In this paper, we have tested our techniques on high to low level requirements and on requirements to design[2].

Requirements tracing consists of document parsing, candidate link generation, candidate link evaluation, and traceability analysis. As an example, consider requirements in a high level document such as a System Specification being traced to elements in a lower level document such as a Software Requirement Specification. The most common tracing approach in industry then proceeds as follows. After the documents have been parsed and requirements have been extracted from the two document levels,

[1]Considering that: a) CCHR system IV&V now routinely uses manual approaches to generate candidate links for generating the RTM; and b) our prior research [20] showed that our techniques outperformed the techniques in use by a CCHR IV&V agent, we feel that our approach is at least as appropriate for CCHR systems as the technologies presently in use.

[2]As we discuss in Section V-B similar techniques have been applied by other researchers [3], [4], [31] to tracing documentation to code. In addition, in [55] we report on our experiments on tracing requirements to bug reports.

an analyst reads each high level requirement and low level element and assigns keywords to each. Keyword assignment is either completely manual, or aided by the use of search functions from a word processor/spreadsheet. In some cases, the keywords are chosen from an analyst-defined ontology, created in advance. A keyword-matching algorithm is then applied to build lists of low-level elements that may potentially satisfy a given high-level requirement. These are called candidate links. There are two commonly accepted measures for evaluating candidate link lists: *recall*, measuring the percentage of correct matches that were found, and *precision*, measuring the percentage of found matches that were correct.

In the process called *candidate link evaluation*, the analyst reviews the candidate links and determines those that are actual, or true links, and those that are not links (*false-positives*, *bad links*). To achieve this, the analyst typically visually examines the text of the requirements, determines the meanings of the requirements, 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 requirements that do not have children and the low level elements that do not have parents (traceability analysis).

Current approaches to after-the-fact tracing have numerous shortcomings: they require the analyst to perform interactive searches for potential linking elements, they require the analyst to assign keywords to all the elements in both document levels prior to tracing, they return many candidate links that are not correct, they fail to return correct links, and they do not provide support for easily retracing new versions of documents. To ensure requirement completion and to facilitate change impact assessment, a method for easy "after-the-fact" requirements tracing is needed. For ease of illustration, in this paper we will discuss the tracing of requirements to design. Note that our methods and tool can be used to trace any textual artifact to any other textual artifact.

Previously, we focused solely on the problem of generating candidate links, discussed in [20]. Specifically, we showed that information retrieval (IR) methods were effective and efficient when used to generate candidate link lists. Our focus then broadened to the overall requirements tracing process [21]. A goal of this NASA-funded research is to develop an efficient, effective tracing tool that makes the best use of the analyst's time and expertise (the ultimate goal being the actual improvement of requirements tracing analysis). To that end, this paper provides numerous contributions:

(i) we investigate the analyst responsibilities in performing tracing;
(ii) we derive unique high-level analyst-oriented tool goals from these;
(iii) we implement analyst feedback into the tracing process;
(iv) we develop new measures for assessing the tool goals, and
(v) we present a prototype tool, RETRO (REquirements TRacing On-target), and evaluate it with respect to the goals.

This paper extends [21] by considering an additional IR technique, Latent Semantic Indexing (LSI) [11] for requirements tracing. In addition, we describe the result of our experiments on two datasets: MODIS [30], [33] used in [21], [20] and CM-1 [32], a new, larger dataset. The paper is organized as follows. Section II presents the goals for an effective requirements tracing tool. Section III discusses our tool and how it satisfies the goals of Section II. Section IV discusses the results obtained from evaluation. Related work in requirements tracing and analysis is presented in Section V. Finally, Section VI presents conclusions and areas for future work.

II. Goals for an effective requirements tracing tool

To set the stage for our work, we must first understand the responsibilities of an analyst who has been tasked to perform a requirements trace. In the description that follows, we assume that the analyst is tasked with performing a trace between two requirements documents. Without loss of generality, we call one set of requirements high level and the other low level, and assume that tracing has to be performed from the high level document to the low level document. The process of requirements tracing is described in Table I.

Let us examine how automation may facilitate these responsibilities. A tool could easily assist

Step	Task
(a)	identify each requirement
(b)	assign a unique identifier to each requirement
(c)	for each high level requirement, locate all matching low level requirements
(d)	for each low level requirement, locate a parent requirement in the high level document
(e)	determine if each high level requirement has been completely satisfied
(f)	prepare a report that presents the traceability matrix
(g)	prepare a summary report that expresses the level of traceability of the document pair

TABLE I
THE REQUIREMENTS TRACING PROCESS IN A NUTSHELL.

the analyst in the identification and subsequent extraction and "tagging" of requirements ((a), (b)). Similarly, generation of requirements traceability matrix reports and traceability summary reports lends itself well to automation ((f), (g)). In fact, a number of proprietary tools, such as SuperTrace-Plus (STP)[19], [34], and commercial tools already address these items.

The remaining items are of greater interest and importance to researchers and practitioners. Items (c)-(e) conceivably require the analyst to examine every low level requirement for each high level requirement. Even in a small document pair that consists of 20 high level requirements and 20 low level requirements, an analyst might have to examine 20 x 20 = 400 candidate links.

The goal of our research is to study the ways in which requirements tracing can be automated. Items (c) - (e) are prime candidates for automatation. However, because analysts have critical responsibilities in the requirements tracing process, its full automation cannot be achieved. Indeed, it is the role of the analysts to evaluate candidate links; make decisions on whether or not candidate links should be accepted or rejected; make decisions on whether or not to look for additional candidate links; make decisions on whether or not a requirement has been satisfied completely by its links; and decide if the tracing process is complete. It is clear that a human analyst must have the final say in all decisions. The key to successful automation lies not in removing the human decision-maker from the loop, but rather, in introducing an automated agent that takes care of the mundane, time-consuming parts of the process and allows the analyst to concentrate on the parts that really require human decision-making. What can be automated, as shown in [20], is the *generation of candidate links* to address items (c) and (d). With this in mind, we move to the identification of the desirable attributes of an effective tracing tool.

Most research in the area of requirements tracing has focused on models of requirements tracing [40] or has looked at recall and precision to assess the accuracy of the applied linking algorithms [3], [31]. To our knowledge, there has not been work published that details the goals for an effective requirements tracing tool [3]. While prior work has been done to define the capabilities required for a requirements **management** tool [29], these capabilities (see Table II) are not appropriate for our tracing tool. The management tool requirements are very far reaching, whereas we are narrowly focused on tracing and even more narrowly focused on dynamic trace generation. In addition to specifying such goals, we provide a validation mechanism for each goal, and then in Sections III and IV demonstrate that our tracing tool satisfies the goals we have addressed to date.

First, we define a requirements tracing tool as a special-purpose software that takes as input two or more documents in the project document hierarchy (without loss of generality we assume that individual requirements in these documents have been successfully defined and are easily extractable) and outputs a traceability matrix, that is a mapping between the requirements of the input documents. In the rest of the paper, we concentrate on the process of forward tracing for a pair of documents — most other requirements tracing tasks can be reduced to this problem.

[3]besides our work in [21].

ID	Capability
1.	Capturing Requirements/IdentificationCapturing
1.1	Input document enrichment / identification
	Using existing document information, aid the user in requirements analysis, identification of requirements, etc.
1.1.1	Input document change/comparison analysis
	The ability to compare/contrast two different versions of a source document
2.	Capturing system element structure
2.1	Graphically capture system structure
2.2	Textual capture of system structure

TABLE II

SAMPLE CAPABILITIES FOR A REQUIREMENTS MANAGEMENT TOOL PER INCOSE [29].

From the perspective of a development manager or a safety manager (in the case of a safety-critical system), the most important attribute that a requirements tracing tool can possess is that its final results are believable and can be trusted. Similarly, the analysts who use the tool should have confidence in the candidate link lists provided by the software (addressing items (c) and (d)). Lack of this quality in a tool might result in an analyst wasting time by searching for additional candidate links. We refer to this attribute as "believability," and it constitutes the first goal.

: **Goal 1: "Believability"** - The requirements tracing tool shall generate candidate links and shall solicit analyst feedback and shall re-generate candidate links based on the feedback such that the final trace shall very accurately reflect the theoretical "true trace." Believability is constituted of three sub-elements, discussed below: accuracy, scalability, and utility.

Accuracy.*:* The extent to which a requirements tracing tool returns all correct links and the extent to which the tool does not return incorrect links.

Scalability. *:* The extent to which the requirements tracing tool is able to achieve accuracy for "small" tracesets as well as "large" tracesets. It has been argued that one obstacle to the transfer of research results to industry is the lack of realism of studies used in controlled experiments, specifically, that these studies are too small, or "toy"-like[47]. Thus, in our opinion, the ability of a requirements tool to scale its performance to the size of the tracing problem, contributes to its believability.

In this context, we define a "small" traceset to constitute 3000 combinatorial links or less. For example, a traceset consisting of 20 high level requirements and 50 low level requirements would have $20 \times 50 = 1000$ combinatorial links. Any traceset with more than 3000 combinatorial links is considered large. We set this "line" for large tracesets based on 16+ years of industry experience of the first author [4] and on proprietary information on how many hours it takes to trace sets of varying sizes. We offer this as a starting point for discussion with other researchers and practitioners.

Utility.*:* The extent to which an analyst believes the tool has helped to achieve good trace results. If the analyst has (justified) confidence in the accuracy and scalability of the tool, the tool possesses utility for the analyst. In addition to analyst belief about accuracy and scalability, other items can impact utility. This is a very subjective item, and we are still in the process of elucidating its sub-elements. Thus far we have defined Operability, Process Enforcement, and Usefulness. Operability is the capability of the software product to enable the user to operate and control it [7]. Process Enforcement refers to the tool implementing tracing

[4] This experience includes: manual tracing; on-going comprehensice assessement of existing tracing technologies and tools; the specification of requirements for a proprietary tracing tool in the mid-80s; oversight of the development of the tool, testing and use of the tool to trace mutliple textual artifacts (including source code) for large mission and/or safety-critical projects including: instrumentation and control systems for nuclear power plants, weapon systems, and manned and unmanned flight systems; the training of and managing of analysts performing tracing using tools (and manually) in the previous organization the first author of over 200 analysts managed by her; the design and execution of industrial tracing experiments ranging back to 1991; and academic work on traceability, using industry datasets, for the past five years.

in such a way that the analyst is guided through the process.

We also consider Usefulness to be a sub-element. At this point, we see this as having a subjective and objective aspect. Subjectively, the user interface contributes greatly to how convenient the tool is to use. Objectively, the tool must generate benefits that convince the analyst that it is better to use the tool than not to use it. For example, if the tool can greatly reduce the amount of decisions an analyst must make, it has generated a savings in effort. This should positively influence the analyst's opinion about usefulness.

Validation mechanism.: The standard measures of accuracy are recall and precision mentioned in Section I and defined formally in Section IV-B. Accuracy can be measured objectively, but only when we have the theoretical "true trace" (i.e., the actual traceability matrix) available. Even when we do not have such an "answer set" a priori, we can build an RTM using the tool, capturing the candidate links returned at each stage. Then, we can compare the candidate links supplied by the tool at each stage to the final RTM (treating it as the answer set).

Precision and recall quantify accuracy in two different, complimentary, ways. In an ideal setting, a list of candidate links is accurate when it contains all the high-low level requirements pairs that trace to each other and does not contain any extra pairs. Recall measures the degree to which the first condition is met, while precision looks at the second. We note a certain asymmetry between the two measures. In general, an imperfection in the list of candidate links can come from two sources: an error of commission — a false positive link was added to the list (Type II error), or an error of omission — a true link was not recognized (Type I error). Errors of comission decrease precision, while errors of omission decrease recall both recall and precision, but generally, have a more drastic effect on recall. As a rule, human analysts are much better in detecting errors of comission (examining a given link and determining whether it belongs to the answer set) than they are in detecting (and rectifying) errors of omission. The latter require understanding that the current set of links is insufficient in some way, followed by a thorough search through the low-level document for missing links. Thus, candidate link lists with very high recall but lower precision are preferable to the candidate link lists with high precision and lower recall. In Section IV-B, we specify precisely what we mean by "excellent", "good," and "acceptable" precision and recall.

For scalability, we must examine the tool's results for both small and large tracesets to determine that the accuracy has not been significantly degraded. Validation of utility requires the study of the users as much as it requires the study of the methods. In addition, we must first establish accuracy and scalability before progressing to the study of the users, thus ensuring that the tool performs in such a way that there is a basis for analyst confidence. The study of users is left for further research [25], [26].

: **Goal 2: "Discernability"**. The requirements tracing tool shall generate candidate links and display their similarity measures in such a way to make it easy for the analyst to discern true links (from the theoretical "true trace") from false links (candidate links that are not really links).

Validation mechanism.: There are several aspects to this goal. In general, we want to ensure that the software communicates information (such as requirement text), process flow (such as what to do next), and results in a manner that facilitates the tracing process. We refer to this as communicability. In addition, we want to ensure that, as the stages of tracing proceed, good links (true links) rise to the top of the candidate link list and that bad links (false links) fall to the bottom. And we want to ensure that the similarity measures given for candidate links reflect the "cut off" line between true and false links. To that end, we define objective measures for all the items above except communicability. "Good links rising" and "bad links sinking" are measured using Lag, defined (informally) as the average number of false positives with higher relevance (a value between 0 and 1 where 1 indicates the highest possible similarity) than a true link in a list of candidate links. The existence of a cutoff is studied using different filtering techniques on the candidate link lists. These measures are formally defined in Section IV-B.

Goal 3: "Endurability:" The requirements tracing tool shall generate candidate links and shall solicit analyst feedback and shall re-generate candidate links based on the feedback such that the process of requirements tracing is not arduous.

Validation mechanism.: Part of Endurability can be measured objectively by examining the time it takes to complete a tracing project using the tool. In addition, the analyst's effort can be measured by the number of mental comparisons (s)he must make. In Section IV-B, we propose a measure called *Selectivity* to capture this. At the same time, Endurability also refers to subjective satisfaction of the analyst with the tool and requires subjective measures and a separate experimental design. This study is left for future research.

A. Study of Methods vs. Study of Users

The goal of our research is to improve requirements tracing during the IV&V process by using IR methods for candidate link generation and analysis. Our intention is not to replace the human analysts, but rather to provide better tools for their use. As such, our concern here is dual. First, we must ensure that the tools we build *are capable of providing accurate results fast*. However, because the final result of the IV&V tracing process must come from a human analyst, we are also interested in *what human analysts do* with the results provided by our tools.

This duality of interest is directly reflected in the high-level goals presented above. Indeed, all three goals, believability, discernability and endurability, have components directly relevant to each of the two interests. We can judge the ability of the software (the methods implemented in it) to deliver results by: examining its accuracy and scalability; by measuring its ability to eventually separate true links from false positives; and by ensuring that no matter how large a tracing task is, only a small fraction of all possible links is being examined. However, by itself, this does not guarantee that a human analyst working with the tool will make the right decisions and produce a correct final trace.

To accommodate these two complimentary interests, we conduct our research in two directions. In this paper, we study the applicability of IR methods to requirements tracing. The primary goal of this paper is to show that these methods are capable of providing good candidate link lists.

We study analyst interaction with tracing software separately. This latter study is predicated upon building automated tools that provide good candidate link lists (otherwise, the tracing process is in danger of turning into a garbage in — garbage out affair). At present, we have conducted a pilot experiment involving three analysts that showed that the issue of analyst interaction with software needs to be studied in more detail. The preliminary report and an in-depth discussion of this issue can be found in [25], [26].

III. Effective requirements tracing with RETRO

A. Why Use Information Retrieval?

The problem of requirements tracing boils down to determining if each pair of requirements from high- and low-level requirements documents are "similar." Stated as such, requirements tracing bears a striking similarity to the standard problem of Information Retrieval (IR): given a document collection and a query, determine those documents from the collection that are relevant to the query. In the forward tracing scenario, high-level requirements play the role of queries, while the "document collection" is made up of low-level requirements (these roles are switched if back-tracing is desired). The key to understanding whether IR methods can aid requirements tracing lies in examining the concept of requirement "similarity." This concept is used by the analysts to determine the trace. We must see if requirements similarity can be modeled, or at least approximated, by the document relevance notions on which different IR algorithms rely.

The major difference in the similarity concepts used by analysts and the measures used in IR algorithms is that human analysts are not limited in their decisions by purely arithmetical considerations. A human analyst can use any tool available in her arsenal to determine the trace, and that may include "hunches," jumping to conclusions, and/or ignoring assignments prescribed by any specific methodology. Such diversity of sources for human decision-making can be both a blessing and a bane to the requirements tracing process. On one hand, it may lead to discovery of hard-to-find matches between the requirements. On the other hand, human analysts do make errors in their work. These errors may be explicit, the analyst discards correct links and keeps incorrect ones, and implicit, the analyst does not notice some of the true links between the documents. Similarity (relevance) measures computed by IR algorithms are not prone to errors in judgment. But they may fail to yield connections that humans might notice despite differences in text.

Even taking this observation into account, there is still enough evidence to suggest that IR methods are applicable. Indeed, the actual procedures employed by an IR algorithm in RETRO and by the analyst working with (for example) the STP tool [19], [34] are very similar. In both cases, the lists of requirements from both document levels are scanned and for each requirement a representation based on its text is chosen. After that, in both instances, matching is done automatically and the analyst then inspects the candidate links.

B. RETRO

In contrast with such comprehensive requirements management tools as DOORS [52], RETRO (REquirements TRacing On-target) is a special-purpose tool, designed exclusively for requirements tracing. It can be used as a standalone tool to discover traceability matrices. It can also be used in conjunction with other project management software: the requirements tracing information is exported in a simple, easy-to-parse XML form. The overall look of the RETRO Graphical User Interface (GUI) (Win32 port) is shown in Figure 1.

At the heart of RETRO lies the IR toolbox (C++): a collection of implementations of IR methods adapted for the purposes of the requirements tracing task. Methods from this toolbox are accessed from the GUI block (Java) to parse and analyze the incoming requirements documents and construct relevance judgments. The Filtering/Analysis component (C++) of RETRO takes in the list of candidate links constructed by any of the toolbox methods and prepares a list to be shown to the analyst. This preparation may involve the application of some cleaning, filtering, and other techniques. The GUI of RETRO guides the entire requirements tracing process, from setting up a specific project, to evaluating the candidate link lists.

At the top of the screen, the analyst sees the list of high level requirements (left) and the list of current candidate links for the selected high level requirement with relevance judgments (right). Below the text of the current pair is displayed. In this case, a software requirement specification (SRS) requirement is shown in the High-level text window and a design specification element is shown in the Low-level text window. At the bottom, there are controls allowing the analyst to make a decision on whether the candidate link under consideration is, indeed, a true link [5]. This information is accumulated and, upon analyst request, is fed into the feedback processing module (C++). The module takes the results of analyst decisions and updates the candidate link discovery process (discussed below in III-C) consistent with the changes. If needed, the IR method is re-run and the requirements tracing process proceeds into the next iteration.

C. Information Retrieval methods in RETRO

The IR toolbox of RETRO implements a variety of methods for determining requirement similarity. For this study we have used two IR algorithms implemented by us previously [20]: *Tf-Idf vector retrieval* and *vector retrieval with a simple thesaurus* and one newly implemented method, *Latent Semantic Indexing* (*LSI*)[11]. Tf-Idf-based methods were selected for their simplicity and efficiency. LSI is a dimensionality-reduction method, which allows one to capture the similarity of underlying concepts, rather than simple keyword matches. For traditional Information Retrieval tasks that involve collections of millions of documents, LSI is inefficient, but requirements tracing tasks are much smaller. LSI has been successfully applied by Marcus and Maletic [31] to tracing of code to requirements; we investigate here, whether it also holds for requirements-to-requirements traceability. To process user feedback, we have used the Standard Rochio [5] method for the vector models and a variation of it for the LSI [11]. The methods used are briefly described below.

1) Tf-Idf retrieval.: Let $V = \{k_1, \ldots, k_N\}$ be the vocabulary (list of keywords) of a given document collection. Then, a vector model of a document d is a vector $(w_1, \ldots\ldots, w_N)$ 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 $idf_i = \log_2\left(\frac{n}{df_i}\right)$, where n is the number of documents in the collection and df_i is the number of documents in which keyword k_i occurs. Given a document vector $d =$

[5]Current version of RETRO also has a browse mode in which the analyst can simply read requirements from both high- and low-level documents and make decisions to add any pair of requirements to the trace. This mode is not shown on Figure 1 and had not been used in the experiments

$(w_1, \ldots, w_N)$ 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:

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

2) Tf-idf + Simple Thesaurus.: The second method used in [20] extends the TF-IDF model with a simple thesaurus of terms and key phrases. From our prior industry experience, we knew that many software engineering specification pairs (e.g., a design document and a requirement specification) are written using different terminology, acronyms, and technical "lingo." We knew that a method such as TF-IDF that can only identify relevance based on matching keywords would suffer due to this. Hence, we decided to adopt the thesaurus approach to assist in matching elements, that have been written differently, from one specification to those from another specification. 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 coefficient between them. 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 document/query vectors $d = (w_1, \ldots ..., w_N)$ and $q = (q_1 \ldots, ..., q_N)$, the similarity between d and q is computed as:

$$sim(d, q) =$$

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

3) Latent Semantic Indexing.: We wanted to examine a more sophisticated technique. We selected LSI because we knew that our small datasets (as compared to typical IR datasets) would not create the performance issues that one may see with larger datasets. Latent Semantic Indexing (LSI) [11] reduces the dimensionality of the document-by-term matrix used in IR by replacing it with a matrix of orthogonal components obtained as a result of Singular Value Decomposition (SVD) [11] of the original matrix X with dimensions $M \times N$: $X = TSD^T$.

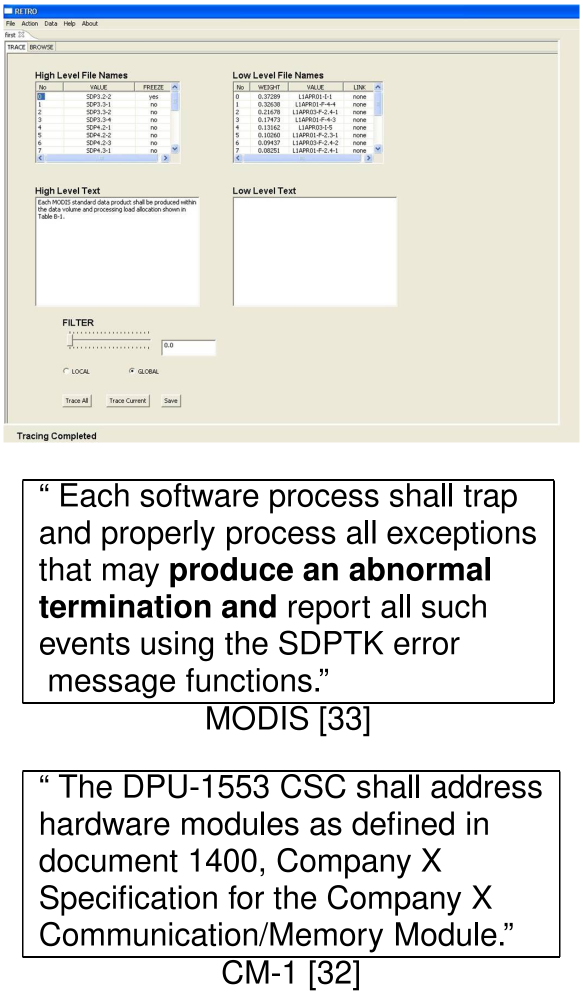


Fig. 1. A screenshot of RETRO and sample requirements for MODIS and CM-1 datasets.

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 $X' = TS_k D^T$ is then used in place of X. In our experiments, matrix X was constructed out of both high-level and low-level requirements together, in order to capture the underlying structure that unifies both requirements documents.

4) Incorporating Relevance Feedback.: In [20] we have considered the application of IR methods to the tracing problem, in which the IR method was run once and its output was measured for accuracy (recall and precision). Yet, we observe that when tracing is a part of the V&V or IV&V process, the analyst performing the task **must** inspect the output of the IR method and render a "link"/"no link" decision for each candidate link found[6]. It is possible, however, to provide some help to the analyst in the process of inspection, by using decisions already rendered by the analyst to (*potentially*!) (a) automatically fix some errors of commission; (b) automatically fix some errors of omission and (c) restructure candidate link lists in a way that true links are visited earlier. This procedure is called relevance feedback analysis.

Relevance feedback analysis is a technique to utilize user input to improve the performance of the retrieval algorithms. Relevance feedback techniques for TF-IDF methods adjust the keyword weights of query vectors according to the relevant and irrelevant documents found for them, as supplied by the analyst. We selected this because we knew that tracing is a highly interactive, repetitive process, where users examine various element pairs and decide if they are related. We felt that an ability to capture the analyst's opinion and use it to improve the results shown to them would be very useful. We define the process next. Let q be a query vector, and D_q be a list of document vectors returned by some IR method given q. Further, assume that D has two subsets: D_r of size R of documents relevant to q and D_{irr} of size S of irrelevant documents that have been indicated by the analyst. Note that D_r and D_{irr} are disjoint, but do not necessarily cover the entire set D_q. We use the Standard Rochio [5] feedback processing method, which modifies the query vector for the next iteration of the IR procedure:

$$q_{new} = \alpha \cdot q + \left(\frac{\beta}{R} \sum_{d_j \in D_r} d_j \right) - \left(\frac{\gamma}{S} \sum_{d_k \in D_{irr}} d_k \right).$$

Intuitively, query q is adjusted by adding to its vector a vector consisting of the document vectors identified as relevant, and subtracting from it the sum of all document vectors identified as false-positives. The first adjustment is designed to potentially increase recall. The second adjustment can potentially increase 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. Once the query vectors have been recomputed, the selected IR algorithm is re-run with the modified query vectors. This cycle can be repeated until the analyst is satisfied with the results.

[6]Some decisions might be rendered in bulk, e.g., excluding all links with relevance less than 0.025. The analyst also has the option of conducting manual search for missing links.

IV. Evaluation

This section presents an overview of the experiment conducted and measures collected as well as a detailed look at the experiments by dataset.

A. Datasets Used

Our experiments have been conducted using two datasets: a small MODIS dataset and a large CM-1 dataset[7] and the Promise website [39], [51]. The CM-1 dataset is also available on the Promise website [39], [51]. The MODIS dataset has been constructed from two publically available high-level requirements [33] and low-level requirements [30] documents for NASA's Moderate Resolution Imaging Spectrometer (MODIS). From these two documents, we have selected 19 high-level and 49 low-level elements. A typical requirement (high- or low-level) is one to two sentences in length. The Flesch Reading Ease of a typical MODIS requirement is 32.1 and the Flesch-Kincaid Grade Level is 12 [15],[16]. These measures examine the relative "complexity" of the text. A sample high-level requirement is shown in Figure 1. *The "theoretical true trace," i.e., the list of all true links in the dataset,* ***has been constructed manually and verified***. The list includes 41 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) [32]. The text of the documents has been altered by NASA prior to public release in order to hide the identity of the instrument. A typical requirement is one to two sentences in length (see Figure 1). A typical design element is several paragraphs in length, with paragraphs averaging four to five sentences in length. The Flesch Reading Ease of a typical CM-1 requirement is 40.5 and the Flesch-Kincaid Grade Level is 12 [15],[16].

[7]The MODIS dataset is available on the Software Engineering Empirical Website (SEEWEB)[35]

The CM-1 dataset has 235 high-level requirements, 220 design elements, and the final traceability matrix (theoretical true trace) contains 361 true links.

We performed a rigorous manual verification of the true trace of both datasets. For MODIS, we started with the RTM provided in the high-level MODIS requirement specification. Two senior analysts then manually verified the trace. For CM-1, we used four junior analysts to manually verify a trace that was generated by RETRO. After they completed their task, we had a senior analyst verify that trace. He consulted with a second analyst. At the second stage, the analysts worked with the text of the requirements and design documents and the traces generated by junior analysts. The senior analysts have very carefully studied both documents and the trace for both errors of commission and errors of omission. Our assessement of the process lead us to believe that because of significant manual tracing effort undertaken at the second stage of the process, any potential bias introduced by the use of RETRO to jump-start the tracing process had been remedied at the second stage.

For both datasets, when we encountered "borderline" cases (might be links, might not be links depending on interpretation), we added it as a link in the traceset. In that, inclusive, way, we err on the safe side. As we mentioned above, correcting errors of omission is harder than correcting errors of omission. Thus, we opted to have the "borderline" links visible in the answer set, rather than "hidden" outside of it.

B. Measures Used

The key measures of success of any Information Retrieval task are *recall* and *precision*. Their weighted harmonic mean, called *f-measure* is used when a single measure is needed to describe the accuracy. In addition, in [21] we have introduced a number of so called *secondary measures* that allow us to track progress of the feedback process even when precision and recall numbers do not change significantly. In this section, we briefly introduce all measures used in the tracing tests and discuss their relationships with different requirements from Section II.

Let the requirements tracing project consist of a set $\mathcal{H}$ of a high-level requirements, $|\mathcal{H}| = M$, and a set of low-level design elements (requirments) $\mathcal{D}$ of size N. For a given high-level requirement q, let there be R_q true links between q and the low-level elements. Let an IR algorithm return n_q candidate links, out of which r_q are true.

Recall. Recall measures the percentage of true links found by IR algorithms, i.e., given a requirement q, the recall for this requirement is: $recall = \frac{r_q}{R_q}$. Our main measure will be overall recall of the algorithm on all requirements, i.e., the percentage of all true links recovered by an algorithm:

$$recall = \frac{\sum_{q \in \mathcal{H}} r_q}{\sum_{q \in \mathcal{H}} R_q}.$$

For a trace-focused IR algorithm to perform well, overall recall must be high.

Precision. Precision measures the accuracy of the returned candidate link list. Given a requirement q, the precision is $precision = \frac{r_q}{n_q}$. Our main precision measure is the overall precision for the entire requirements document:

$$precision = \frac{\sum_{q \in \mathcal{H}} r_q}{\sum_{q \in \mathcal{H}} n_q}.$$

F-measure. F-measure is a harmonic mean of precision and recall. Achieving high precision and high recall is a balancing act (as precision increases, recall tends to decrease and vice versa), and f-measure represents the balance — the max of f-measure indicates the "best" achievable combination of precision and recall. F-measure can be weighted - tilting the balance towards one of its two components. A weighted f-measure is computed as follows:

$$f = \frac{1 + b^2}{\frac{b^2}{recall} + \frac{1}{precision}}.$$

Here, $b = 1$ means recall and precision are equally important, $b < 1$ means precision is more important, and $b > 1$ means recall is more important.

Selectivity. In general, when performing a requirements tracing task manually, an analyst has to vet $M \times N$ candidate links - i.e., perform an exhaustive search. Selectivity measures the improvement of an IR algorithm over this number:

$$Selectivity = \frac{\sum_{q \in \mathcal{H}} n_q}{M \cdot N},$$

The lower the value of selectivity, the fewer links that a human analyst needs to examine[8]

Lag. Lag is a non-parametric measure for evaluating the level of separation between true links and false positives in a candidate link list.

Definition 1: Let q be a requirement and (q, d) be a true link returned by an IR method in the list of candidate links for q. The Lag of the link (q, d), denoted $Lag(q, d)$, is the number of *false positive links* (q, d') that have higher relevance score than (q, d).

Informally, we compute the Lag of a true link by counting the number of false positives above it in the list of candidate links. Let C be the set of all candidate links returned for all requirements and let T and F be the sets of true links and false positives respectively: $T \cup F = C$. The total Lag of C is the average of the Lags of the true links in C:

$$Lag = \frac{\sum_{(q,d)\in T} Lag(q, d)}{|T|}.$$

Lag specifies, on average, how many false positives are found in the candidate link lists above true links. The lower the Lag, the higher the separation between true links and false positives.

The specifics of the requirements tracing process requires us to establish clear boundaries of quality of methods based on the values of the measures. In Table III, we show the values of three of our measures, precision, recall, and Lag, that are deemed *acceptable*, *good*, and *excellent*. Note, that these values *relate to the evaluation of the quality of the candidate link lists* ***produced by the automated methods***. We do not apply such criteria to selectivity because by itself it is not a measure of quality of the method. These settings have been derived from the industrial experience of the first author in performing and validating many traces. This required much work with RTMs of varying quality levels as well as candidate link lists of varying quality levels that were generated using manual and semi-automated means. The quality levels have been generated to represent *varying levels of analyst effort that is required*. We estimate that candidate link lists with excellent recall and precision require relatively little effort on an analyst's part, whereas non-acceptable results mean that the analyst must spend so much time that they may as well perform the work manually. Note that these is our first attempt to "draw a line in the sand" and we welcome feedback from other researchers and practitoners. In particular, the desire to find empirically where the lines are drawn is one of the main motivations behind our planned study of the users [25], [26].

Measure	Acceptable	Good	Excellent
Recall	60% — 69%	70% — 79%	80% – 100%
Precision	20% – 29%	30 — 49%	50% — 100%
Lag	3 – 4	2 — 3	0 — 2

Requirement	Measures
Believability::Accuracy	*precision, recall, f-measure*
Believability::Scalability	*precision, recall, f-measure*
Believability::Utility	*selectivity*
Discernability	*Lag*
Endurability	*selectivity*

TABLE III
CLASSIFICATION OF RESULTS AND RELATIONSHIP BETWEEN MEASURES AND REQUIREMENTS.

In Section II, a set of high-level goals for a tracing tool were presented. We also defined measures for evaluating IR algorithms and the behavior of the tool. The measures can be used to help us evaluate RETRO's satisfaction, or lack thereof, of each of the high level goals. Table III depicts how the measures relate to the high-level goals. Recall, precision and f-measure assist with the evaluation of the accuracy and scalability sub-requirements of believability. Lag assists solely with assessment of the discernability requirement. Discernability deals with the analyst being able to differentiate easily between relevant and irrelevant candidate links. By measuring the number of false positive candidate links above true links in candidate link lists, Lag also assists with evaluating discernability. Finally, selectivity assists us in evaluating a portion of the endurability and believability goals. If we reduce the amount of work that an analyst needs to do in order to complete a trace, we assist with their ability to "endure" the task. Note that we want low selectivity in addition to high recall.

C. Execution of Experiments

We have conducted a battery of experiments, described in this section, on both MODIS and CM-1 datasets. The main goals of our experiments were:

[8]The term "selectivity" owes its name to a similar measure from database theory. There, given a database query, its selectivity is the percentage of rows in a table that are retrieved [50].

(a) determine whether RETRO is capable of producing accurate tracing results; (b) determine whether RETRO is capable of separating true links from false positives as a result of a feedback process; and (c) determine if RETRO scales well.

Our experiments were conducted in the following manner. First, the lists of high- and low-level elements were extracted from the source documents and put in a format readable by RETRO[9]. After that, both high- and low-level elements were parsed and stemmed using Porter's algorithm [5]. In addition, common stopwords such as "and," "a," "the," etc. were removed. The resulting keyword stream was then passed to the specific IR method for the purpose of creating vectors of term weights. This completed the preparation stage.

Once the vectors were created, the selected IR method was invoked to produce lists of candidate links for each high-level element. This list was then passed to the *feedback simulator*. The feedback simulator was provided with a copy of the answer set and tasked with simulating *ideal* analyst feedback - i.e., the feedback provided by the simulator was always correct. We studied 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, 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, that updated the query vectors and passed control back to the IR method for the next iteration.

Each experiment was run for eight iterations. For each of the datasets, we tested four methods: TF-IDF, TF-IDF+Thesaurus, LSI, and LSI+Thesaurus, each with all four feedback strategies. For LSI and LSI+Thesaurus, we also altered the number of dimensions in the reduced matrix. For MODIS, all results shown are for 10 dimensions. For CM-1, all results are shown for 100 (out of 455 possible) dimensions. At each iteration, we also produced lists of candidate links with relevance higher than one of the predefined levels: 0.05, 0.1, 0.15, 0.2 and 0.25 (this process is calles "filtering").

The results from each iteration of each test run were archived and later run through our analysis tool (part of the RETRO toolkit) which compared the candidate link lists to the answer set and computed precision, recall, f-measure, selectivity, Lag, and effects of filtering.

Below, we report on the most interesting results obtained in our experiments. In particular, we limit our reporting to Top 2 feedback strategy. In our experiments, we found it to be a good ballance of quality of results and amount of feedback per iteration. Results for Top 1 strategy were significantly worse, while results for Top 3 tended to be very similar to those of Top 2, occasionally reaching the same precision/recall numbers one or two iterations earlier.

D. Analysis of Experimental Results

Figures 2—9 show the results of some of our experiments. In Figures 2 and 3, we show the recall vs. precision graphs for four methods running on the MODIS dataset. We observe that all four methods achieve good recall and precision numbers, with TF-IDF+Thesaurus showing the best combination of precision and recall at filter levels of 0.1 and 0.15. It should be noted that all filtering levels exhibit similar behavior, with a slow increase in recall that picks up and a steady increase in precision. Overall, this means that the use of feedback resulted in some errors of omission being fixed automatically (increased recall), as well as many false positives being automatically excluded from the candidate link lists (increased precision). LSI appears to underperform as compared to TF-IDF-based methods.

Results for the CM-1 dataset are shown in Figure 4. Thesaurus-based methods produced almost identical results to their base methods, so we show only the recall vs. precision plots for TF-IDF and LSI. Both methods appear to achieve similar recall levels for similar filters, but TF-IDF exhibits higher precision (best result of 0.55 versus 0.38).

Figures 5 and 6 plot the f-measure for the MODIS and CM-1 datasets. We elected to use the value $b = 2$ for the f-measure weighting parameter, meaning that we consider recall to be twice as important as precision[10]. In Figure 5, we see the change in f-measure during the feedback process at each filter level for the MODIS dataset. For TF-IDF (Figure 5.(a)), f-measure starts between 20 and 30% at all

[9]In the experiments, each requirement/design element was stored in a separate file, with filename serving as its unique identifier; at present, RETRO works with other formats as well.

[10]$b = 2$ is a standard value for the case when recall is valued higher than precision.

filter levels, and gradually improves to the vicinity of 60-70% for all non-zero filters. For TF-IDF + Thesaurus (Figure 5.(b)), f-measure behaves even better, starting at 35-40% and ending around 78-86% for non-zero filters. F-measure for TF-IDF for the CM-1 dataset (Figure 6.(a)) exhibits different behavior. For the non-filtered case, as well as for filters of 0.05 and 0.1, it shows no significant change from its starting value ($\sim$ 8%, 19% and 33% respectively). For the filters of 0.15 and 0.2, the f-measure improves throughout the feedback process, eventually overtaking 50%. Figure 6.(b) compares the behavior of the f-measure for different methods over the MODIS and CM-1 datasets for the filter of 0.1.

Figure 7 summarizes our findings about the accuracy of RETRO with respect to the two datasets studied. The graphs show the recall-precision "footprint" of RETRO for the MODIS (a) and CM-1 (b) datasets using Top 2 feedback strategy. The "footprint" is a scatterplot of all precision-recall values achievable by any method/filter/iteration combination (we show all four methods for MODIS for filter values of 0 – 0.2 and TF-IDF and LSI for CM-1 for filter values of 0 - 0.4). The lines on the plot show the acceptable, good, and excellent boundaries for precision and recall as specified in Table III. We can see that RETRO achieves excellent combinations of precision and recall for MODIS and, in general, a multiple method/filter combination allows us to obtain good or better accuracy. At the same time, we see, that RETRO performs only marginally well for CM-1. Acceptable precision-recall have been achieved, and there are a few points in the good recall-acceptable precision range. However, there is only a single point in acceptable recall-good precision range. The plot very clearly shows that RETRO routinely achieves high recall at low precision or high precision at mediocre recall.

While RETRO could not produce good recall - good precision candidate link lists for the CM-1 dataset, we can evaluate its performance using selectivity instead of precision. Figure 9 shows recall-vs.-selectivity plots for TF-IDF method over the MODIS (a) and CM-1 (b) datasets. We note here that while RETRO could not construct good precision - good recall lists on CM-1, it successfully constructs excellent recall - low selectivity lists. Indeed, at filter level of 0.1, we can obtain recall of around 85% with selectivity around 5-6%.

Finally, Figure 8 documents the changes in Lag for various method/filter combinations for both the CM-1 and MODIS datasets. We see that for the majority of plotted runs, Lag tends to decrease to the level of 0-2. Lag behavior improves when filtering is applied (compare, for example, the behavior of CM-1 TF-IDF and LSI runs with no filter and with filter of 0.1). In general, we conclude that RETRO is capable of achieving good-to-excellent separation between true links and false positives in the candidate link lists.

1) Evaluation Summary: We now examine RETRO in terms of the high level goals presented in Section II. **Accuracy**, a sub-goal of *believability*, is measured using precision, recall, and f-measure. Using the classifications from Table III, the figures above indicate that excellent recall and precision can be achieved with TF-IDF for the MODIS dataset. LSI can achieve acceptable recall with excellent precision for this dataset also. We can achieve good recall with acceptable precision or acceptable recall with good precision for TF-IDF for the CM-1 dataset. We can achieve good recall with acceptable precision using LSI for CM-1. Recalling earlier discussions, recall is of most importance to us in tracing requirements. Therefore, overall, it appears that RETRO meets the acuracy sub-goal. At first glance, it appears that RETRO does not achieve the **scalability** sub-goal of believability. The LSI method does not work well on CM-1. The f-measure does not look promising for the CM-1 dataset. However, the recall and precision for CM-1 are acceptable for TF-IDF: recall slightly above 60% (acceptable) with precision at 42% (good) and recall at 75% (good) with precision at 20% (acceptable). Also, selectivity is quite low while recall is high for CM-1 (85% recall with 5 - 6% selectivity). Though we'd like for precision to be higher, we still see that that the M x N sized collection of elements that must be examined for tracing (e.g., M high level requirements and N design elements) has been drastically reduced, with a very high probability that all true links are shown in the list that has been retrieved (high recall). This represents a tremendous effort savings for an analyst, going from M x N worth of work to 0.05 *(M x N) worth of work. Therefore, it appears that RETRO scales. Lag decreases greatly over subsequent feedback iterations, as shown above. This shows that the *Discernability* goal has been met. The true links rise to the top of

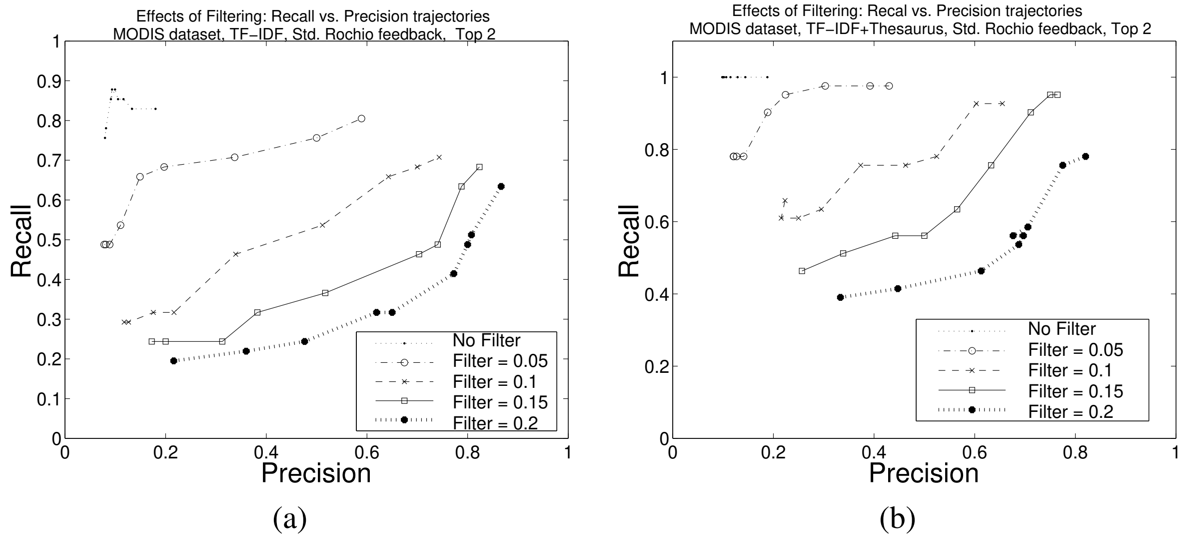


Fig. 2. Results for MODIS dataset, TF-IDF (a) and TF-IDF+Thesaurus (b), Top 2 feedback.

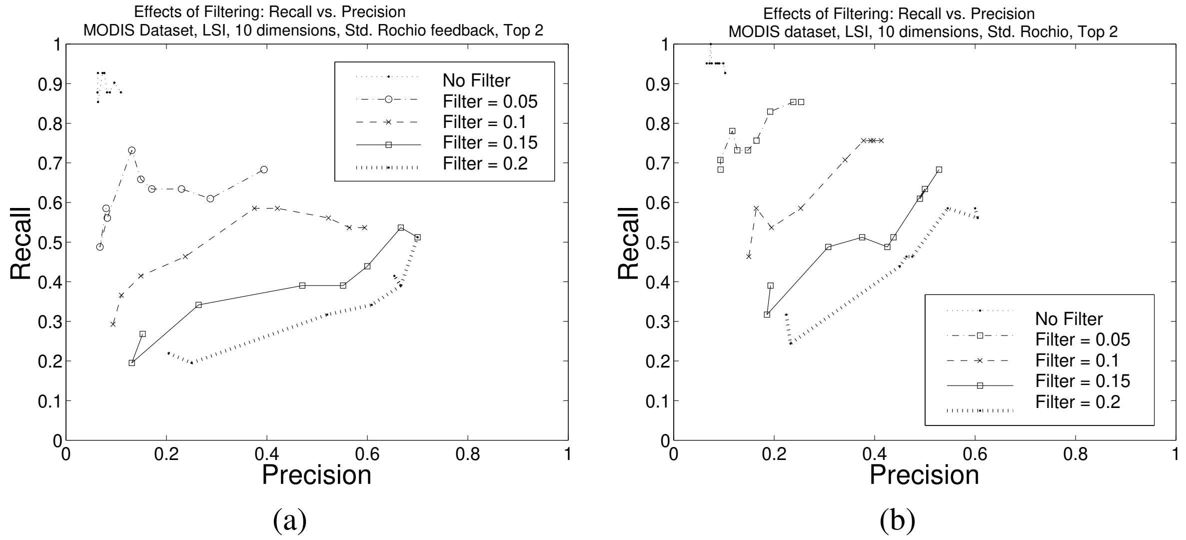


Fig. 3. Results for MODIS dataset, LSI (a) and LSI+Thesaurus (b), Top 2 feedback.

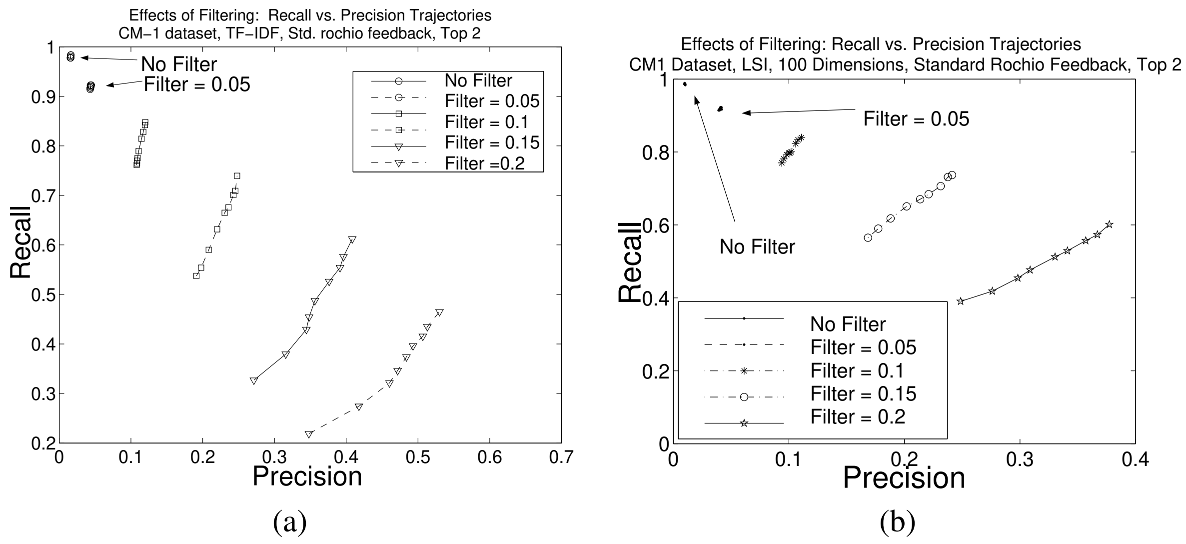


Fig. 4. Results for CM-1 dataset, TF-IDF (a) and LSI (b), Top 2 feedback.

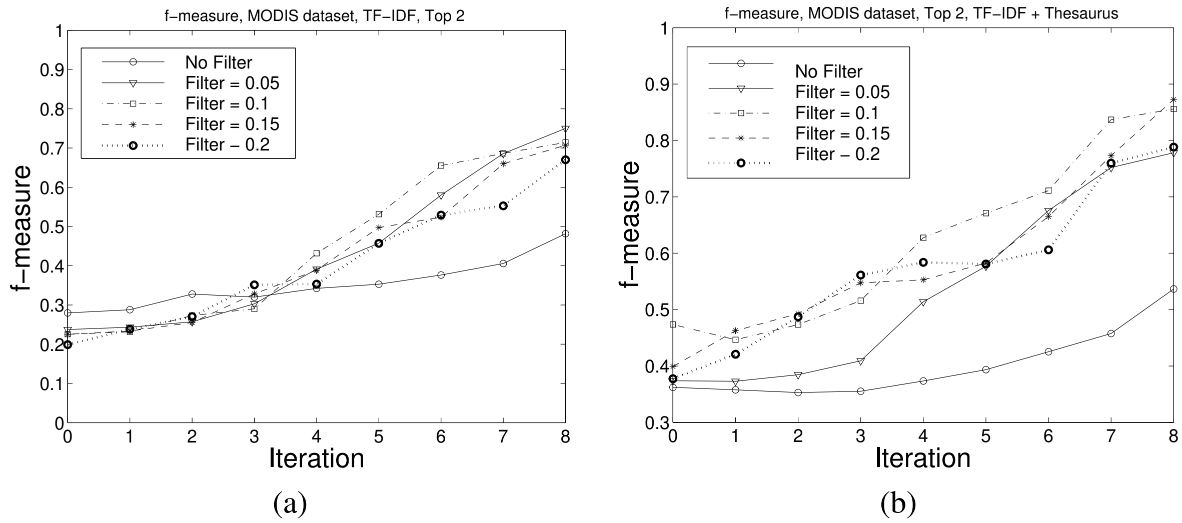


Fig. 5. F-measure, MODIS dataset, TF-IDF (a) and TF-IDF + Thesaurus (b), Top 2 feedback.

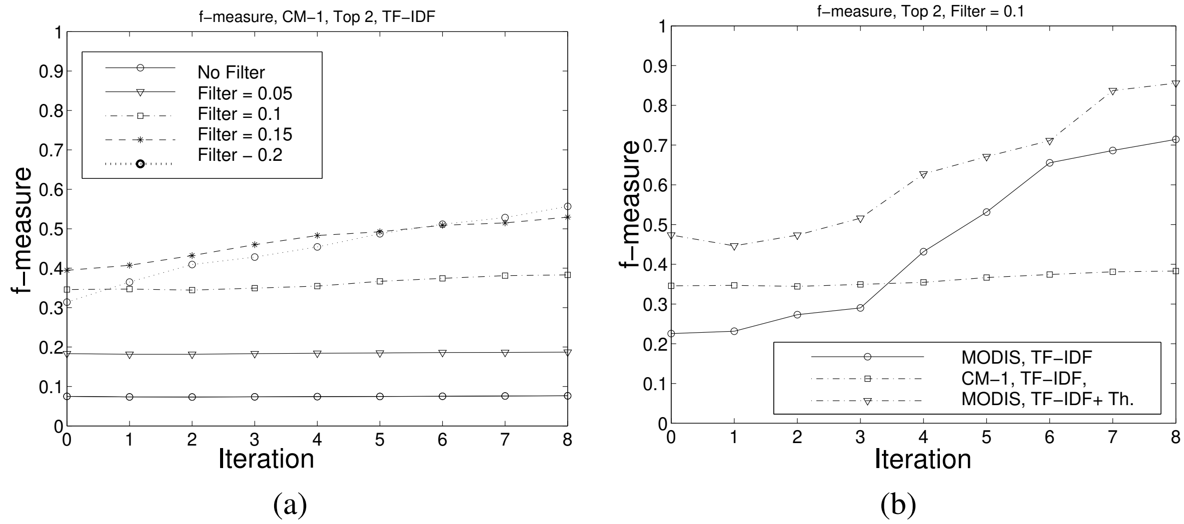


Fig. 6. F-measure, CM-1 dataset, TF-IDF (a) and comparison for MODIS and CM-1 (b), Top 2 feedback.

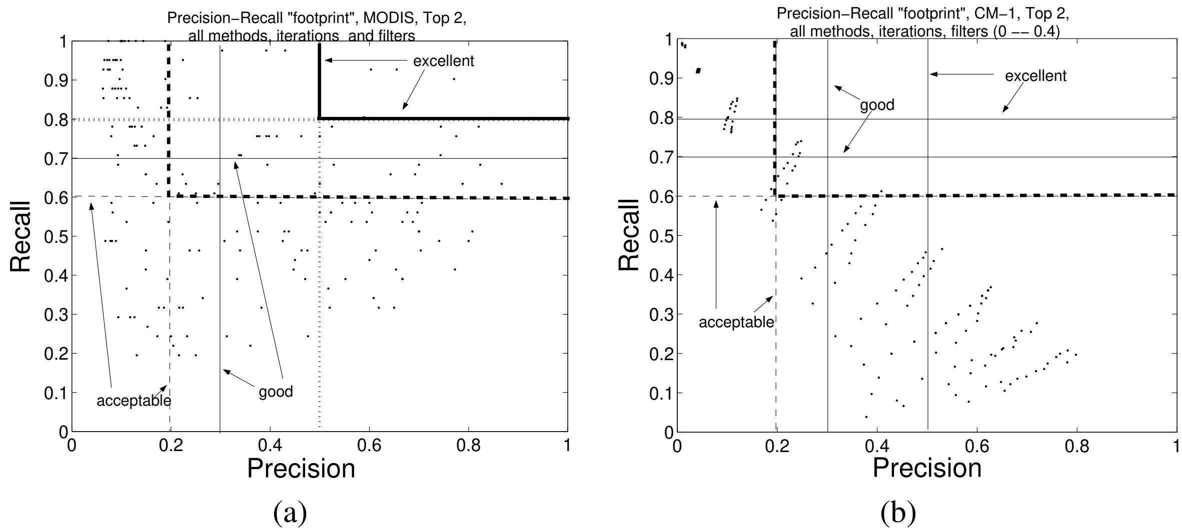


Fig. 7. Precision-Recall footprints, MODIS (a) and CM-1(b) datasets, Top 2 feedback.

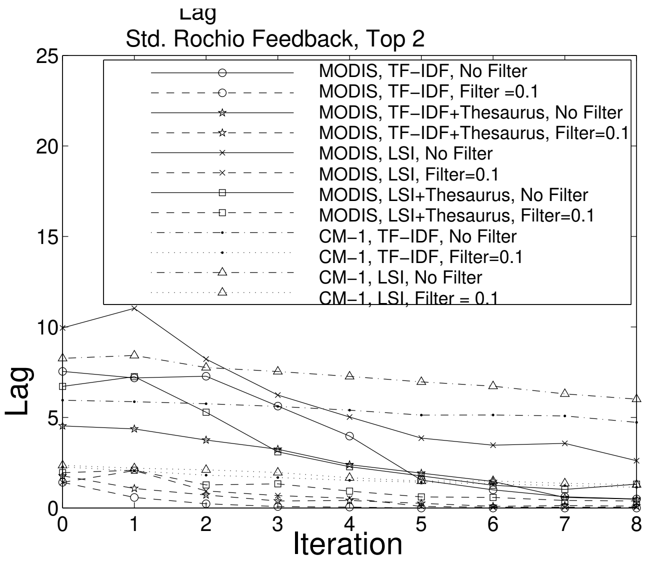


Fig. 8. Lag, Top 2 feedback.

the list while the false links fall. *Endurability* has been addressed since selectivity drops significantly for MODIS and CM-1 for TF-IDF, getting close to 0 for CM-1 with acceptable recall, and close to 5% for MODIS with recall of 70%.

Though as researchers we strive for higher accuracy, better scalability, etc., we feel justified in asessing RETRO as a success from a practitioner's view. Having performed manual traces as well as traces with various keyword matching techniques, etc., it is clear that the amount of effort required of the tracing analyst has been substantially reduced. Also, the need to search for relevant items has been drastically reduced. Our experience in industry has shown that analysts will render a decision on links presented to them, but may rarely or never go and "hunt" for potential links that were not shown to them. So the need for high recall is obvious, and the "secondary" nature of precision is also apparent. Selectivity is very important because it tells us how much smaller the potential task has become for the analyst. If the potential M x N tracing job gets smaller and smaller with each subsequent provision of feedback by the analyst, closing in on 0 - 5%, then RETRO is achieving desired results. The selectivity graphs for RETRO show this to be the case.

V. Related Work

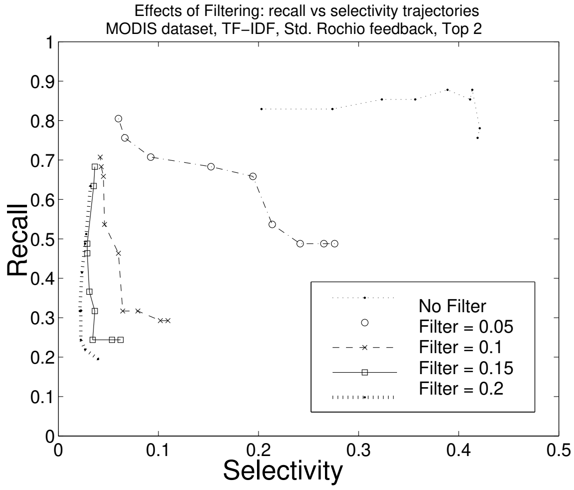
Related work is organized into two sections; very early, qualitative work in tracing (section 5.1) and the application of information retrieval methods to tracing (section 5.2).

A. Requirements tracing

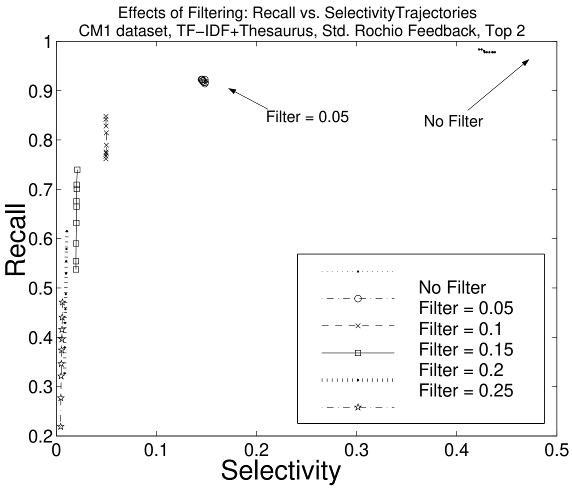
We start by providing an overview of the research conducted on requirements tracing and traceability in the past 10—15 years. This early work predates the use of information retrieval methods for tracing and for candidate link generation. It also pre-dates the use of currently accepted measures to quantitatively assess the accuracy of tracing methods. This makes direct comparison of this work with ours infeasible. Hence we present the work to provide historical context from which research on automating tracing has emerged. In Section 5.2, we discuss recent work on automating tracing and traceability and compare that work to ours.

Early research in traceability falls into a number of areas: early tracing using DBMS, study of the traceability process, research on change tracing, research on traceability rules and some general work on traceabilty. We describe each area in turn.

Early tracing using DBMS.: We have been tackling the requirements tracing problem for many decades. In 1978, Pierce [37] designed a requirements tracing tool as a way to build and maintain a requirements database and facilitate requirements analysis and system verification and validation for a large Navy undersea acoustic sensor system. Hayes et al [19] built a front end for a requirements tracing tool called the Software Automated Verification and Validation and Analysis System (SAVVAS) Front End processor (SFEP). This was written in Pascal and interfaced with the SAVVAS requirements tracing tool that was based on an Ingres relational database. SFEP allows the extraction of requirement text as well as the assignment of requirement keywords through the use of specified linkwords such as "shall," "must," "will," etc. These tools are largely based on keyword matching and threshold setting for that matching. Several years later, the tools were ported to hypercard technology on Macs, and then to Microsoft Access and Visual Basic running on PCs. This work is described by Mundie and Hallsworth in [34]. These tools have since been further enhanced and are still in use as part of the Independent Verification and Validation (IV&V) efforts for the Mission Planning system of the Tomahawk Cruise Missile as well as for several NASA Code S science projects.



(a)



(b)

Fig. 9. Selectivity, MODIS dataset (a) and CM-1 dataset (b), TF-IDF, Top 2 feedback.

Traceability process: Gotel and Finkelstein [17] present the requirements traceability problem based on their empirical studies. They analyze the difference between pre-traceability and post-traceability, and demonstrate the necessity of increased focus on pre-traceability to improve the requirements traceability process. Pohl [36] presents an approach for tracing requirements to their origins called pre-traceability. Pohl presents a three dimensional framework for a requirements engineering trace repository to enable selective trace retrieval and to enable automated trace capture. Abrahams and Barkley [1], Ramesh [40], and Watkins and Neal [54] discuss the importance of requirements tracing from a developer's perspective and explain basic concepts such as forward, backward, vertical, and horizontal tracing. Ramesh et al examine reference models for traceability. They establish two specific models, a low-end model of traceability and a high-end model of traceability for more sophisticated users [41].

Change tracing.: Ramesh and Dhar [42] developed a conceptual model, called Representation and MAintenance of Process knowledge (REMAP), that captures process knowledge to allow one to reason about requirements and the effects of changes in the system design and maintenance. Casotto [9] examined run-time tracing of the design activity. Her approach uses requirement cards organized into linear hierarchical stacks and supports retracing. Cleland-Huang et al [10] propose an event-based traceability technique for supporting impact analysis of performance requirements. Data is propagated speculatively into performance models that are then re-executed to determine impacts from the proposed change.

Traceability rules.: Spanoudakis [48] traces textual requirements to object models using heuristic traceability rules. Three types of beliefs are described and measured: belief in rule satisfiability, belief in rule correctness, and belief in traceability relation. Based on the values of these beliefs, traceability rules and relations are modified. Spanoudakis et al [49] use two types of rules, namely, requirements-to-object-model (ROTM) and inter-requirement traceability (IREQ) rules, to automate the generation of traceability relations. They describe a prototype system that incorporates a traceability rule which interprets ROTM and IREQ traceability rules and generates traceability relations. Egyed et al, in [14], discuss a technique for automating requirements tracing using Trace Analyzer [13]. They take known dependencies between software development artifacts and "common ground" such as source code. They then build a graph based on the common ground and its overlap with the artifacts. The graph structure is manipulated iteratively using large numbers of rules. For them, trace dependency implies that two artifacts relate to at least one common node in the graph. The usage cases are tested against code to find trace dependencies. Egyed et al. [14] focus on dependencies between requirements and code and between model elements and code, whereas the current work focuses on dependencies between unstructured, textual software artifacts.

Miscellaneous other research on traceability.: Hoffman et al. [28] present a requirements catalog for requirements management (RM) tools. The catalog helps users compare and select requirements management tools based on functional requirements met by the tools. Requirements for requirements management tools are defined from the point of views of developers, project administrators and tool administrators. Requirements address areas such as information model, views of the data, format, change management, documentation of history, baselines, tool integration, document generation, workflow management, installation and administration of projects, database, encryption, etc. The requirements pertaining to tracing are fairly high level. As RETRO currently focuses on candidate link generation, the Hoffman et al. requirements do not yet apply. As we expand to investigate other areas of tracing, we will re-examine the Hoffman et al. paper. Tsumaki and Morisawa [53] discuss requirements tracing using UML. Specifically they look at tracing artifacts such as use-cases, class diagrams, and sequence diagrams from the business model to the analysis model and to the design model (and back) [53].

There have also been significant advances in the area of requirements elicitation, analysis, and tracing. Work has been based on lexical analysis, such as extraction and analysis of phoneme occurrences to categorize and analyze requirements and other artifacts [45]. Bohner's work on software change impact analysis using a graphing technique may be useful in performing tracing of changed requirements [7]. Anezin and Brouse advance backward tracing and multimedia requirements tracing in [2], [8].

Pinheiro and Goguen [38] describe a tool called Traceability of Object Oriented Requirements (TOOR). TOOR permits three types of tracing: selective tracing, interactive tracing, and nonguided tracing.

The work mentioned above presents a historical perspective of requirements tracing. The majority of the research concentrates on understanding the tracing process and discovery of traceability rules. This has served to establish a general framework for research on traceability. However, our specific interest (as described in this paper) is much more narrow and concentrates on applying Information Retrieval (and similar) techniques to automate parts of the tracing process. In the next section we concentrate on related work in this research direction.

B. Information Retrieval in Requirements Analysis

In general, the software tools described above address the overall problem of requirements management during the lifecycle of a software project. Their requirements tracing components typically rely, one way or another, on manual keyword assignment - a long and arduous process. With time, practitioners realized the potential benefits of, and the researchers started working on, methods for automating the requirements tracing process. Of the many methods examined, Information Retrieval techniques appear to offer much promise for this automation.

Two research groups worked on requirements-to-code tracebility. Antoniol, Canfora, De Lucia and Merlo [3] considered two IR methods: probabilistic IR and vector retrieval (TF-IDF). They have studied the traceability of requirements to code for two datasets. In their testing, they retrieved the top i matches for each requirement for $i = 1, 2, \ldots$ and computed precision and recall for each i. Using improved processes, they were able to achieve 100% recall at 13.8% precision for one of the datasets. In general, they have achieved encouraging results for both TF-IDF and probabilistic IR methods. Following [3], Marcus and Maletic [31] applied the latent semantic indexing (LSI) technique to the same problem. In their work, they used the same datasets and the same retrieval tests as [3]. They have shown that LSI methods show consistent improvement in precision and recall and were able to achieve combinations of 93.5% recall and 54% precision for one of the datasets.

Antoniol, Caprile et al [4] performed an experiment that examined a process for recovering "as is" design from code, comparing recovered design with the actual design and helping the user to deal with inconsistency. The process evaluated consisted of a number of steps: code and Object Model Technique (OMT) [44] design is translated to Abstract Object Language (AOL) using a tool; AOL is parsed to produce an Abstract Syntax Tree (AST) by a tool; a relations traceability check is performed; a dictionary traceability check that computes edit distance between attribute names is performed; a maximum matching algorithm and maximum likelihood classifier is applied; and results are displayed visually [4]. The project evaluated was an industrial telecommunications system and consisted of 29 C++ components, about 308 KLOC, for which object oriented object models and code was available [4]. Settimi et al [46] discuss the use of information retrieval techniques to dynamically generate traces. Though they primarily focus on tracing requirements to UML artifacts, they also compare different information retrieval techniques for tracing requirements to code and test cases. The authors have analyzed the effectiveness of the Vector space model and pivot normalization based score using both thesaurus and no thesaurus. They show that requirements tracing to UML diagrams produces impressive results.

While [3] and [31] studied requirements-to-code traceability, in [20] and [12] we have addressed the problem of tracing requirements between different documents in the project document hierarchy. In the preliminary study [20], we have implemented three methods: TF-IDF, TF-IDF with key phrases and TF-IDF with simple thesaurus. We reported on the success of these methods in identifying links between two requirements documents. In our study, retrieval with simple thesaurus outperformed other methods on our test dataset, producing recall of 85% with precision of 40%. [21] continues the research started in [20]. We extended the baseline TF-IDF and thesaurus retrieval methods with analyst relevance feedback processing capability [21].

In [21], we introduced requirements for a tracing tool from an analyst's perspective. Note that the requirements proposed in [21] and this paper have two components: objective, that can be evaluated by studying the software outputs, and subjective, that can only be evaluated by studying the work of human analysts with the tool and their reactions to the outputs. This study concentrates on the objective aspects of the work, a subjective study is currently in development stages. This paper extends our work in [21] as well as [24] and [23] by introducing latent semantic indexing, by evaluating the sub-goal of scalability, by examining a number of secondary measures, and by using a new, large dataset for validation.

In [22] we have developed a framework for comparing traceability studies and we have used it to compare four of the abovementioned studies, [3], [4], [31], [20], in a detailed manner. Examining our current work, we note that [3] and [31] remain the closest related research. The work described in this paper has used two of the three methods found in [3], [31]. However, we should note the key differences in our research and theirs. Both Antoniol et al. and Marcus and Maletic have applied their method to the documentation-to-code traceability problem, whereas our current work addresses requirements-to-requirements and requirements-to-design traceability. In addition, the key aspect of our study in this paper is the effects of relevance feedback processing on tracing – the question not addressed in the work prior to ours. If we factor out the feedback from our study, we see both similarities and differences in the results. Quantitatively (despite the fact that [3], [31] used somewhat different measurement techniques), the precision-recall results we are getting are similar to the numbers obtained by them, although we stress here that direct comparison of numbers is not very meaningful, because of the difference in datasets used. Qualitatively, Marcus and Maletic[31] showed that LSI outperformed tf-idf on the same datasets for documentation-to-code traceability. In our experiments, we have observed that tf-idf outperformed LSI.

VI. Conclusions and Future Work

In this paper, we examined the effectiveness of information retrieval methods in automating the tracing of textual requirements. Specifically, we found that analyst feedback improves the final trace results using objective measures. We also posited a set of goals for an effective tracing tool and then evaluated our tracing tool, RETRO, in this light. We found evidence that RETRO does satisfy the Believability sub-goals of Accuracy and Scalability as well as the Discernability and Endurability goals. There is also preliminary evidence for the objective aspect of the Usefulness portion of the Utility sub-goal of Believability.

Much work remains to be done, however. In terms of the effectiveness of methods, we can see that we are on the right track. We are able to achieve high levels of recall at reasonable levels of precision. But we are not achieving high levels of precision without the assistance of filtering. This indicates to us that we may need other methods to address precision.

One line of research we intend to pursue involves determining the important words in a textual artifact, using, for example, the Chi-square.

The remaining goals (Believability::Utility and Endurability) need to be evaluated. A subjective study will be required to evaluate these goals and is currently in the planning stages.

VII. Acknowledgements

Our work is funded by NASA under grant NNG04GP67G. We thank Stephanie Ferguson and Ken McGill. We thank Mike Chapman and the Metrics Data Program. We thank the MODIS project for maintaining their website that provides such useful data. 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. We would also like to thank Doug Oard for useful discussions about measurement in Information Retrieval. Last but not least, we would like to thank anonymous reviewers for their helpful comments.

References

[1] Abrahams, M. and Barkley, J., RTL Verification Strategies, in *Proc. IEEE WESCON/98*, 15 - 17 September 1998, pp. 130-134.

[2] Anezin, D., Process and Methods for Requirements Tracing (Software Development Life Cycle), *Dissertation*, George Mason University, 1994.

[3] 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.

[4] Antoniol, G., Caprile, B., Potrich, A., Tonella, P., Design-Code Traceability for Object Oriented Systems, *Annals of Software Engineering*, volume 9, 35 - 38, 1999.

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

[6] Bandwidth. www.bandwidthmarket.com/resources/glossary/S2.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, in *Proceedings of the IEEE/ACM International Conference on Computer-aided Design*, pp. 350–355, Santa Clara, CA, 1993.

[10] Cleland-Huang, J., Chang, C.K., Sethi, G., Javvaji, K.; Hu, H., Xia, J., Automating speculative queries through event-based requirements traceability, in *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02)*, Essex, Germany, 9-13 September, 2002, pages: 289- 296.

[11] Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., and Harshman, R., Indexing by Latent Semantic Analysis, *Journal of the American Society of Information Science*, Vol 41, No. 6, pp. 391–407, 1990.

[12] Dekhtyar, A., Hayes, J. Huffman, and Menzies, T. Text is Software Too, in *Proccedings of the International Workshop on Mining Sofwtare repositories (MSR)*,2004, pp. 22-26, May 2004, Edinborough, Scotland.

[13] Egyed, A. A Scenario-Driven Approach to Traceability, in *Proceedings of the 23rd International Conferenceon Software Engineering (ICSE)*, Toronto, Canada, May 2001, pp. 123-132.

[14] Egyed, A., Gruenbacher, P. Automatic Requirements Traceability: Beyond the Record and Replay paradigm, in *Proc. of the 17th IEEE International Conference on Automated Software Engineering*, pp. 163–171, Edinburgh, UK, September, 2002.

[15] Flesch, R.F. A New Readability Yardstick, *Journal of Applied Psychology*, 32: 221-233 (1948).

[16] Flesch, R.F. Estimating the Comprehension Difficulty of Magazine Articles, *Journal of General Psychology*, 28: 63-80 (1943).

[17] Gotel O., Finkelstein A. An Analysis of the Requirements Traceability Problem, in *Proc. of the 1st International Conference in Requirements Engineering*, 94-101, 1994.

[18] Hayes, J.Huffman. Energizing Software Engineering Education through Real-World Projects as Experimental Studies, in *Proceedings of the 15th Conference on Software Engineering Education and Training (CSEET)*, 192–261, February 2002, Cincinnati, Ohio.

[19] Hayes, J. Huffman. Risk reduction through requirements tracing. In *Proc., Software Quality Week*, 1990, San Francisco, California, May 1990.

[20] 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.

[21] 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)*, pp. 249-261, Kyoto, Japan, September 2004.

[22] Hayes, J. Huffman, Dekhtyar, A., A Framework for Comparing Requirements Tracing Experiments, *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, Vol. 15, No. 5 (October 2005), pp. 185-215, 2005.

[23] Hayes, J. Huffman, Dekhtyar, A., and Sundaram, S., Measuring the Effectiveness of Retrieval Techniques in Software Engineering, *University of Kentucky Technical Report* TR 422-04, October 2004.

[24] Hayes, J. Huffman, Dekhtyar, A., and Sundaram, S., Advancing Requirements Tracing: An Objective Look, *University of Kentucky Technical Report* TR 423-04, November 2004.

[25] Hayes, J. Huffman, Dekhtyar, A., and Sundaram, S., Text Mining for Software Engineering: How Analyst Feedback Impacts Final Results, in *Proceedings, 2nd International Workshop on Mining Software Repositories (MSR'2005)*, pp. 58-62, 2005.

[26] Hayes J. Huffman, Dekhtyar A., Humans in the Traceability Loop: Can't Live With'em, Can't Live Without'em., *Proc. 3d International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*, pp. 20–23, Long Beach, CA, 2005.

[27] HoLagent Corporation product RDD-100,

[28] Hoffman, M.,Kuhn, N., Weber, M., and Bittner, M, Requirements for Requirements Management Tools, in *International*

Conference on Requirements Engineering, Los Alamitos, California: IEEE Computer Society Press, 2004, pp. 301-308.
[29] INCOSE Requirements Management Tool Survey. http://66.34.135.97/tools/survey/rtm.html.
[30] Level 1A (L1A) and Geolocation Processing Software Requirements Specification, *SDST-059A, GSFC SBRS*, September 11, 1997.
[31] 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.
[32] MDP Website, CM-1 Project, http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1.
[33] MODIS Science Data Processing Software Requirements Specification Version 2, *SDST-089, GSFC SBRS*, November 10, 1997.
[34] 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.
[35] Offutt, J., Yang, Y., Hayes, J. Huffman, SEEWeb: Making Experimental Artifacts Available, in *Proceedings, Workshop on Empirical Research in Software Testing (WERST)*, in ACM SIGNOTES, Volume 29, Issue 5, pp. 1–3, September 2004, Boston, MA, July 2004.
[36] Pohl, K. PRO-ART: Enabling Requirements Pre-Traceability, in *Proceedings of the IEEE International Conference on Requirements Engineering (RE)*, pp. 76–85, 1996.
[37] Pierce, R., A requirements tracing tool, in *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues,* 1978.
[38] Pinheiro F., Goguen J., An Object-Oriented Tool for Tracing Requirements, *IEEE Software*, 52-64, March 1996.
[39] Predictor Models in Software Engineering (Promise) Software Engineering Repository. http://promise.site.uottawa.ca/SERepository.
[40] Ramesh, B., Factors Influencing Requirements Traceability Practice, *Communications of the ACM*, December 1998, Volume 41, No. 12, pp. 37-44.
[41] 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.
[42] Ramesh, B., Dhar, V., Supporting Systems Development Using Knowledge Captured During Requirements Engineering, *IEEE Transactions in Software Engineering*, June 1992, 498-510, 1992.
[43] Rational RequisitePro, http://www.rational.com/products/reqpro/index.jsp
[44] Rumbaugh, J., Blah, M., Premerlani, W., Eddy, F., Lorensen, W. Object-Oriented Modeling and Design. *Prentice-Hall*, 1991.
[45] Savvidis, I. A Multistrategy Framework for Analyzing System Requirements (Software Development), *Dissertation*, George Mason University, 1995.
[46] Settimi, R., Cleland-Huang, J., Ben Khadra, O., Mody, J., Lukasik, W., DePalma, C., Supporting software evolution through dynamically retrieving traces to UML artifacts, in *Proceedings of 7th International Workshop on Principles of Software Evolution*, 2004, 49- 54.
[47] Sjoberg, D.I.K., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M.; Karahasanovic, A., Koren, E.F., Vokac, M. Conducting realistic experiments in software engineering, in *Proc. International Symposium on Empirical Software Engineering*, 2002, pp. 17 - 26.
[48] Spanoudakis, G. Plausible and Adaptive Requirement Traceability Structures, in *14th International Conference on Software Engineering and Knowledge Engineering*, pp. 135–142, Ischia, Italy, July 2002.
[49] Spanoudakis, G., Zisman, A., Perez-Minana, E., Krause, P., Rule-Based Generation of Requirements Traceability Relations, *Journal of Systems and Software*, pp. 105–127, 2004.
[50] SQL Server Performance. www.sql-server-performance.com/glossary.asp.
[51] Sundaram, S.K., Hayes, J. Huffman, and Dekhtyar, A. Baselines in Requirements Tracing, in *Proceedings, Workshop on Predictor Models in Software Engineering (PROMISE)*, May 2005, St. Louis, MO.
[52] Telelogic product DOORS, http://www.telelogic.com/products/doorsers/doors /index.cfm
[53] Tsumaki, T. and Morisawa, Y. A., Framework of Requirements Tracing using UML, in *Proceedings of the Seventh Asia-Pacific Software Engineering Conference*, 2000, 5 - 8 December 2000, pp. 206 - 213.
[54] Watkins, R. and Neal, M., Why and How of Requirements Tracing, *IEEE Software*, Volume 11, Issue 4, July 1994, pp. 104-106.
[55] Yadla, S., Hayes J. Huffman, Dekhtyar A. Tracing Requirements to Defect Reports, *Innovations in Systems and Software Engineering: A NASA Journal*, Vol. 1, No 2, pp. 116–124, September 2005.

Jane Huffman Hayes. Jane Huffman Hayes is an Assistant Professor in the Department of Computer Science at the University of Kentucky. She holds a Ph.D. in Information Technology from George Mason University. Her research interests include requirements, software verification and validation, traceability, maintainability, and reliability. She is a member of the IEEE Computer Society.

Alex Dekhtyar. Alex Dekhtyar is an Assistant Professor in the Department of Computer Science at the University of Kentucky. His interests include traceability, management and reasoning with uncertain information, digital libraries, computing in humanities, and management of XML data. He has a Ph.D. in Computer Science from the University of Maryland at College Park. He is a member of ACM, ACM SIGMOD, SIGIR and SIGKDD, American Association for AI, and Association for Logic Programming.

Senthil Karthikeyan Sundaram. Senthil Karthikeyan Sundaram is a Ph.D. candidate at the Department of Computer Science, University of Kentucky. He graduated from Madras UIniversity with a Bachelor of Engineering in Computer Science and Engineering. His interests include traceability, requirements engineering, software design, information retrieval, and data mining. He is a member of the IEEE Computer Society.