Final Report for Robust Requirements Tracing via Internet Search Technology: Improving an IV&V Technique – Phase II

Grant Number: NAG5-12868
CSIP Number: CSIP02-24
Document Number: UK-NASA-04001

13 February 2004

Prepared for:

National Aeronautics and Space Administration
Software IV&V Facility
100 University Drive
Fairmont, West Virginia 26554

Prepared by:

Dr. Jane Hayes, Dr. Alex Dekhtyar, U. KY, Science Applications International Corporation

Table of Contents

Final Report for Robust Requirements Tracing via Internet Search Technology: Improving an IV&V Technique – Phase II

1. Introduction

The problem statement and project objectives are presented below.

1.1 Problem Statement

The fundamental purpose of Verification and Validation (V&V) and Independent Verification and Validation (IV&V) is to insure 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 [Telelogic], RDD-100 [Holagent], and Rational RequisitePro [Rational] 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*). The analyst reviews the candidate links and determines which are actual links (candidate link evaluation). Finally, 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.

1.2 Project Objectives

There are three major objectives to this phase of the work.

(1) Improvement of IR methods for IV&V requirements tracing. Information Retrieval methods are typically developed for very large (order of millions - tens of millions and more documents) document collections [Baeza-Yates] and therefore, most successfully used methods somewhat sacrifice precision and recall in order to achieve efficiency. At the same time typical IR systems treat all user queries as independent of each other and assume that relevance of documents to queries is subjective for each user. The IV&V requirements tracing problem has a much smaller data set to operate on, even for large software development projects; the set of queries is predetermined by the high-level specification document and individual requirements considered as query input to IR methods are not necessarily independent from each other. Namely, knowledge about the links for one requirement may be helpful in determining the links of another requirement. Finally, while the final decision on the exact form of the traceability matrix still belongs to the IV&V analyst, his/her decisions are much less arbitrary than those of an Internet search engine user. All this suggests that the information available to us in the framework of the IV&V tracing problem can be successfully leveraged to enhance standard IR techniques, which in turn would lead to increased recall and precision. We developed several new methods during Phase II.

(2) IV&V requirements tracing IR toolkit. Based on the methods developed in Phase I and their improvements developed in Phase II, we built a toolkit of IR methods for IV&V requirements tracing. The toolkit has been integrated, at the data level, with SAIC's SuperTracePlus (STP) tool.

(3) Toolkit testing. We tested the methods included in the IV&V requirements tracing IR toolkit on a number of projects.

2. Accomplishments

The two major objectives, improvement of IR methods and implementation of IR methods for IV&V requirements tracing, are discussed below.

2.1 Improvement of IR Methods

During the course of the project, we have studied the applicability of basic Information Retrieval (IR) methods for tracing requirements. While traditional "bread-and-butter" IR approaches appear to produce good results in traditional IR domains, requirements tracing presents a number of challenges to such methods. In particular, such traditional IR methods are designed to work on very large (hundreds of thousands to billions of documents) document collections. They are also designed to work on relatively large documents. In the requirements tracing setting, the size of the collection is reasonably small: on the order of hundreds or thousands of individual requirements, while individual requirements are usually relatively small – one to three sentences long.

With this in mind, we first considered the behavior of three traditional IR methods on requirements tracing problems. The methods implemented were:

(A) TF-IDF vector retrieval. Vector model represents each document (requirement) and each query (higher-level requirement) as vector of keyword weights and computes similarity between documents and queries as a cosine of the angle between the vectors.

(A) TF-IDF vector retrieval with simple thesaurus. An extension of the standard TF-IDF retrieval with a simple thesaurus of synonyms and antonyms.

(A) Probabilistic Retrieval. Also known as Binary Independence Retrieval or Naïve Bayes Retrieval, this method attempts to estimate the probability that a given document is relevant to the query.

In addition, we are currently in the process of adoption and development of two more IR methods for Requirements Tracing:

(a) Latent Semantic Indexing (LSI). LSI uses Single-Value Decomposition (SVD) for the matrix of keyword weights constructed by the TF-IDF method. It allows

us to reduce the dimensionality of the problem and "condense" information, potentially capturing some "hidden concepts" in the document collection.

(b) Retrieval for Single Documents. A number of Text Mining methods have been recently proposed to construct descriptions of individual documents based on co-occurrence of terms in document sentences. We are, at present, working on adopting these techniques for the purpose of Information Retrieval.

Together with the three IR methods already developed, we have implemented *user feedback processing* techniques. Feedback processing allows our requirements tracing software to establish a dialog with the analyst doing requirements tracing. Our IR methods provide the analyst with the list of candidate links. The analyst examines some of them and determines whether or not matches are found. This information is fed back into the IR method, which, in turn, produces a new list of candidate links that tries to take advantage of the information communicated by the analyst.

More detailed information about the methods used in this work can be found in the papers attached in Appendix A.

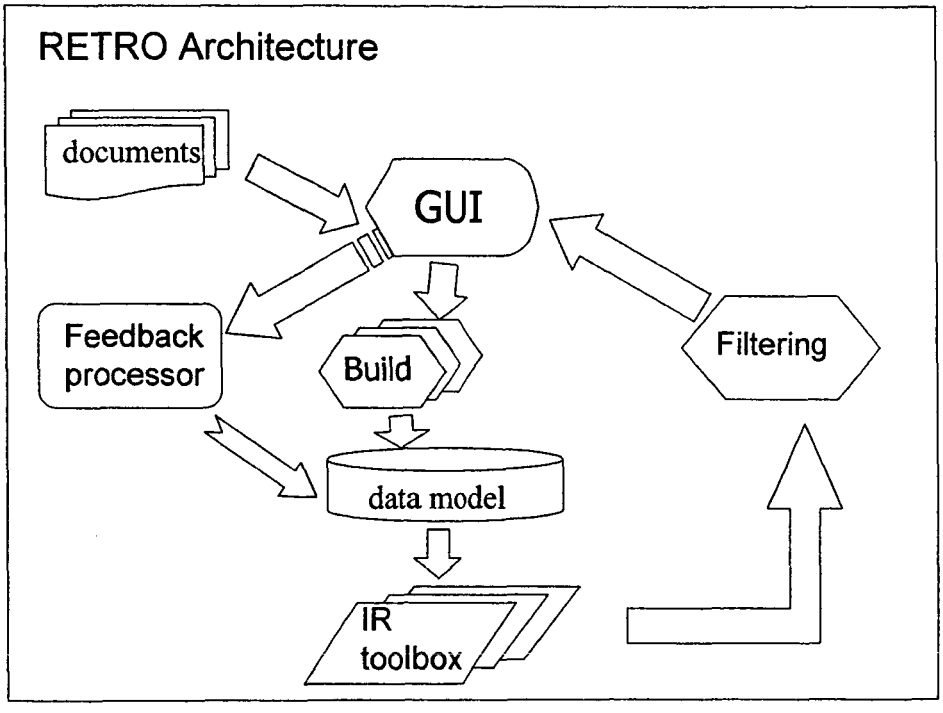


Figure 1. Architecture of RETRO.

2.2. Implementation of IR Methods for IV&V Requirements Tracing

This section describes the details of implementation of the IR methods for the purposes of Requirements Tracing. Our work yielded two concurrent developments: the standalone

requirements tracing tool RETRO (REquirements TRacing On-target), and a package integrating our IR methods into SAIC's SuperTracePlus (STP) tool.

2.2.1 RETRO - Standalone tool

First, the existing set of standalone tools were documented and debugged. Next, these tools were integrated with a JAVA GUI (recall that the standalone tools (IR methods) have been developed in C++). Some parts of the GUI were modified based on user feedback. The other tools were maintained as well. The Feedback loop tool was then developed and integrated with RETRO. A new method for probabilistic IR had also been integrated.

The overall architecture of RETRO is shown in Figure 1. The GUI is used to set up the project. Once the high- and low-level requirements are specified, and the IR method is chosen, the Build component, constructs the representation of the requirements and stores it on disk. Next, the specified method from the IR Toolbox is used to produce the first list of candidate links. This list, encoded in XML (See Appendix B for the DTD and examples), is then, optionally, processed by the Filtering component, designed to make the list somewhat smaller, without sacrificing accuracy. Using the GUI (Figure 2), the analyst checks a number of links and delivers his/her decisions back to RETRO. Encoded in XML, this information is provided to the Feedback Processor, which prepares the data for the next application of the IR method.

Next, we ported the Linux version of RETRO to Windows. The Windows version of RETRO uses FLEX for Windows and also XERCES Java and C++ parsers for XML.

2.2.2. Integration with SAIC

The main goal in integrating with SAIC's STP tool was to allow our process of candidate link generation substitute, at user's option, the traditional process employed in STP, which requires manual keyword assignment. The key aspects of integration involve porting the IR toolkit of RETRO to Windows, and providing the facilities for passing information back and forth between the toolkit and the main body of STP.

We wrote several procedures in JAVA to convert the SFEP input data from SAIC's format into the format that RETRO accepts. We also wrote procedures to convert the XML results produced by RETRO into an ASCII delimited format that SAIC requires.

We also wrote a procedure in VB which SAIC will use to call our Feedback methods. The input parameter to this procedure is a set of "yeslink " and "nolink" information about each parent-child pair in the candidate links. The feedback method will take this information and update the candidate links. The result will be sent back to SAIC's STP tool in ASCII delimited format.

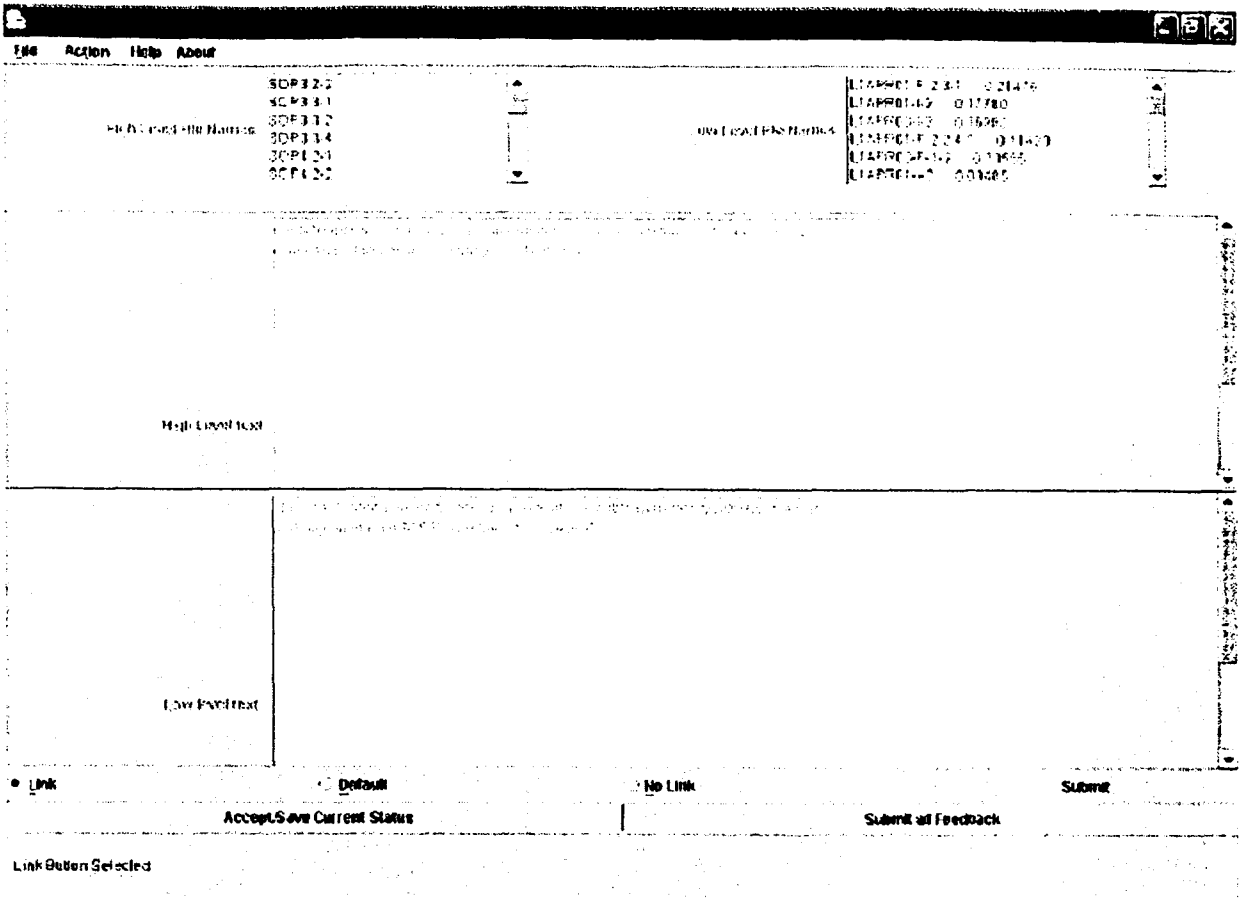


Figure 2. RETRO GUI.

2.3 Toolkit Testing

We have conducted a series of experiments designed to determine the applicability of our approach to requirements tracing. Our first objective was to measure the quality (accuracy) of the candidate link lists generated by the implemented IR methods and the improvement. The standard IR measures of precision and recall have been used to evaluate accuracy.

Our second objective was to study the question of method applicability in a more general sense. As we explain in [RE04], obtaining a high quality list of candidate links is not enough: requirements tracing is the process, by its nature, driven by a human analyst. Therefore, the real decisions about the success of our approach can only be made by studying how well the human analyst can perform requirements tracing using our tools.

The latter question has two components: objective and subjective. The objective component concerns the ability of our software to improve the candidate link lists taking advantage of the analyst activity in trace verification. In particular, this involves treating analyst's choices as feedback and processing it in order to find a better list of candidate links. Here, our interest lies not only in the pure accuracy of candidate link lists, but also in the quality: we want to ensure as the analyst proceeds to evaluate individual candidate

links, true links will tend to "rise" to the top of candidate link lists produced by the software, while false positives will tend to drop. A number of special metrics were designed to address the issue of measuring the change in the quality of the candidate link lists in addition to the accuracy.

We also note that the subjective component of testing must take into account the overall usability of the software and other related human factors. For example, determination of the circumstances under which the analyst work tends to lead to best improvement.

In [RE03] we have reported some preliminary results on the accuracy of the main methods tested so far: TF-IDF and TF-IDF with simple thesaurus. In [RE04] we have reported the results of our further experiments related to testing the objective component. These involved simulating a variety of "perfect" analyst behaviors for user feedback ("perfect" means providing the Feedback Processor only with correct information) and recording the changes in the accuracy and quality of the candidate link lists. The results obtained in the testing are discussed in detail in [RE04] (see Appendix A) and were very encouraging.

2.4 Data Used

The main dataset that we have been working with consists of open source documents for the NASA Moderate Resolution Imaging Spectroradiometer (MODIS) [Level 1A, MODIS]. The dataset contains 19 high level and 49 low-level requirements. The trace for the dataset was manually verified and 42 correct links were found.

We also have three other datasets, in varying degrees of completion. The International Space Station provided us with several levels of documents. We do not have the answer set for these though. Similarly, the Metrics Data Program provided us with a nice data set called CM-1. We have traced this dataset, but are still in the process of verifying that the trace is correct so that we can use it as a test set. Finally, we have a very small dataset from an open source imaging radio telescope called the Low Frequency Array (LOFAR). This dataset has been useful to us because the wording in both document levels is nearly identical and there is much use of many generic words. It is a particularly challenging dataset that we are using to improve our methods.

2.5. Publications

During Phase II of the work, we wrote two papers, listed below and attached as Appendix 1 and Appendix 2. The first paper was submitted to the International Conference on Software Engineering (ICSE). It was not accepted for publication. [Note that the typical acceptance rate is below 15%] The second paper was submitted to the International Conference on Requirements Engineering and is currently under review. Authors will be notified in April.

Jane Hayes, Alexander Dekhtyar, Senthil Sundaram, Sarah Howard, "On Effectiveness of User Feedback-based Information Retrieval Methods for Requirements Tracing," submitted to the International Conference on Software Engineering in September 2003.

Jane Hayes, Alexander Dekhtyar, Senthil Sundaram, Sarah Howard, "Helping Analysts Trace Requirements: An Objective Look," submitted to the International Conference on Requirements Engineering in January 2004.

3. Conclusions/Future Work

Below, we present our conclusions for this work as well as the research areas that are still before us.

3.1 Conclusions

The conclusions we have reached after Phase 2 of the project are two-fold.

(A) Traditional "bread-and-butter" IR methods allow us to drastically decrease the time it takes for the analysts to obtain the lists of candidate links. Recall values are quite high, that is, the vast majority of the true links are captured one way or another. Precision, on the other hand, is fairly low. At the same time, the overall accuracy of our best method (TF-IDF thesaurus) exceeded that of the state-of-the-art requirements tracing tool in a head-to-head experiment [RE03].

(B) The biggest potential (i.e., measured objectively) improvement in the requirements tracing process occurs when IR methods generating candidate link lists are combined in an iterative process with user feedback (aided by filtering of the candidate link lists) and feedback processing.

3.2 Future Work

Our future work will proceed in a number of directions: (a) study of the subjective aspects of analyst work with RETRO, including the study of usability of the tool; (b) development of more complex methods for candidate link list generation; and (c) extension of the functionality of RETRO and its ability to work with the analysts. Each direction is briefly addressed below.

3.2.1 Usability Feedback Needs

As our work proceeds, we hope to gain valuable information from those who use RETRO and/or the STP tool with our IR toolbox. Some examples of usability feedback of interest follow:

- Is the user interface easy to understand?
- Is the user interface easy to use?
- Are unnecessary steps or mouse clicks required in order to perform work?

- Is the data arranged on the screen in such a way as to facilitate the tracing process?
- How many candidate links are you willing to examine for each high level requirement?
- How many iterations of tracing with feedback are you willing to perform?
- Is it convenient to start working with the tool on a new project?
- Do you have suggestions regarding the input formats for the data?

A comprehensive study of the usability of RETRO, and analyst tendencies in requirements tracing procedures is in preparation.

3.2.2 Planned Future Work

We plan to continue to study feasibility and applicability of different IR methods to the problem of candidate link generation. As mentioned in Section 2.1, two methods are currently in the works. Other methodology, involving more complex supporting artifacts, or models can be evaluated in the future. In addition to evaluation of the accuracy, feasibility of using each particular method must be addressed: generation of certain artifacts may be a time-consuming, human labor-intensive process.

In parallel, we are planning to continue the development of RETRO. Without the benefit of the usability/analyst tendencies study, the current version of RETRO does not take into account a variety of important information that it can easily collect about the *modus operandi* of a specific analyst who is using it. Knowing such information may allow us to better analyze the candidate link lists constructed by the IR toolbox methods and determine filtering techniques and policies to be used, that are likely to improve the work of the analyst. By examining the tracing process with the analyst in the loop, we hope to improve not only the recall and precision of candidate link lists, but also improve the "tracing experience" of the IV&V analyst who is performing the work. This should also contribute to improved answer sets as analysts will be willing to spend more time performing the trace. Improved answer sets will in turn improve the overall IV&V process as so many IV&V activities, such as change impact assessment and testing, depend on the traceability information.

4. References

[Baeza-Yates] Ricardo Baeza-Yates and Berthier Ribeiro-Neto, "Modern Information Retrieval", ACM Press, Addison-Wesley, 1999.

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

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

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

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

[RE03] Jane Hayes, Alexander Dekhtyar, James Osbourne, "Improving Requirements Tracing via Information Retrieval," in Proceedings of the International Conference on Requirements Engineering (RE), Monterey, California, September 2003.

[RE04] Jane Hayes, Alexander Dekhtyar, Senthil Sundaram, Sarah Howard, "Helping Analysts Trace Requirements: An Objective Look," submitted to the International Conference on Requirements Engineering in January 2004, University of Kentucky Technical Report number TR 392-04.

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

Appendix A – Research Papers

Appendix B - Mark-up Language for Candidate Link Lists

On Effectiveness of User Feedback-based Information Retrieval Methods for Requirements Tracing

Jane Huffman Hayes
Department of Computer Science
Lab for Advanced Networking
University of Kentucky
hayes@cs.uky.edu
(corresponding author)

Alex Dekhtyar
Department of Computer Science
University of Kentucky
dekhtyar@cs.uky.edu

Senthil Karthikeyan Sundaram
Department of Computer Science
University of Kentucky
skart2@uky.edu

Sarah Howard
Department of Computer Science
University of Kentucky
sehowa2@uky.edu

Abstract

This paper presents an approach for improving requirements tracing by adding user feedback processing to information retrieval (IR) techniques. Specifically, we focus on improving recall and precision in order to reduce the number of missed traceability links as well as to reduce the number of irrelevant potential links that an analyst has to examine when performing requirements tracing. An iterative user feedback processing method was applied to two IR algorithms to address this problem. We evaluated our algorithms by comparing their results and performance to an oracle (the correct trace results). Initial results suggest that user feedback processing increases recall by about 20% while also decreasing the number of irrelevant potential links. At the same time the quality of candidate link lists improves, with true links occupying more prominent positions.

Research

1. Introduction

Despite the existence and increasing adoption of Computer-Aided Software Engineering (CASE) tools, there are still many software projects for which no requirements trace exists. Lack of this information in a typical development effort hinders debugging, testing, change impact analysis, and cost estimations, just to mention a few. For a safety-critical or mission-critical project, lack of this information could easily halt work. For example, an instrumentation and control (I&C) software subsystem of a nuclear power plant will not pass safety requirements if it cannot prove that all lines of source code emanate from approved requirements. Requirements tracing addresses this.

Unfortunately, requirements tracing is not a pleasant task. There is currently much manual, boring, person time that is required. Take for example the task of tracing a requirement specification to all its children design specifications. Analysts must interactively search through large softcopy requirements specification and alternately search through several design specifications to find potential links or traces. In addition to being distasteful, it is highly error prone work. Automated assistance for this is largely aimed at aiding developers to build the trace as they perform software development. But often that does not occur, and there is a lack of automated tools for assisting analysts who must perform tracing "after the fact."

This research addresses the requirements tracing problem through application of information retrieval methods. Previously, we focused on the problem of generating candidate links, discussed in [11]. Specifically, the tf-idf vector model and simple thesaurus model were applied and found to be effective and efficient. Our focus has now broadened to the overall requirements tracing process. The penultimate goal is to develop an efficient, effective tracing tool that makes the best use of the analyst's time and expertise. A typical requirements tracing process used by independent verification and validation (IV&V) agents includes two stages of interest: candidate link generation and assessment of these links. Existing requirement tracing tools such as SuperTracePlus [10,15], as well as [11], provide assistance for the first stage but not the second.

This work builds on the successes of [11] to address the second stage of candidate link assessment. To that end, we have implemented a feedback method that allows the analyst to evaluate a small subset of the candidate links at each step. This information is used by the tool to generate improved candidate links on the next step. We found that

this approach to candidate link evaluation leads to improvement of the quality of the links and provides a structured way for the analysts to work with the data. In particular, the feedback method allowed us to find around 80% of the links within a trace (e.g., of a requirements specification to its children design specifications) with one in four of the candidates being actual links. The feedback processing method allowed us to improve the percentage of found links by 19% while doubling the signal-to-noise ratio. We have also discovered that minimal interaction with the analyst yields better, or at least as good, results as high interaction. In addition, filtering techniques allowed us to increase signal-to-noise ratio from 1:3 to over 2:1, while not leading to significant decreases in number of true links found.

There are two important metrics in evaluating requirements traces: 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*). Recall and precision values for current tracing methods are not widely known or generalized. This research is aimed at improving the state of the art of after the fact requirements tracing.

IR background, IR methods, and the relevance feedback method applied are presented in Section 2. The tool and the results obtained from evaluation are discussed in Section 3. Section 4 addresses related work. Finally, Section 5 presents conclusions and areas for future work.

2. Information retrieval (IR) for requirements tracing

The main problem studied in the field of Information Retrieval (IR) is determination of relevant documents in document collections given user-specified information needs [8,9]. Most IR methods operate by converting each document in the collection into a mathematical representation that tries to capture the information content of the document and comparing such representations to similar representations of user information needs (queries). The majority of IR methods are keyword-based: the document and query representations incorporate information about the importance of specific keywords found in the document.

2.1 Requirements Tracing as an IR problem

Generally speaking, requirements tracing can be viewed as a problem of document comparison. High- and low-level requirements form two collections of documents. The analyst then compares high-/low-level requirement pairs and for each such pair makes an explicit or implicit similarity judgment.

Such a setup appears to be similar to the basic IR problem. In forward requirements tracing, the low-level requirements form the document collection and the high-level requirements form the set of queries. For each such high-level requirement query, relevant low-level requirements would, ideally, be the ones that trace back to it. In [11] we have studied the applicability of some of the traditional IR methods to the problem requirements tracing. We found that simple IR methods are not enough for robust determination of the trace. Simple keyword-based retrieval using vector model showed insufficient recall and poor precision. We were able to increase the recall to about 85% by using a simple thesaurus of terms and key phrases, however, precision still remained around 30-40%.

Such a showing is hardly surprising. While on the surface requirements tracing shares a lot of common traits with the traditional Information Retrieval tasks, there are also significant differences that make direct applications of IR methods less effective. These differences are:

1. *Size of the domain.* A typical size of a real requirements tracing project does not exceed thousands of requirements for both levels. Smaller projects have the numbers of requirements in hundreds. At the same time, typical IR methods are designed to work on document collections that contain millions of documents. The larger the size of the collection, the more processing time is needed, but also, more information can be gathered about the collection and the vocabulary used will be larger, more diverse.
2. *Size of requirements.* Individual requirements in the requirements and design documents are, typically, no longer than a couple of sentences. Such small texts mean fewer keywords per document in the constructed document representations. Together with smaller number of documents (requirements) themselves, this gives rise to anomalies when one or two casual keyword matches in unrelated requirements get high relevance scores. This stands in contrast with traditional IR domains where documents are much larger and thus have expressive representations not prone to the influence of a single keyword.
3. *Interdependence of requirements.* In a standard IR system all queries issued by users are considered to be independent. The results of one query are not really compared with the results of another query directly (although using collaborative filtering techniques it is possible to suggest relevant documents based on results of prior similar queries). In requirements tracing, because all queries come

from the same document, and some of them represent related requirements, comparison of candidate links for such related requirements may yield extra information. Standard IR methods, however, do not do that.

We can attempt to improve the performance of IR methods in requirements tracing in two ways. The first is implementation of more complex algorithms that take advantage of more than just straightforward keyword matches between the documents. The second way is to use iterative techniques, such as user feedback processing to improve the performance of already implemented methods – our current work.

The use of feedback processing is particularly appropriate for the requirements tracing problem because we believe that despite the clear need to automate the process itself, the final word on the trace has to belong to a human analyst. The use of feedback processing inserts the analyst into the appropriate place in the process of requirements tracing: the analyst will serve as a facilitator of the process and validator of the results.

Given an IR algorithm, the user feedback loop proceeds as follows. On the first iteration, the analyst chooses the high- and low-level requirements and starts the IR algorithm. Once the algorithm produces the candidate trace, the analyst examines it and makes decisions concerning the candidate links. The analyst can vote to include a suggested candidate link in the trace, exclude it from the trace, or leave the status of the link as-is for the moment. At some point, the analyst may stop the examination process and submit the intermediate results. The analyst's choices of relevant (included in the trace) and irrelevant (excluded from the trace) requirements cause the feedback processing part of the loop to change the representation of the high-level requirements. This causes the subsequent rerun of the IR algorithm to find more requirements similar to the ones deemed relevant and relegate requirements found to be similar to the irrelevant ones. The analyst then can continue the same process with the newly obtained and improved list of candidate links. The loop repeats until the analyst is completely satisfied with the trace. The key issue to note here is that the analyst does not have to examine every single candidate link on any of the steps. Standard feedback processing methods show good results for very few answers provided by the user and tend to converge in very few iterations.

2.2 Methods applied

For this study we have used two IR algorithms implemented previously [11]: vanilla vector retrieval, otherwise known as *tf-idf retrieval* and vector (tf-idf) retrieval with a simple thesaurus. On top of these algorithms we have implemented the Standard Rochio [8] feedback processing method.

2.2.1 Tf-Idf model

Standard vector model (also known as tf-idf model) for information retrieval is defined as follows. Each document and each query are represented as a vector of keyword weights. More formally, let $V = \{k1,...,kN\}$ be the vocabulary of a given document collection. Then, a vector model of a document d is a vector $(w1,...,wN)$ of keywords weights, where wi is computed as follows:

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

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 $d=(w1,...,wN)$ and a similarly computed query vector $q=(q1,...,qN)$ the similarity between d and q is defined as the cosine of the angle between the vectors:

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

2.2.2 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. While thesauri organized in such a way are indeed quite simple – they do not contain any ontologies or taxonomies, it is still possible to express a number of important features using them. In particular, one can specify using simple thesauri:

1. Synonyms:
   (error, fault, 1.0)
2. Important key phrases:
   (Greenwich meridian, Greenwich meridian, 1.0);
3. Similar key phrases:
   (sequence of keystrokes, standard input, 0.8);
4. System IDs:
   (Device_Not_Found, Error Message, 0.7);

5. Abbreviations:
(RE, Requirements Engineering, 1.0).

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 consists of more than one keyword) are added as separate keywords to the document collection vocabulary. For example, after processing the second example from the list above, the vocabulary will contain entries for *"Greenwich,"* *"meridian,"* and *"Greenwich meridian."* The weight of a vocabulary entry (single keyword, or a thesaurus term) in a document remains unchanged. However, given a thesaurus $T=\{<ki,kj,\alpha ij>\}$, and document and query vectors $d=(w1,...,wN)$ and $q=(q1,...,qN)$, the similarity between d and q is computed as:

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

2.3 Incorporating relevance feedback from analysts

Relevance feedback is a technique to utilize a users input to improve the performance of the retrieval algorithms. Relevance feedback techniques adjust the vector weights of the query according to the relevant and irrelevant documents found for it, as supplied by the user. The document vectors remain unchanged. The methods for changing the weights are based on the ideal formula:

$$q_{opt} = \frac{1}{R} \cdot \sum_{d_j - relevant} d_j - \frac{1}{N-R} \cdot \sum_{d_k - irrelevant} d_k,$$

where R is the number of documents relevant to a specific information need in the entire document collection and N is the size of the collection.

However, this formula cannot be used since finding the set of relevant documents is the goal of the algorithm. Instead, given a query q, the result of executing an IR algorithm on q and a list of relevant and irrelevant documents contained in it, an approximation needs to be constructed. More formally, let q be a query vector, and Dq be a list of document vectors returned by some IR method given q. Further, assume that D has two subsets: Dr of size R of documents relevant to q and $Dirr$ of size S of irrelevant documents that have been provided by the user. Note that Dr and $Dirr$ are disjoint, but do not necessarily cover the entire set Dq. One of the most popular methods for computing the new representation of query q is Standard Rochio [8]:

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

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 should lead to the inclusion of documents similar to the relevant ones into the answer set on the next step – thus potentially increasing recall. The second adjustment should result in documents similar to the known irrelevant documents getting significantly lower relevance rating and dropping from the answer set, thus potentially increasing precision. The constants a, ß, ? in the formulas above can be adjusted in order to emphasize such 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 run with the modified query vectors. This cycle can be repeated until the user is satisfied with the results. After all of the vectors for the high level requirements have been modified, the retrieval formulas are computed again, and the new results are displayed.

3. Evaluation

This section presents the prototype tool, the tests used to evaluate our approach, and a discussion of results.

3.1 Requirements tracing tool

A prototype requirements tracing tool has been built and used in the tests. The tool implements three IR methods, as described in [11], and Standard Rochio feedback processing. Only two of the IR methods have been used in this study (see Section 2.2). For each method, the tool implements three tasks: a) building document models, b) building query models, and c) generating candidate link lists. Feedback processing can be used with any of the IR methods. The front-end to the tool has been implemented in Java, whereas all the IR algorithms and feedback processing are implemented as a C++ toolkit with uniform APIs, allowing for simple extensions of the toolkit with implementations of new methods. XML has been used for uniform data transmission between tool components. Use of the tool for the study is further described in Section 3.2.

3.2 Study Design

In our study, we used slightly modified implementations of tf-idf and simple thesaurus retrieval algorithms applied in [11]. The improvements were mostly internal and did

not affect the output of the methods. These two algorithms have been incorporated in a prototype requirements tracing tool. In addition, the relevance feedback loop using Standard Rochio method has been implemented within the tool.

To assess the effectiveness of requirements tracing using relevance feedback method, we performed tests of both the tf-idf and thesaurus approaches. We used a modified dataset from [11] based on open source NASA Moderate Resolution Imaging Spectroradiometer (MODIS) documents [12,14]. The dataset contains 19 high level and 49 low-level requirements. The trace for the dataset was manually verified and 42 correct links were found.

To ensure that we evaluated the effect of relevance feedback processing on recall and precision and not on analyst ability to correctly select matches from a candidate match list, we designed the tests as follows. Graduate student volunteers (analysts) used the verified trace (the "right answers") to provide feedback to our tracing tool. The study design is described in Table 1. At the beginning of each test, the analyst loaded the dataset into the tool and selected the IR method: tf-idf or simple thesaurus. The method was run on the dataset and the results were displayed for the user to observe and modify.

Analyst interaction with the requirements tracing tool via the feedback loop was guided by the *Behavior* parameter. *Top i* behavior means that at each iteration, the analyst was to correctly mark the top *i* unmarked candidate links from the list for each high-level requirement. For example, for each high level requirement, the analyst implementing Top 1 behavior examined the top candidate link suggested by the IR procedure. If this link had not been marked yet, the analyst determined whether the link was correct (using the answer set provided), and selected a "link" or "no link" choice to provide relevance feedback to the system. If the top link had already been marked as "link" on previous steps, the analyst looked for the first unmarked link in the list for that requirement. After repeating the *Top i* relevance feedback procedure for each high level requirement, the analyst submitted the answers to the requirements tracing tool. The tool processed relevance feedback using Standard Rochio procedure and submitted the new queries to the IR method. New results were reported to the analyst, starting a new iteration. The process continued for eight iterations or until the analyst noticed that the results had converged.

The key question the tests were designed to answer is:

- How does the quality of the list of candidate links change from iteration to iteration?

Table 1. Experiment Design.

IR method	Feedback method	Behavior	# Iterations
Tf-idf	Standard Rochio	Top 1 Top 2 Top 3 Top 4	8
Thesaurus	Standard Rochio	Top 1 Top 2 Top 3 Top 4	8

We evaluated the quality of the list of candidate links using a number of metrics. In particular, we considered the following questions:

- How many real links are found and how many false positives are returned?
- What is the structure of the list of candidate links? Are true links more prominent than false positives?

The first question was addressed by computing precision and recall for each iteration of each test. To answer the second question we have introduced a number of secondary measures, designed specifically to compare and contrast the positions and/or relevances of true links and false positives. The metrics considered are:

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

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

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

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

3.3 Results

The precision and recall results obtained in our tests are summarized in Table 2. In each cell, precision is indicated first followed by recall. 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 1, which contains the precision/recall trajectories for all experiments, grouped by the IR method used (top: tf-idf, bottom: simple thesaurus). From these results we can make the following observations:

- Both tf-idf and simple thesaurus retrieval,when used without feedback, produce moderately high recall(57.1% for tf-idf and 64.2% for simple thesaurus), but the precision is very low: 11.3% for tf-idf and 12.2% for simple thesaurus.

Table 2. Results of Experiments: Precision and Recall.

I	Top 1		Top 2		Top 3		Top 4	
	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus
0	11.3%, 57.1%	12.2%, 64.2%	11.3%, 57.1%	12.2%, 64.2%	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%	7.3%, 61.9%	7.8%, 66.6%	8.3%, 69%	8.6%, 76.1%
2	7.7%, 59.5%	8.3%, 64.2%	9.2%, 69%	9.9%, 73.8%	11.6%, 73.8%	10.6%, **83.3%**	12.1%, **76.1%**	12.4%, **80.9%**
3	9.2%, 66.6%	8.6%, 61.9%	12.2%, **73.8%**	12.1%, 80.9%	14.6%, 73.8%	13.6%, 80.9%	15.1%, 73.8%	15.3%, **80.9%**
4	9.9%, 71.4%	10.6%, 71.4%	15.1%, **73.8%**	15.7%, **83.3%**	18.6%, **76.2%**	17.4%, **83.3%**	13%, 61.9%	16.7%, 78.5%
5	12.5%, **76.1%**	12.5%, **78.5%**	17.9%, 71.4%	15.8%, 80.9%	17.1%, 71.4%	18.1%, **83.3%**	19.4%, 73.8%	**18.6%**, 78.5%
6	**15.3%**, **76.1%**	14.8%, 76.1%	20%, 69%	17.6%, **83.3%**	20.9%, 73.8%	20.2%, 80.9%	**22.7%**, 71.4%	
7		16%, 71.4%	23.3%, 69%	21.9%, 80.9%	21.4%, 69%	**24.6%**, 80.9%	**22.7%**, 71.4%	
8		**17.3%**, 73.8%	**25.6%**, 69%	**25%**, 78.5%	**22.7%**, 66.6%			

- With relevance feedback, both tf-idf and simple thesaurus retrieval exhibited similar recall patterns. Over the course of the iterations, the recall increased by about 19%, before either stabilizing or somewhat decreasing in some cases. The drop-off in recall was smaller for the simple thesaurus retrieval method.
- With relevance feedback, both tf-idf and simple thesaurus retrieval exhibited similar precision patterns. On early iterations, precision decreased slightly. But starting with iteration 3, it would grow monotonically with each new iteration. For Top 2 and Top 3 behaviors, precision would eventually eclipse 20% --- a 90%-100% increase from the original precision values.
- The quality of answers obtained for behaviors with little interaction with the system (Top 1 and Top 2) was as good, or better than, that for high-interaction behaviors (Top 3 and Top 4). In fact, for both tf-idf and simple thesaurus retrieval, the highest overall precision numbers were obtained at iteration eight in the Top 2 behavior. The best overall recall numbers were obtained at iteration 5 in Top 1 behavior for tf-idf and at iterations four and six in Top 2 behavior for simple thesaurus.?
- In all tests the major quantitative jump in recall occurs fairly early on: at iterations 2, 3 or 4 (this can be seen on the trajectories of all experiments in Figure 2). After this jump, the recall tended to level off while precision continued to grow.

As seen from the results and discussion above, while the use of relevance feedback allowed us to achieve high (up to 83%) recall, the precision continued to be the Achilles heel of the approach: even in the best cases, only 1 out of every 4 - 5 candidate links was correct. To see if precision can be improved without significant damage to recall, we have applied a number of filters, designed to throw out some of the candidate links from the list produced on each iteration. In particular, we looked at the following three filters:

Table 3. Filtering Summary.

		Top 1		Top 2		Top 3		Top 4	
TF-IDF		Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
No Filter	Best	*15.3*	*76.1*	*25.6*	*73.8*	*22.7*	*76.1*	*22.7*	*76.1*
Above 0.1	Best	46	54.7	38.2	61.9	35.4	52.3	35.2	57.1
	Difference	**30.7**	**-21.4**	**12.6**	**-11.9**	**12.7**	**-23.8**	**12.5**	**-19**
Within 0.5	Best	42.4	33.3	48.7	45.2	46.6	50	40.4	45.2
	Difference	**27.1**	**-42.8**	**23.1**	**-28.6**	**23.9**	**-26.1**	**17.7**	**-30.9**
Top 42	Best	61.5	57.1	64.8	57.1	60.9	59.5	61.9	61.9
	Difference	**46.2**	**-19**	**39.2**	**-16.7**	**38.2**	**-16.6**	**39.2**	**-14.2**
Thesaurus									
No Filter	Best	*17.3*	*78.5*	*25*	*83.3*	*24.6*	*83.3*	*18.6*	*80.9*
Above 0.1	Best	44.4	61.9	37.9	71.4	39.5	80.9	34.8	71.4
	Difference	**27.1**	**-16.6**	**12.9**	**-11.9**	**14.9**	**-2.4**	**16.2**	**-9.5**
Within 0.5	Best	63.6	33.3	54.7	54.7	53	61.9	40	57.1
	Difference	**46.3**	**-45.2**	**29.7**	**-28.6**	**28.4**	**-21.4**	**21.4**	**-23.8**
Top 42	Best	64.1	59.5	71.7	66.6	73.8	73.8	61.9	61.9
	Difference	**46.8**	**-19**	**46.7**	**-16.7**	**49.2**	**-9.5**	**43.3**	**-19**

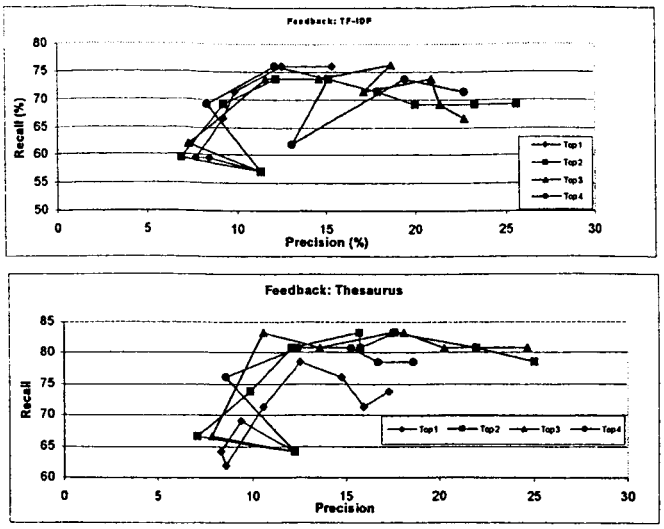


Figure 1. Recall, Precision for All Behaviors and IR Methods.

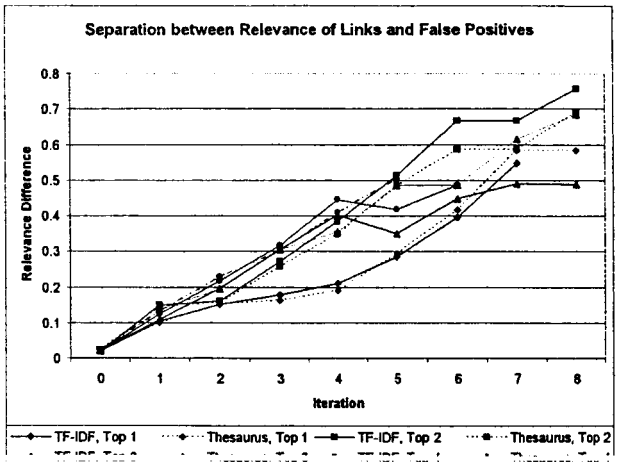


Figure 2. Separation Between Average Relevance of Links and False Positives.

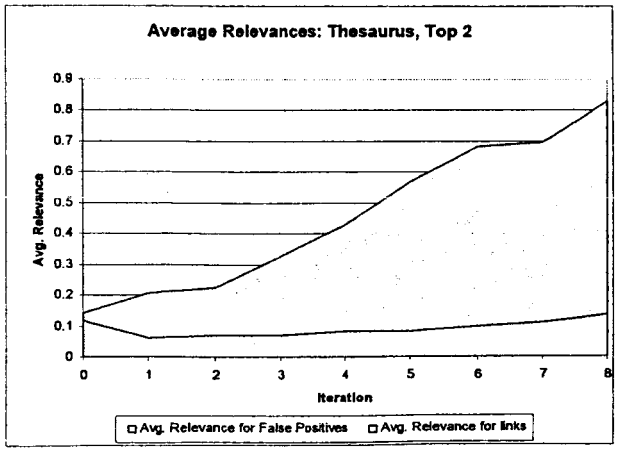


Figure 3. Average Relevances: Thesaurus, Top 2.

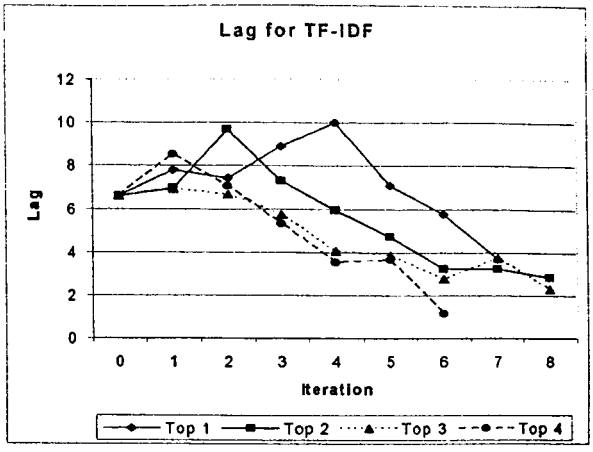


Figure 4. Lag for TF-IDF Tests.

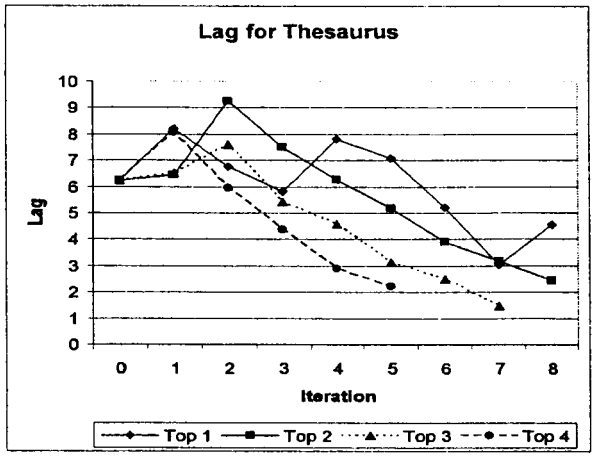


Figure 5. Lag for Thesaurus Retrieval Tests.

? *Above 0.1.* A candidate link was thrown out if its relevance was below 0.1.

– *Within 0.5.* For each high-level requirement, the relevance of each candidate link was compared with the relevance of the top candidate link. The candidate link was retained *iff* its relevance was within 0.5 of the relevance of the top link.

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

Table 3 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 had always been 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. For tf-idf retrieval, *Above 0.1* and *Within 0.5* filters produce precision of 35-50%, with typically a two-fold increase from the "vanilla" list. At the same time, recall suffers a significant (20-42%) decrease, with the only notable exception being *Above 0.1* for Top 2 behavior (a drop of 11.9%). *Top 42* filter, as expected, shows even bigger improvement in precision, with the drop in recall being around 14-17% for all cases but Top 1. Notice that for this filter, the precision is expected to be equal to recall, or slightly exceed it if the size of the filtered list of candidate links is smaller than 42. More importantly, for the simple thesaurus retrieval, both *Above 0.1* and *Top 42* demonstrate an increase in precision with only moderate penalty in recall. The best results were obtained for Top 3 behavior where for both filters the decrease in recall was less than 10%, while the increase in precision was over 50% for *Above 0.1* and exactly 300% for *Top 42.*

Filtering the lists of candidate links is one of the ways to study the inner structure of the lists of candidate links: the filters work well when disproportionately many "bottom" candidate links are false positives. Thus, the better the degree of separation between the true links and the false positive links in the lists, the more effective the filters will be in increasing precision without compromising recall. We have therefore also used other metrics to study our progress in the degree of separation in the lists generated.

Figure 2 shows the changes in the *DiffAR* metric: the difference in average relevances between the true links (*ART*) and false positives (*ARF*). 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-

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. In most cases, the value of *ART* monotonically grows from iteration to iteration. At the same time, the value of *ARF* drops by about one half on the first iterations of each test, then slowly grows back towards its original value (around 1.1), sometimes slightly exceeding it. Figure 3 shows the progress of *ART* and *ARF* metrics for simple thesaurus retrieval with Top 2 behavior.

While *ART, ARF* and *DiffAR* measure 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. High-interaction behaviors (Top 3 and Top 4) appear to produce better (smaller) *Lags:* the final iterations of these methods for tf-idf give *Lags* of 2.28 and 1.13, while for thesaurus retrieval they are 1.47 and 2.21.

3.4 Discussion of results

During the tests we have established that for both of the IR methods, using relevance feedback mechanisms consistently improves recall by just under 20%. We have found that precision also improves, occasionally by as much as 100%. But, because the starting precision is rather small, the improvement is not significant in absolute numbers. However, we found that application of filtering techniques is promising. In many cases, filtering resulted in drastic improvement in precision, while the decrease in recall was not very significant. Our tests have also produced evidence that with each iteration, the generated lists of candidate links tend to be of better overall quality: true links rise to the top, while false positives tend to sink to the bottom. Also, the gap between relevance weights for true links and false positives grows from insignificant (0.02) to large (0.6-0.8 in most cases). This suggests that analyst effort in providing relevance feedback pays off: it is possible to generate lists of candidate links with high precision and high recall, saving analyst time during the final verification of the trace.

4. Related work

There are two areas of interest: requirements tracing and IR as it has been applied to the problem of requirements analysis. Each is addressed below.

4.1. Requirements tracing

We have been tackling the requirements tracing problem for many decades. In 1978, Pierce [16] 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 [10] 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 [15]. 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.

Abrahams and Barkley, Ramesh, and Watkins and Neal [1, 17, 22] discuss the importance of requirements tracing from a developer's perspective and explain basic concepts such as forward, backward, vertical, and horizontal tracing. Casotto [6] examined run-time tracing of the design actvity. Her approach uses requirement cards organized into linear hierarchical stacks and supports retracing. Tsumaki and Morisawa [21] 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) [21].

There have also been significant advances in the area of requirements elicitation, analysis, and tracing. Work has [illegible] analysis of phoneme occurrences to categorize and analyze requirements and other artifacts [19]. Bohner's work on software change impact analysis using a graphing technique may be useful in performing tracing of changed requirements [4]. Anezin and Brouse advance backward tracing and multimedia requirements tracing in [2,5].

Cleland-Huang et al [7] 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. 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 [18]. They found that a typical low end user created traceability links to model requirement dependencies, to examine how requirements had been allocated to system components, to verify that requirements had been satisfied, and to assist with change control. A typical high-end user, on the other hand, uses traceability for full coverage of the life cycle, includes the user and the customer in this process, captures discussion issues, decision, and rationale, and captures traces across product and process dimensions [18].

4.2 IR in requirements analysis

Recently, a number of research groups has considered using Information Retrieval methods for various problems in requirements analysis. Two research groups, in particular, worked on the requirements-to-code tracebility. Antonio, 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 top *i* matches for each requirement for *i=1,2,...,* 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 [13] applied 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.

While [3] and [13] studied requirements-to-code traceability, in [11] we have addressed the problem of tracing requirements between different documents in the project document hierarchy. In the preliminary study [11] we have implemented three methods: tf-idf, tf-idf with key phrases and tf-idf with simple thesaurus and have reported [illegible] requirements documents. In our study, retrieval with simple thesaurus outperformed other methods on our test dataset, producing recall of 85% with precision 40%. This work continues the research started in [11]. Here, we extend the baseline tf-idf and thesaurus retrieval methods with analyst relevance feedback processing capability.

5. Conclusions and future work

In this paper we have studied the effect of relevance feedback processing on the success of IR methods for requirements tracing. We have found that taking into account even limited user feedback results in consistent, and at times, significant increases both in precision and recall on subsequent iterations.

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

a. implementation of more intricate IR algorithms;

b. a comparative study of different relevance feedback techniques;
c. study of the work of analysts in requirements tracing.

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

Acknowledgments

Our work is funded by NASA under grant NAG5-11732. 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.

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

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

[7] 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.

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

[9] Frakes, W. and Baeza-Yates, R. (Eds.), *Information Retrieval: Data Structures and Algorithms*, Prentice Hall, 1992.

[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," accepted to the International Conference on Requirements Engineering, to be presented in Monterey, California, September 2003.

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

[13] 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.

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

[15] 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.

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

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

[18] 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.

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

[20] Sparck Jones, K. and Willet, P. *Readings in Information* Retrieva,l Morgan Kaufmann Series in Multimedia Information and Systems, Morgan Kaufmann, 1997.

[21] 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.

[22] Watkins, R. and Neal, M. "Why and How of Requirements Tracing," *IEEE Software*, Volume 11, Issue 4, July 1994, pp. 104-106.

University of Kentucky Technical Report TR 392-04

Helping Analysts Trace Requirements: An Objective Look

Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, Sara 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 insure 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*). The analyst reviews the candidate links and determines which are actual links (candidate link evaluation). Finally, 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. To that end, this paper provides three contributions: we investigate the analyst responsibilities in performing tracing; we derive tool requirements from these; and, we present a prototype tool, RETRO, 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 may 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. 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.

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 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. Each are 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 will possess 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" 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). 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 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 will decide whether candidate links are correct or not and will decide whether to search for additional candidate links. The analyst must feel confident that good results have been achieved by using the tool.

Scalability will not be addressed in this paper as we do not currently have large tracesets with a "true trace." Accuracy will be 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 give a clear 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 [illegible] "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 [illegible] information is exported in a simple, easy-to-parse XML form. The overall look of RETRO GUI (Win32 port) is shown in Figure 1.

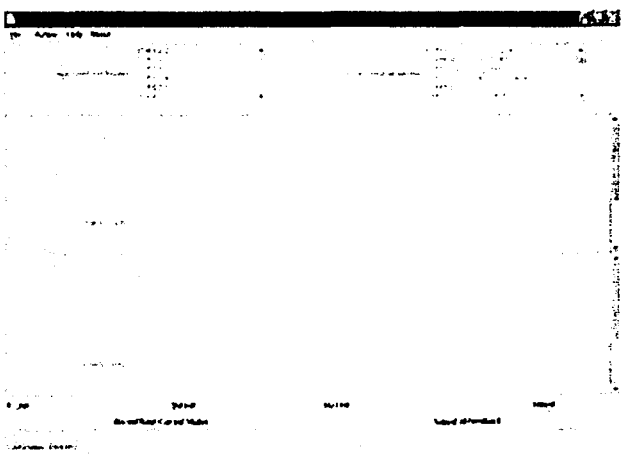


Figure 1. A screenshot of RETRO.

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 $V = \{k1,...,kN\}$ be the vocabulary of a given document collection. Then, a vector model of a document d is a vector $(w1,...,wN)$ 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 $d=(w1,...,wN)$ and a similarly computed query vector $q=(q1,...,qN)$ 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 Dq be a list of document vectors returned by some IR method given q. Further, assume that D has two subsets: Dr of size R of documents relevant to q and $Dirr$ of size S of irrelevant documents that have been provided by the user. Note that Dr and $Dirr$ are disjoint, but do not necessarily cover the entire set Dq. 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 a, ß, ? 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 and 42 correct links were found.

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 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 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 in the list). The *Lag* of a list of candidate links is the average *Lag* of its true links.

Table 1. Results of experiments: Precision and recall.

I	Top 1		Top 2		Top 3		Top 4	
	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus	Tf-IDF	Thesaurus
0	11.3%, 57.1%	12.2%, 64.2%	11.3%, 57.1%	12.2%, 64.2%	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%	7.3%, 61.9%	7.8%, 66.6%	8.3%, 69%	8.6%, 76.1%
2	7.7%, 59.5%	8.3%, 64.2%	9.2%, 69%	9.9%, 73.8%	11.6%, 73.8%	10.6%, **83.3%**	12.1%, **76.1%**	12.4%, **80.9%**
3	9.2%, 66.6%	8.6%, 61.9%	12.2%, **73.8%**	12.1%, 80.9%	14.6%, 73.8%	13.6%, 80.9%	15.1%, 73.8%	15.3%, **80.9%**
4	9.9%, 71.4%	10.6%, 71.4%	15.1%, **73.8%**	15.7%, **83.3%**	18.6%, **76.2%**	17.4%, **83.3%**	13%, 61.9%	16.7%, 78.5%
5	12.5%, **76.1%**	12.5%, **78.5%**	17.9%, 71.4%	15.8%, 80.9%	17.1%, 71.4%	18.1%, **83.3%**	19.4%, 73.8%	**18.6%**, 78.5%
6	**15.3%**, **76.1%**	14.8%, 76.1%	20%, 69%	17.6%, **83.3%**	20.9%, 73.8%	20.2%, 80.9%	**22.7%**, 71.4%	
7		16%, 71.4%	23.3%, 69%	21.9%, 80.9%	21.4%, 69%	**24.6%**, 80.9%	**22.7%**, 71.4%	
8		**17.3%**, 73.8%	**25.6%**, 69%	**25%**, 78.5%	**22.7%**, 66.6%			

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

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.

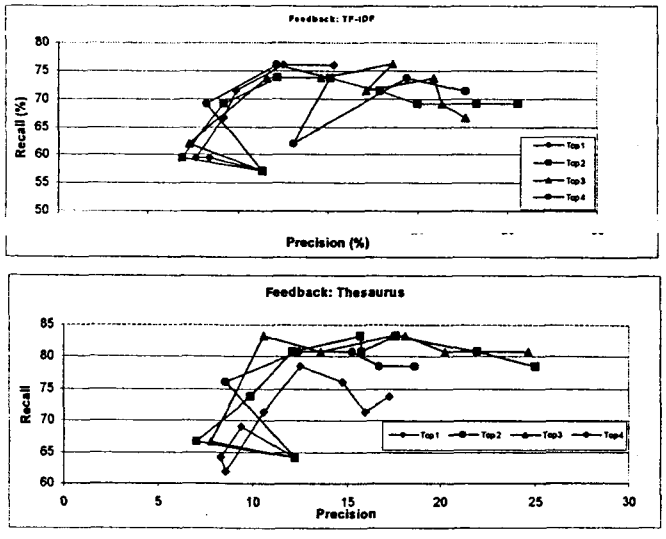


Figure 2. Recall, precision for all behaviors and IR methods.

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.

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

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.

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.

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

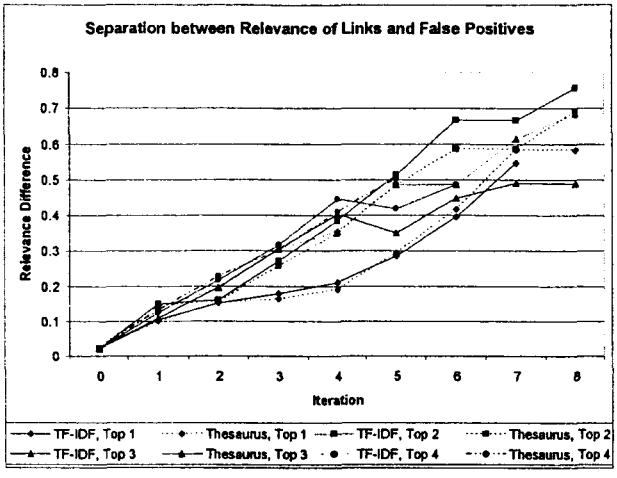
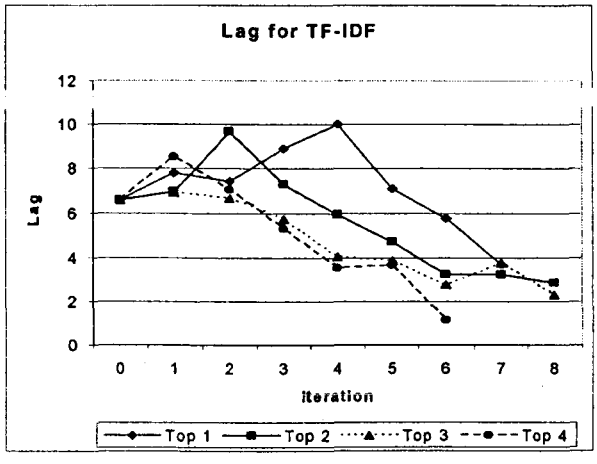
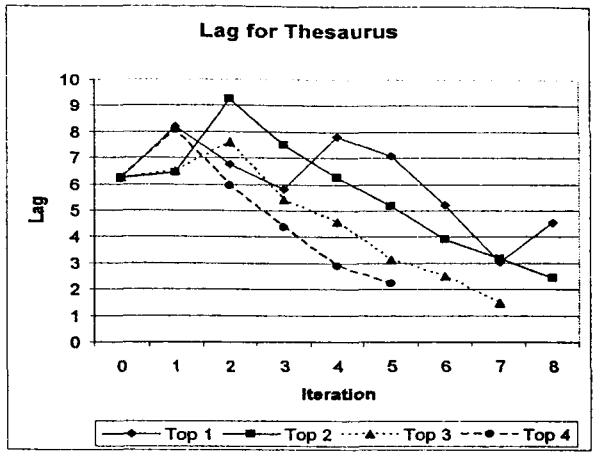


Figure 4. Lag for TF-IDF tests.



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 5. Lag for thesaurus retrieval 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.

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.

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] 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] Daeza-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 [illegible] 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.