Helping Analysts Trace Requirements: An Objective Look

Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, Sarah Howard
Computer Science Department
University of Kentucky
hayes@cs.uky.edu, dekhtyar@cs.uky.edu, skart2@uky.edu, sehowa2@uky.edu

Abstract

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

Research

1. 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 [23], RDD-100 [12], and Rational RequisitePro [20] 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. Similarly, the RTM can be viewed as the backbone of V&V/IV&V.

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. Generally, after the documents have been parsed and requirements have been extracted from the two document levels, an analyst will manually read each high level requirement and low-level element and assign keywords to each. 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 metrics in evaluating candidate links: the percentage of actual matches that are found (*recall*) and the percentage of correct matches as a ratio to the total number of candidate links returned (*precision*).

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 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 user to perform interactive searches for potential linking requirements or design elements, they require the user to assign keywords to all the elements in both document levels prior to tracing, they return many potential or 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.

Previously, we focused solely on the problem of generating candidate links, discussed in [11]. Specifically, we showed that information retrieval (IR) methods were effective and efficient when used to generate candidate link lists. Our focus has now broadened to the overall requirements tracing process. The penultimate 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 in requirements tracing analysis). To that end, this paper provides three contributions: *(i)* we investigate the analyst responsibilities in performing tracing; *(ii)* we derive unique high-level analyst-oriented tool requirements from these; and, *(iii)* we present a prototype tool, **RETRO** (**RE**quirements **TR**acing **O**n-target), and evaluate it with respect to the requirements.

The paper is organized as follows. Section 2 presents the requirements for an effective requirements tracing tool. Section 3 discusses our tool and how it satisfies the requirements of Section 2. Section 4 discusses the results obtained from evaluation. Related work in requirements tracing is presented in Section 5. Finally, Section 6 presents conclusions and areas for future work.

2. Requirements 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. The analyst is required to: ***(a)*** identify each requirement; ***(b)*** assign a unique identifier to each requirement; ***(c)*** for each requirement to be traced (say for example from a high level document to a low level document), locate all children requirements present in the lower level document; ***(d)*** for each low level requirement, locate a parent requirement in the high level document; ***(e)*** examine each high level traced requirement and determine if it has been completely satisfied by the low level requirements that were selected as links; ***(f)*** prepare a report that presents the traceability matrix (low level requirements traced to high level requirements); and ***(g)*** prepare a summary report that expresses the level of traceability of the document pair (that is, what percentage of the high level requirements were completely satisfied, what percentage of low level documents had no parents, etc.).

Let us next examine how automation may facilitate these responsibilities. A tool could easily assist 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 SuperTracePlus (STP) [10,16], 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 400 candidate links.

If we build a tool to automate items (c) - (e), the analyst will still have certain critical responsibilities. These include evaluating candidate links; making decisions on whether or not candidate links should be accepted or rejected; making decisions on whether or not to look for additional candidate links; making decisions on whether or not a requirement has been satisfied completely by its links; and deciding 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 [11], 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 [19] or has looked at recall and precision to assess the accuracy of the applied linking algorithms [3, 14]. To our knowledge, there has not been work published that details the requirements for an effective requirements tracing tool. In addition to specifying such requirements, we provide a validation mechanism for each requirement, and then in Sections 3 and 4 demonstrate that our tracing tool satisfies the requirements we have addressed to date. Note that we have chosen to define the requirements in an informal, textual narrative format. We do not claim that these requirements possess the quality attributes that should be present in formal software requirements. Rather, we offer them as a starting point for discussion with other researchers.

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.

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 requirement.

Requirement 1:

Specification:

"**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-requirements or sub-elements: accuracy, scalability, and utility discussed below.

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. 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 x 50 = 1000 combinatorial links. Any traceset with more than 3000 combinatorial links is considered large.

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* and *Process Enforcement*. Operability is the capability of the software product to enable the user to operate and control it [4]. Process Enforcement refers to the tool implementing tracing in such a way that the analyst is guided through the process.

Validation mechanism:

The standard measures of accuracy are recall and precision. 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, even orthogonal, 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. A candidate link list with high recall and low precision means that the analyst has to weed out the many false-positive links from it before the requirements tracing task is complete. On the other hand, if the same list has high precision and low recall, the analyst would have to examine a lot of potential links **outside** the list. In this respect, high-recall, low-precision lists of links appear to be preferable to high-precision, low recall links. That is because humans seem to be better at determining whether a specific pair of links from the list is a match than at discovering new pairs of links in the document from scratch.

For scalability, we must examine the tool's results for both small and large trace sets to determine that the accuracy has not been significantly degraded. Validation of utility requires subjective measures and hence a separate experimental design. In addition, we must first establish accuracy and scalability before progressing to a subjective study, thus ensuring that the tool performs in such a way that there is a basis for analyst confidence. This study is left for future research.

Discussion:

Believability is a high level, overarching requirement. Utility is important because in any tracing exercises other than controlled experiments, the theoretical "true trace" will not be known. Therefore, an analyst has to decide whether candidate links are correct or not and/or whether to search for additional candidate links. The analyst must feel confident that good results have been achieved by using the tool.

Scalability is not addressed in this paper, as we do not currently have large trace sets with a "true trace" (See Section 6). Accuracy is evaluated, though. Recall is more important in tracing than precision because we do not want analysts to have to search for additional candidate links. We also want precision to be as high as possible. But note that precision values can be a bit misleading. For example, 50% precision means the existence of one false positive for each true link, which would be relatively easy for the analyst to deal with. Improvement beyond 50% does not provide as much benefit to the analyst as for example improving from 10% to 33% (which corresponds to improving from one true link out of 10 to one true link out of three candidates). Thus, drastic improvements in precision occur only at low percentages. The true measure of the effectiveness of a tracing tool lies in its ability to help an analyst find the correct links, as easily as possible. In earlier studies [11], we found that an analyst using the STP requirements tracing tool actually ended up with a worse final answer than the tool had originally proposed. If the analyst throws away good links, recall will decrease. If the analyst keeps bad links, precision will decrease. It is important that the tool prompts/assists the analyst to make the right choices (addressing items (c) and (d)). To that end, we have requirement 2, "discernability."

Requirement 2:

Specification:

"**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 four aspects to this requirement. 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 *DiffAR* and *Lag,* while the existence of a cutoff is studied using different filtering techniques on the candidate link lists. These measures are formally defined in Section 4.

Discussion:

This requirement must be satisfied to support the satisfaction of Requirement 1. As has been suggested above, requirements tracing is an iterative process. An analyst will examine a subset of the candidate links and then determine if the links are good or not. This information, even for a small number of candidate links, is very valuable and should be fed back into the algorithms to support the generation of more accurate candidate links. If the tool does not provide the candidate links in a manner that facilitates discernment, the analyst will get frustrated with the tool and will not be able to efficiently complete the task. That leads us to our final requirement, "endurability."

Requirement 3:

Specification:

"**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. However, 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.

Discussion:

In general, requirements tracing is a very time consuming, arduous process, even when using a tool. We strive to decrease the tedium of the tracing experience for the analyst (addressing items (c) - (e)). This is a subjective item, tying in with usability. A separate study is planned to assess analyst attitude toward our tracing tool.

3. Effective requirements tracing with RETRO

3.1 Why use Information Retrieval?

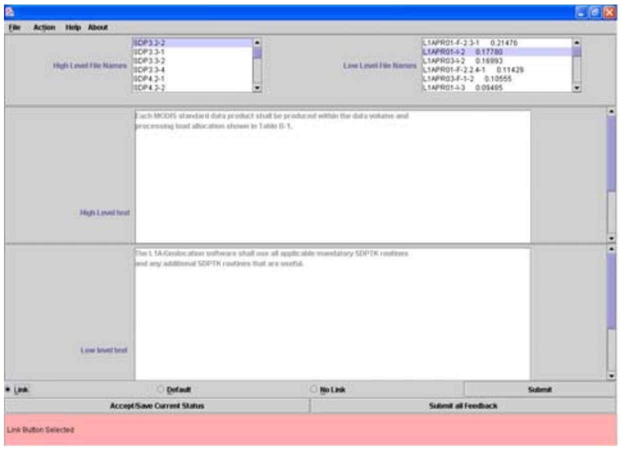
The problem of requirements tracing boils down to determining, for each pair of requirements from high- and low-level requirements documents, whether they are "similar." Stated as such, requirements tracing bears a striking similarity to the standard problem of Information Retrieval (IR): given a document collection and a query, determine which documents from the collection are relevant to the query. In the 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, for example with the STP tool [10,16] 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.

3.2 RETRO

In contrast with such comprehensive requirements management tools as STP [10, 16], 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 RETRO GUI (Win32 port) is shown in Figure 1.



SDP3.2-2
SDP3.3-1
SDP3.3-2
SDP3.3-4
SDP4.2-1
SDP4.2-2

L1APR01-F-2.3-1 0.21476
L1APR01-I-2 0.17780
L1APR03-I-2 0.16893
L1APR01-F-2.2.4-1 0.11429
L1APR03-F-1-2 0.10555
L1APR01-I-3 0.09485

Each MODIS standard data product shall be produced within the data volume and processing load allocation shown in Table B-1.

The L1A Geolocation software shall use all applicable mandatory SDPTK routines and any additional SDPTK routines that are useful.

Figure 1. A screenshot of RETRO.

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 going through 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 it, with relevance judgments (right). In the middle part of the interface, the text of the current pair of requirements is displayed. 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. 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 discovery process consistent with the changes. If needed, the IR method is re-run and the requirements tracing process proceeds into the next iteration.

3.3 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 previously [11]: *tf-idf vector retrieval* and *vector retrieval with a simple thesaurus*. To process feedback we have used the Standard Rochio [9] method for the vector model. The methods used are briefly described below.

3.3.1 Tf-Idf model. Standard vector model (also known as tf-idf model) for information retrieval is defined as follows. Let $V = \{k1,\ldots,kN\}$ be the vocabulary of a given document collection. Then, a vector model of a document *d* is a vector $(w1,\ldots,wN)$ of keywords weights, where *wi* is computed as $w_i = tf_i(d) \cdot idf_i$. Here *tfi(d)* is the so-called *term frequency:* the frequency of keyword *ki* in the document *d,* and *idfi,* 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 document collection and *dfi* is the number of documents in which keyword *ki* occurs. Given a document vector $d=(w1,\ldots,wN)$ and a similarly computed query vector $q=(q1,\ldots,qN)$ the similarity between *d* and *q* is defined as the cosine of the angle between the vectors:

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

3.3.2 Tf-Idf + Simple Thesaurus. The second method used in [11] extends tf-idf model with a simple thesaurus of terms and key phrases. A simple thesaurus T is a set of triples $<t, t', \alpha>$, where t and t' are *matching thesaurus terms* and α is the similarity coefficient between them. Thesaurus terms can be either single keywords or key phrases – sequences of two or more keywords. 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 $T=\{<ki,kj,\alpha ij>\}$, and document/query vectors $d=(w1,...,wN)$, $q=(q1,...,qN)$, the similarity between d and q is computed as:

$$sim(d,q) = \cos(d,q) = \frac{\sum_{i=1}^{N} w_i \cdot q_i + \sum_{<ki,kj,\alpha_{ij}> \in T} \alpha_{ij}(w_i \cdot q_j + w_j \cdot q_i)}{\sqrt{\sum_{i=1}^{N} w_i^2 \cdot \sum_{i=1}^{N} q_i^2}}.$$

3.4 Incorporating relevance feedback

Relevance feedback 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 user. More formally, 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 provided by the user. Note that D_r and D_{irr} are disjoint, but do not necessarily cover the entire set D_q. We use Standard Rochio [9] feedback processing method:

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

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 (in our tests all three values were set to 1). 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 user is satisfied with the results.

4. Evaluation

4.1 Study design

The purpose of our current study is to see whether RETRO satisfies the requirements specified in Section 2. We notice that all three major requirements have two components: objective, that can be measured by running tests on the tool, and subjective, examining user interaction with it. This study validates parts of the requirements that can be measured objectively. A study of the use of the tool by analysts for the purpose of determining its usability is planned next. In particular, in this study, we concentrate on determining the accuracy and discernability of the results of the analysis.

To assess the accuracy and discernability of requirements tracing with RETRO, we performed tests on tf-idf and thesaurus approaches, as described in Section 3.3. We used a modified dataset from [11] based on open source NASA Moderate Resolution Imaging Spectroradiometer (MODIS) documents [13,15]. The dataset contains 19 high level and 49 low-level requirements. The trace for the dataset was manually verified. The "theoretical true trace" built for this dataset consisted of 42 correct links.

In the test, we have assumed that during the feedback process, analysts provide correct information to the tool. That is, both true links and false positives, when discovered are marked as such. At the beginning of each test, the traceset was loaded into RETRO and a particular IR method (tf-idf, or thesaurus) was selected. For each method, four different feedback strategies or behaviors, called *Top 1, Top 2, Top 3* and *Top 4* were tested. The *Top i* behavior meant that at each iteration, we simulated correct analyst feedback for the top *i* unmarked candidate links from the list for each high-level requirement. For example, for each high level requirement, *Top 1* behavior examined the top candidate link suggested by the IR procedure that had not yet been marked as true. If the link was found in the verified trace, it was marked as true, otherwise – as false. After repeating the *Top i* relevance feedback procedure for each high level requirement, the answers were submitted to the feedback processing module. At this point, the Standard Rochio procedure was used to update query (high-level requirement) keyword weights, and to submit the new queries to the IR method. The process continued for a maximum of eight iterations or until the results had converged.

To check the accuracy of the results, we measured precision and recall of the candidate link lists produced at each iteration of the process.

To check discernability, we devised and computed a number of measures that allow us better insight into the evolution of the candidate link lists provided by RETRO from iteration to iteration. As stated in Section 2, we want to ensure that (a) true links rise to the top of the lists, (b) false positives sink to the bottom of the lists, and that (c) a reasonable cut-off is possible that separates the majority of the true links from the majority of the false positives. The metrics measuring the degrees to which (a) and (b) were satisfied are:

ART: Average relevance of a true link in the list;

ARF: Average relevance of a false positive in the list;

DiffAR = *ART* – *ARF*: the difference between average relevances of true links and false positives; and

Lag: average number of false positives with higher relevance coefficient than a true link.

To measure the ability to establish a cut-off, we have examined a number of *filtering techniques.* A filtering technique is a simple decision procedure that examines each candidate link produced by the IR method and decides whether to show it to the analyst. In our study, in addition to the test run involving no filtering, we used the following three filtering techniques:

Above 0.1: throw out a candidate link if its relevance is below 0.1.

Within 0.5: For each high-level requirement, compare the relevance of each candidate link with the relevance of the top candidate link. The candidate link is retained *iff* its relevance is within 0.5 of the relevance of the top link.

Top 42. For each high-level requirement *i,* retain the top *ki* candidate links, where *ki* is the number of true links for *i*. The filtered answer set contained 42 (or fewer) candidate links distributed exactly as in the answer set.

4.2 Results

We address accuracy followed by discernability. As discussed above, recall and precision will be used to assess accuracy. The precision and recall results obtained in our tests are summarized in Table 1. The first column indicates the iteration, with iteration 0 being the iteration before the feedback. In each cell, precision is indicated first followed by recall. For example, iteration 7, Top 3, for Thesaurus method had precision of 24.6% and recall of 80.9%. The maximal precision and recall achieved in each experiment are highlighted and the maximal values for each retrieval method are also underlined. The results are also visualized in Figure 2, which contains the precision/recall trajectories for all experiments, grouped by the IR method used (top: tf-idf, bottom: simple thesaurus).

The importance of the results ties back to the requirement of believability in Section 2. The candidate link lists generated using the thesaurus method are decent, but are greatly improved with analyst feedback. Also, improvements in recall are seen in early iterations (as early as iteration 3) and with the analyst only providing feedback on the Top 2 links. We feel that these accuracy results should also contribute to utility and endurability. Note that our shortcoming in recall is accounted for by a few requirements for which the IR methods did not return any true candidate links at iteration 0. This meant that feedback methods could not improve as they could not acquire positive feedback information.

Precision does not appear to improve as much. It doubles over six or seven iterations, but on iterations 1 and 2 it decreases before starting to increase again with iteration 3. However, precision was improved without much impact to recall by using filtering. Table 2 summarizes the results of applying different filters to the candidate link lists. For each filter and for each test run, the best precision/recall pair was always achieved on the last iteration of the experiment – a contrast with the results without filtering. For each filter, the table also contains the differences in percentages for recall and precision between the list with no filtering and the list obtained by applying the filter. As evidenced in the table, the improvement in precision is significant for most filter---IR algorithm---behavior combinations. An important observation is that removal of a candidate link from the list by a filtering technique at some iteration does not preclude this link from appearing again in a subsequent iteration. What this means is that if we filtered out some good links originally, they may reappear later with higher similarity measures. Filtering also ties to discernability.

Recall that we use filtering to determine if there is eventual separation between good and bad links in the candidate link list, or the cutoff sub-element of discernability. Our results show that for above 0.1 in Top 42 filters such separation is eventually achieved for most of the behaviors, as precision increases drastically while the decrease in recall is not large. For example, using thesaurus retrieval for Top 3 behavior and above 0.1 filtering, precision is almost 40% with recall of 80%.

The other sub-elements of discernability examine whether good links rise to the top of the list and bad links sink to the bottom. Recall that we use *DiffAR* and *Lag* to assess these. Figure 3 shows the changes in the *DiffAR* metric: the difference in average relevances between the true links (*ART*) and false positives (*ARF*) as the iterations

.

Table 1. Results of experiments: Precision and recall.

I	Top 1		Top 2	
	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus
0	11.3%, 57.1%	12.2%, 64.2%	11.3%, 57.1%	12.2%, 64.2%
1	8.4%, 59.5%	9.4%, 69%	6.9%, 59.5%	7.1%, 66.6%
2	7.7%, 59.5%	8.3%, 64.2%	9.2%, 69%	9.9%, 73.8%
3	9.2%, 66.6%	8.6%, 61.9%	12.2%, **73.8%**	12.1%, 80.9%
4	9.9%, 71.4%	10.6%, 71.4%	15.1%, **73.8%**	15.7%, **83.3%**
5	12.5%, **76.1%**	12.5%, **78.5%**	17.9%, 71.4%	15.8%, 80.9%
6	**15.3%**, **76.1%**	14.8%, 76.1%	20%, 69%	17.6%, **83.3%**
7		16%, 71.4%	23.3%, 69%	21.9%, 80.9%
8		**17.3%**, 73.8%	**25.6%**, 69%	**25%**, 78.5%

I	Top 3		Top 4	
	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus
0	11.3%, 57.1%	12.2%, 64.2%	11.3%, 57.1%	12.2%, 64.2%
1	7.3%, 61.9%	7.8%, 66.6%	8.3%, 69%	8.6%, 76.1%
2	11.6%, 73.8%	10.6%, **83.3%**	12.1%, **76.1%**	12.4%, **80.9%**
3	14.6%, 73.8%	13.6%, 80.9%	15.1%, 73.8%	15.3%, **80.9%**
4	18.6%, **76.2%**	17.4%, **83.3%**	13%, 61.9%	16.7%, 78.5%
5	17.1%, 71.4%	18.1%, **83.3%**	19.4%, 73.8%	**18.6%**, 78.5%
6	20.9%, 73.8%	20.2%, 80.9%	**22.7%**, 71.4%	
7	21.4%, 69%	**24.6%**, 80.9%	**22.7%**, 71.4%	
8	**22.7%**, 66.6%			

progress. Intuitively, the larger the difference, the more likely it is that most of the true links will be at the top of the candidate link lists for high-level requirements. Note from the figure that the difference between *ART* and *ARF* is around 0.2 at iteration 0 for both tf-idf and simple thesaurus retrieval algorithms. At subsequent iterations for both retrieval algorithms and all behaviors, *DiffAR* grows significantly, ranging from 0.489 to 0.758 at the last iterations.

While *DiffAR* measures the quantitative separation between true links and false positives, *Lag* is a measure of qualitative separation. *Lag* is defined for each true link in the list as the number of false positives for its high-level requirement that have a higher relevance (i.e., the number of false positives that are higher up in the list). The *Lag* of a list of candidate links is the average *Lag* of its true links. Note that when *Lag=0*, total separation of links has been achieved: all true links appear higher up in the lists of candidate links than all false positives.

Figures 4 and 5 show the progress of the *Lag* measure for tf-idf and thesaurus retrieval tests respectively. It can be observed that in all experiments *Lag* behaved in a similar manner. For both tf-idf and thesaurus retrieval, *Lag* starts at just above 6. During the first 1-2 iterations, *Lag* grows, and for some experiments can go as high as 10. But at subsequent iterations, *Lag* drops significantly, and in all but one experiment, finishes under 3.

Figure 2. Recall, precision for all behaviors for Thesaurus retrieval.

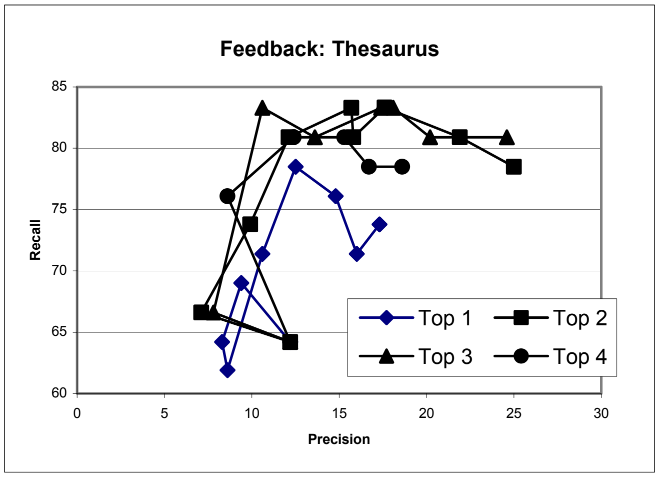
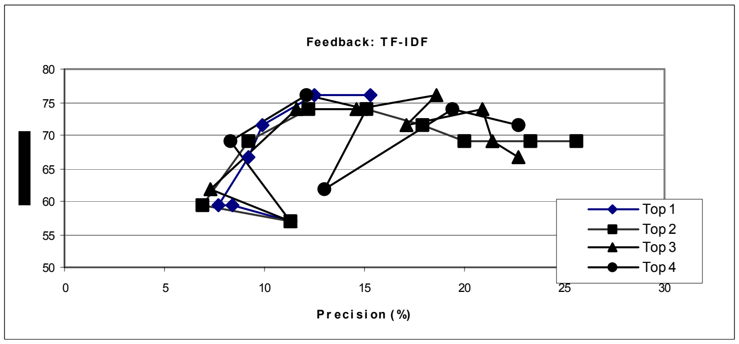


Figure 3. Recall, precision for all behaviors for TF-IDF retrieval.



4.3 Discussion of results

Table 3 summarizes the contributions of the paper. It is evident that RETRO supports the objective sub-elements of discernability. The measures *ART*, *ARF*, and *DiffAR* indicate that using the relevance feedback option of RETRO provides the analyst with similarity measures that clearly discern between good links and bad links. In addition, the *Lag* measure shows that by the later iterations, there are very few bad links at the top of the candidate link lists.

Table 2. Filtering summary.

		No Filter	Above 0.1	Within 0.5	Top 42
Top 1 best	p	17.3	44.4	63.6	64.1
	r	78.5	61.9	33.3	59.5
Top 1 diff	p		27.1	46.3	46.8
	r		-16.6	-45.2	-19
Top2 best	p	25	37.9	54.7	71.7
	r	83.3	71.4	54.7	66.6
Top2 diff	p		12.9	29.7	46.7
	r		-11.9	-28.6	-16.7
Top3 best	p	24.6	39.5	53	73.8
	r	83.3	80.9	61.9	73.8
Top3 diff	p		14.9	28.4	49.2
	r		-2.4	-21.4	-9.5
Top4 best	p	18.6	34.8	40	61.9
	r	80.9	71.4	57.1	61.9
Top4 diff	p		16.2	21.4	43.3
	r		-9.5	-23.8	-19

Table 3. Paper Summary.

Reqt.	**Analyst Rsponsb.**	**Valid. Msr.**	**Obj./ Subj.**	**Study results**
Believability	Items (c), (d)			
Accuracy		recall, precision	Obj.	Recall of 80.9%, precision of 39.2% exceeds other tools
Scalability		recall, precision	Obj.	TBD
Utility		TBD	Subj.	TBD
Discernability	Items (c), (d)			
Communic-ability		TBD	Subj.	TBD
Good link rising		ART, ARF, DiffAR	Obj.	DiffAR grows from 0.2 to .489-.758 at last iterations
Bad link sinking		ART, ARF, DiffAR	Obj.	DiffAR grows from 0.2 to .489-.758 at last iterations
Cutoff		Lag	Obj.	Lag drops on later iterations, ending at 3 or less in all but one test
Endurability	Items (c) - (e)	TBD	Subj.	TBD

A special comment needs to be made concerning the *precision* values obtained during the experiment. As shown in Tables 2 and 3, precision in our experiments hovered between 20 and 25% without filters and went up to 30—40% without significant loss in recall when filtering was used. To put these numbers into perspective, we note, that when precision is 20% (assuming perfect recall), for each correct link in the list of candidate links, there are four false positives, i.e., in our case, the number of false positives in the list would be 42*4 = 168 and the total size of the candidate link list is 42*5 = 210. This is about 22% of the total number of potential links (19*49 = 931). Consider, for example the *Top 3* run using *Above 0.1* filter (see Table 3). Here, recall of 80.9% means that 34 out of 42 links from the theoretical true trace have been identified correctly. Precision of 39.5% means that the total number of the candidate links in the list was 86, i.e., about 9% of the total number (931) of potential links. We also note that when comparing precision numbers, their ratio is more important than their absolute difference. It is much better to be able to raise precision from 10% to 20% (from 420 to 210 candidate links with perfect recall) than from 70% to 80% (from 60 to 52 candidate links with perfect recall), as the decrease in the size of the candidate link lists is proportional to the ratio, not the absolute different. Because of these considerations, precision levels obtained during the experiment are quite acceptable in practice.

The results of this study combined with the results of an earlier study [11] indicate that RETRO is a step forward with respect to other existing tools in terms of the accuracy sub-element of believability. In this study, RETRO with relevance feedback and thesaurus and filtering achieved recall of 80.9% and precision of almost 40%. In a comparable but different study (different part of the MODIS dataset), STP achieved overall recall and precision of 63.4% and 38.8% and RETRO, without feedback or filtering, achieved overall recall and precision of 85.4% and 40.7% on the same dataset [11].

The current study clearly points to avenues for improvement. For example, modifying our methods to ensure that we always return at least one true link per requirement at iteration 0 will greatly enhance our recall in the process of feedback. We also noted that the poor results on just a few requirements greatly influenced the precision measures. By studying these "problem" requirements, we hope to gain insight that will allow us to improve the methods of RETRO.

5. Related work

In the context of our work, there are two areas of interest: requirements tracing and IR as it has been applied to the problem of requirements analysis. Each will be addressed below.

Extensive work in the area of requirements tracing has been performed by numerous researchers, including but not limited to: Pierce [17], Hayes et al [10], Mundie and Hallsworth [16], Abrahams and Barkley [1], Ramesh [18,19], and Watkins and Neal [25] Casotto [7], Tsumaki and Morisawa [24], Savvidis [21], Bohner [5], Anezin and Brouse [2,6], and Cleland-Huang [8]. A survey of work in the field of requirements tracing can be found in [11]. In addition, Spanoudakis [22] proposes a rule-based method for generation of traceability relations. His approach automatically detects traceability relations between artifacts and object models using heuristic traceability rules [22].

Figure 3. Separation between average relevance of links and false positives.

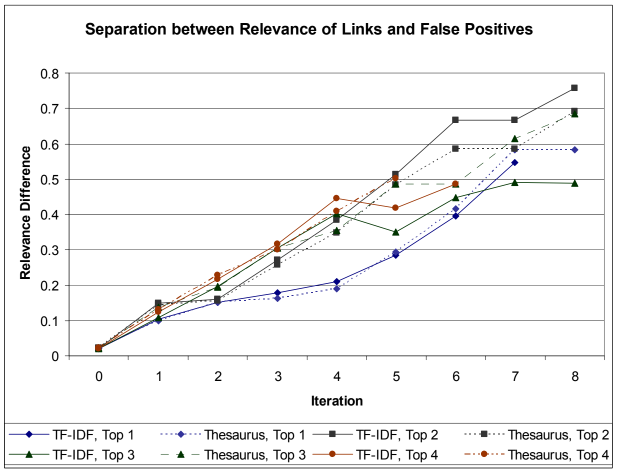
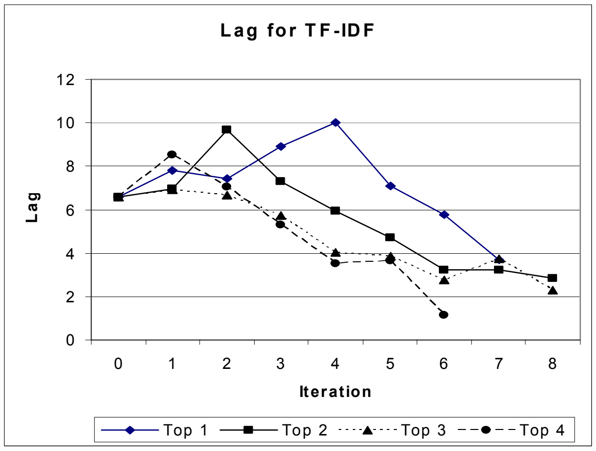
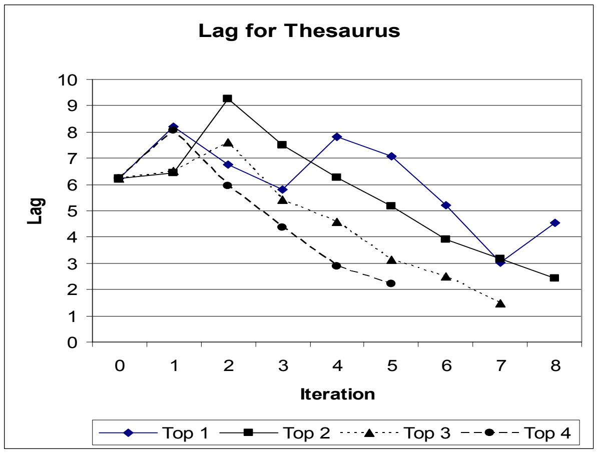


Figure 4. Lag for TF-IDF tests.



Recently, a number of researchers investigated the use of IR methods for requirements analysis. Antoniol, Canfora, De Lucia and Merlo [3] considered two IR methods, probabilistic IR and vector retrieval (tf-idf) in studying the traceability of requirements to code for two datasets. Following them, Marcus and Maletic [14] applied latent semantic indexing (LSI) technique to the same problem. While those papers studied requirements-to-code traceability, in [11] we have addressed the problem of tracing requirements between different documents in the project document hierarchy.

Figure 5. Lag for thesaurus retrieval tests.



6. Conclusions and future work

RETRO was designed for the specific purpose of supporting the IV&V analyst in performing requirements tracing. The analyst's responsibilities for finding and evaluating candidate links have been facilitated by RETRO. In addition, the objective sub-elements of the requirements of believability and discernability have been evaluated. RETRO supports accuracy and the three sub-elements of discernability of ensuring that good links rise to the top of candidate link lists, that bad links sink, and that a cutoff between good and bad links is apparent. Also, Science Applications International Corporation (SAIC), the developer of STP, is in the process of integrating the backend of RETRO (IR toolkit and feedback processing module) with the front end of STP. This is further evidence of RETRO's ability to support IV&V analysts.

Future work can be separated into two directions: improvement of the underlying technologies (IR methods, etc.); and study of the analyst's interaction with RETRO (subjective sub-elements of the requirements). Technical enhancements include use of IR methods better suited for work with small datasets, implementation of additional feedback processing methods, implementation of more intricate techniques for filtering and analysis of candidate link lists, and using IR techniques to predict the coverage or satisfaction of traced requirements by their matches. A study to determine scalability of RETRO will be undertaken. Finally, we will conduct a study of the work of analysts with RETRO. This will be a subjective study to assess the utility sub-element of believability, the communicability sub-element of discernability, and endurability. This study will also yield suggestions for the improvement of the RETRO GUI.

7. Acknowledgments

Our work is funded by NASA under grant NAG5-11732. Our thanks to Ken McGill, Tim Menzies, Stephanie Ferguson, Pete Cerna, Mike Norris, Bill Gerstenmaier, Bill Panter, the International Space Station project, Mike Chapman and the Metrics Data Program, and the MODIS project for maintaining their website that provides such useful data. We thank Hua Shao and James Osborne for assistance with the tf-idf algorithm. We thank Inies Chemmannoor, Ganapathy Chidambaram, Ramkumar Singh S, and Rijo Jose Thozhal for their assistance.

8. References

[1] Abrahams, M. and Barkley, J., "RTL Verification Strategies," 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] Bohner, S., "A Graph Traceability Approach for Software Change Impact Analysis," Dissertation, George Mason University, 1995.

[5] Avouris, N.M. "An Introduction to Software Usability," Workshop on Software Usability, University of Patras, 2001.

[6] Brouse, P., "A Process for Use of Multimedia Information in Requirements Identification and Traceability," Dissertation, George Mason University, 1992.

[7] Casotto, A.. Run-time requirement tracing, Proceedings of the IEEE/ACM International Conference on Computer-aided Design, Santa Clara, CA, 1993.

[8] 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.

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

[10] Hayes, J. Huffman. Risk reduction through requirements tracing. In The Conference Proceedings of Software Quality Week 1990, San Francisco, California, May 1990.

[11] 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.

[12] Holagent Corporation product RDD-100, http://www.holagent.com/new/products/modules.html

[13] Level 1A (L1A) and Geolocation Processing Software Requirements Specification, SDST-059A, GSFC SBRS, September 11, 1997.

[14] Marcus, A.; Maletic, J. "Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing," Proceedings of the Twenty-Fifth International Conference on Software Engineering 2003, Portland, Oregon, 3 – 10 May 2003, pp. 125 – 135.

[15] MODIS Science Data Processing Software Requirements Specification Version 2, SDST-089, GSFC SBRS, November 10, 1997.

[16] Mundie, T. and Hallsworth, F. Requirements analysis using SuperTrace PC. In Proceedings of theAmerican Society of Mechanical Engineers (ASME) for the Computers in Engineering Symposium at the Energy & Environmental Expo 1995, Houston, Texas.

[17] Pierce, R. A requirements tracing tool, Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues, 1978.

[18] Ramesh, B., "Factors Influencing Requirements Traceability Practice," Communications of the ACM, December 1998, Volume 41, No. 12, pp. 37-44.

[19] 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.

[20] Rational RequisitePro, http://www.rational.com/products/reqpro/index.jsp

[21] Savvidis, I. "A Multistrategy Framework for Analyzing System Requirements (Software Development)," Dissertation, George Mason University, 1995.

[22] Spanoudakis, G. "Plausible and adaptive requirement traceability structures," Proceedings of the 14th international conference on Software engineering and knowledge engineering (SEKE), 2002, Ischia, Italy , pp. 135 – 142.

[23] Telelogic product DOORS, http://www.telelogic.com/products/doorsers/doors/index.cfm

[24] 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.

[25] Watkins, R, Neal, M. "Why and How of Requirements Tracing," *IEEE Software*, Vol. 11, No.4, 1994, pp.104-106.