Advances in Dynamic Generation of Traceability Links: Two Steps Closer to Full Automation?

Jane Huffman Hayes
Dep. of Computer Science
University of Kentucky
Lexington, KY 40506
hayes@cs.uky.edu

Alex Dekhtyar
Dep. of Computer Science
University of Kentucky
Lexington, KY 40506
dekhtyar@cs.uky.edu

Senthil Sundaram
Dep. of Computer Science
University of Kentucky
Lexington, KY 40506
skart2@uky.edu

ABSTRACT

The generation of traceability links and/or of a requirements traceability matrix (RTM) is vital to many software engineering enterprise activities. It is also person-power intensive, time-consuming, and error-prone, and lacks tool support. In this paper, we investigate the application of several new information retrieval techniques to the problem of RTM generation. Specifically, we investigate if the analyst's workload can be reduced, and we study the impact of vocabulary base on tracing results.

1. INTRODUCTION

The daily work of many of those in a software development enterprise can benefit from the dynamic generation of traceability links. For example, a third party independent verification and validation (IV&V) analyst must develop a requirement traceability matrix (RTM) for a set of artifacts delivered by the developer. A verification and validation (V&V) analyst must trace two interface specifications to each other to assure consistency. A maintenance programmer must recover traceability links between a modified set of source modules and their textual documentation. A software development manager must build a mapping from a contractual list of required software functions to the technical and cost proposal being prepared for the customer.

Unfortunately, the process of generating such traceability links has largely been manual, tedious, and error prone. Advances in traceability have been realized through the application of Information Retrieval (IR) methods. Many open questions remain in this area, however. This paper seeks to address two such questions: 1) can the work of the analyst be minimized? and 2) does the vocabulary base effect traceability generation results? To put these questions in context, we first examine the historical evolution of traceability approaches, focusing on the analyst tasked with developing an RTM.

1.1 Traceability, Technology, and the Analyst

The approaches to generating traceability links have largely been a bi-product of the currently available technology. To illustrate, let us consider a simple case of a V&V analyst attempting to trace an interface requirements specification to a software requirements specification[1]. In the earliest days, analysts recorded each element of interest (in this case, each "shall" statement from the software requirement specification and interface requirement specification) on a notecard. The notecards were kept in an index card box. One artifact (in this case, the software requirement specification) was viewed as the "parent" document and the other artifact as the "child." The analyst would read a card for one parent element and then would read through the cards for the child document, pulling out matches. The matches would then be filed behind the card for that parent element. This approach posed many problems for the analyst: 1) it required an M x N search of the artifact space (where there are M parent elements and N children elements); 2) it required the analyst to prepare duplicate cards of some of the children elements (if a child element helped satisfy more than one parent element); and 3) it did not result in an RTM that was easy to search.

The advent of Hypercard (and the subsequent Supercard) technology brought an automation of the notecard-based approach. This addressed the problem of duplicate cards and a searchable RTM, but still left the analyst with an M x N task. The emergence of relational data base management systems ushered in the era of keyword-based matching and searching to assist with link generation. Analysts would assign a set of one or more keywords to every element in the parent and child artifact. Then, matching algorithms were executed to generate potential or candidate links for each parent element. Often, thresholds were applied for the matching (e.g., "if 50% or more of the parent keywords exist in the child, return the item as a candidate link"). The analyst then examined the candidate links returned for each parent element. Although the analyst was no longer reading the M x N set of elements, the analyst was still "touching" all M x N elements due to keyword assignment.

Next came the hierarchical keyword list and "legend." Here, the analyst built a detailed list of keywords and a set of "terms or phrases" for assigning those keywords. For example, the analyst might specify that the existence of any of the words "error," "failure," or "bug" should result in the automatic assignment of the keyword "error" to an element. This eliminated the need to "touch" all M x N elements for keyword assignment, but replaced the task with a large one-time task of building the keyword hierarchy and legend.

The next major advance in this area came with the arrival of the internet and the ubiquoutous search engine. As in the past, traceability again found itself influenced by the latest technology. Antoniol et al. [3] were the first to apply information retrieval (IR) tech-

[1]The following historical discussion is based on the first author's 22-year experience with tracing

niques to the problem of dynamically generating traceability links. They applied probabilistic information retrieval and vector retrieval (term frequency - inverse document frequency (tf-idf)) and studied traceability between code and documetation for two datasets. Marcus and Maletic [18] applied latent semantic indexing (LSI) to the same two datsasets. Our group [13, 14] conducted a number of preliminary studies on the use of IR methods for requirements tracing. In [13], we compared the results of using tf-idf and tf-idf augmented with a simple thesaurus to the results obtained from a leading industrial tool [16, 24] and from an analyst working with it. In [14], we reported preliminary results on the use of user feedback for tracing. In [**?**], we provide baseline results for tf-idf, tf-idf plus simple thesaurus, and LSI on two public datasets: MODIS [2, 1] and CM-1 [20] (the first two papers used only the smaller, MODIS dataset). Cleland-Huang et al. [7] applied probabilistic IR to the problem of tracing UML diagrams [2] to functional requirements (their group has also looked at how tf-idf performs on this task).

Another important trend in tracing to consider is the application of resources. Generally, senior analysts are more expensive and in less supply than junior analysts. Typically, managers use their senior analysts for the most demanding and “important result” yielding tasks. It is, therefore, very common for junior analysts to be assigned the job of building RTMs. The analysts may have very little experience in the domain area or project on which they are working. They are trusted to be able to make good decisions on the relevance of candidate links that are returned by retrieval methods. But they are generally not trusted to make decisions on whether or not parent elements have been fully satisfied by children elements. This is typically left to senior analysts.

1.2 Vocabulary Base

In a standard IR approach, only the low level artifact is used to create the vocabulary base or corpus (the collection of terms or keywords that will be used to automate traceability link generation). In Internet searching, the multitude of existing web pages have been used to build a vocabulary base so that a surfer’s query can be handled quickly. In such an application, we do not know the query until it is entered by the user. In the dynamic generation of traceability links for software engineering, we have all the ”querries” at the start. A natural question, then, is whether we should use the high and low level artifacts to build a vocabulary base. This has not been addressed by other researchers. Antoniol et al. [3] and Cleland-Huang et al. [7] do not address the vocabulary base used in their studies. Marcus and Maletic [18] use the high and low-level artifacts for the vocabulary base.

1.3 Contribution

To more closely examine the problem of tracing from the analyst’s perspective, our group developed a list of requirements for an effective tracing tool [14]. The list included: believability, discernability, and endurability. In this paper, we take a closer look at the dynamic link generation process and the role of IR methods in that process, from the analyst’s view. By breaking the tracing process down into steps, we can determine where the analyst is essential and where the analyst can be eliminated. We can then evaluate the success of IR methods to date with respect to the process steps.

To that end, we have examined several new IR methods as well as baseline methods for their comparison. Specifically, we have examined a family of vector retrieval methods (we introduce Okapi and linear threshold unit (ltu) as baseline vector methods), LSI, and keyword extraction method (we introduce the keyword extraction method as well as a baseline comparison method using document frequency). We have applied analyst feedback to the methods. We also compare the use of low level artifact versus high and low level artifacts for vocabulary base. As a result, this paper contributes the following:

- an examination of the tracing process and required analyst input;
- application of new IR methods (Okapi, LTU, keyword extraction, keyword extraction by document frequency) with analyst feedback;
- examination of the effect of vocabulary base; and
- results that support the elimination of two steps from the tracing process.

We use three diverse datasets for testing the methods.

1.4 Paper Organization

The paper is organized as follows. In Section 2, we take a look at the different steps of the tracing process and the analyst’s participation. Section 3 briefly describes the Information Retrieval methods used in the current study. We describe the study and the results we have obtained in Section 4. Conclusions and future work are addressed in Section 5.

2. THE STEPS OF THE TRACING PROCESS: ANALYST REQUIRED?

The tracing process has several steps, as introduced in [14]. We present a more detailed look at the steps below. Note that we assume that the elements have already been extracted from the artifacts.

The potential steps for tracing are:

Step 1: build keyword hierarchy and legend;

Step 2: build probabilistic model;

Step 3: build thesaurus;

Step 4: assign keywords to elements;

Step 5: examine candidate links and render decisions.

Once the RTM is built:

Step 6: examine parent elements and their children and determine satisfaction;

Step 7: look for and address orphans (children elements with no parents).

For now, we only address the building of the RTM, so we do not consider steps 6 and 7. Let us examine the “effort” required by an analyst, depending on the IR method applied. (Note that some steps are optional depending on the IR method.) Below, we list the steps required for each method.

Method	Steps
Vector space retrieval	step 5
Vector space retrieval plus thesaurus	step 3, step 5
Vector space retrieval plus user feedback	step 5
Probabilistic IR	step 2, step 5
LSI	step 5
Keyword extraction	step 5

As described in the Introduction, the application of IR methods has effectively eliminated steps 1 and 4 from the dynamic link generation process. Let us examine the remaining steps a bit closer.

[2]UML diagrams, non-textual artifacts, were represented as bags of words extracted from the diagrams.

	experienced analyst	inexperienced analyst
high level of effort expended	Excellent results [7]	unknown
low level of effort expended	unknown	Poor results [15]

Table 1: Building Probabilistic Models

2.1 Step 2: Build Probabilitic Model

It is an accepted fact that the accuracy of the probabilistic IR method is directly tied to the quality of the probabilistic model that has been developed. We have seen this illustrated in two research results: Cleland-Huang et al. [7] used an experienced statistician to build a probabilistic IR model and excellent results were seen; and a junior analyst was used to build a probabilistic model and very poor results were obtained [15]. From a researcher's viewpoint, this represents a multi-dimensional space to be explored as shown in Table 1 [3].

From a manager's perspective, the first priority is to eliminate unnecessary steps for tracing. The second priority is to reduce the amount of time needed of a senior analyst. Based on this and Table 1, it makes sense to examine the use of an experienced analyst who applies a small amount of time to the building of a probabilistic model (bottom left cell). This is an area for future work. Examining the use of an inexperienced analyst (top right cell) might represent an area of future study. Another option is to select IR methods that do not require this step at all.

2.2 Step 3: Build Thesaurus

Our group [4] has examined the use of a thesaurus (lists synonyms for a thesaurus entry, such as "input," "data entry," that should all map to "input") to augment a number of IR methods. For the small MODIS dataset (19 parent elements x 49 child elements), a senior analyst spent only 20 minutes building a thesaurus. The thesuarus greatly enhanced the tracing results, increasing accuracy [5] by as much as 10–15%. The CM-1 dataset is much larger, 235 x 220 elements, and required one hour of a junior analyst's time in order to build a thesaurus. The CM-1 thesaurus did not result in improved results, however. This lends support to the findings for Step 2, the tracing results are only as good as the thesaurus (see Table 2).

Again, we are interested in not using the senior analyst at all, or as little as possible. As shown in Table 2, those cells have been explored. A possible future study could look at an inexperienced analyst expending high effort (top right cell). Another option is to examine methods that perform well without a thesaurus.

2.3 Step 5: Examine Links

This step cannot be alleviated. The final decision of relevance always remains in the hands of the human analyst.

Based on the analysis of Sections 2.1 and 2.2, it appears that a method that does NOT require the assigning of keywords, the building of keyword lists, the building of probabilistic models, or the building of thesauri is desirable. If such a method can be found, with satisfactory accuracy levels, it can help reduce the analyst's workload.

We now introduce the IR methods and measures used in the study.

[3] Note that only two data points exist to date, therefore we present indications only

[4] To our knowledge, no other researchers have investigated the use of a thesaurus.

[5] Accuracy is measured as recall (% of relevant links found) and precision (% of retrieved links that are relevant.)

3. METHODS AND MEASURES USED

3.1 Methods

In this section, we briefly describe the IR methods we used to generate candidate link lists in our study. Note that in IR terminology, a given element that we are tracing is called a "query" and we search for it in a "document collection." So a trace from a requirement specification to a design document uses each requirement element as a query that is searched for in the "document collection" of design elements. Before applying each method to build the appropriate representations of the high-level and low-level elements, we have used the standard lexical analysis—stopword removal—stemming process. We used a standard list of stopwords, enhanced with some special stopwords such as "NASA" for specific datasets. Porter's algorithm [21] was used for stemming. Information Retrieval methods described below work as follows. Given a (parsed and stemmed) document (requirement, design element), a vector of keyword weights is built [6]. The weight of each keyword indicates the percieved importance of the keyword for the document. Vector space retrieval methods assign weights to all keywords encountered after stopword removal and stemming. Latent Semantic Indexing transforms such vectors into shorter ones, in which each column corresponds to an "implicit" concept within the document collection. Keyword extraction methods determine which of the words found in the requirements/design elements are "important" and which are not, and keep only the weights of "important" keywords.

3.1.1 *Vector space retrieval*

Our first set of methods is standard vector retrieval. We have implemented two ways of building the vocabulary list (keywords) and three term weighting schemes (where "term" is defined as a keyword) for a total of six variants of vector retrieval. The following term weighting schemes were implemented.

Tf-idf. An element d is represented as a vector of keyword weights $d = (w_1, \ldots, w_N)$.The weight w_i of each keyword k_i is computed as a product, $w_i = tf_i(d) \cdot idf_i$, of k_i's *term frequency*, $tf_i(d)$, a normalized number of occurrences of k_i in d and *inverse document frequency*, idf_i, computed as $idf_i = log_2\left(\frac{n}{df_i}\right)$, where n is the total number of elements in the collection and df_i is document frequency of term k_i, i.e., the number of documents in which k_i is found [4]. The idea behind this formula is straightforward: the more often a word occurs in a document, the more important it is for the document (the $tf_i(d)$ term), and the more common the word is, the less discriminatory power it possesses, and therefore, the less important it is (the idf_i term).

The same formula for computing weights was used for both high- and low-level elements. Given vectors of keyword weights $q = (q_1, \ldots, q_N)$ and $d = (w_1, \ldots, w_N)$, the similarity between them

[6] Though the phrase "keyword" might bring to mind manual assignment, note that this is an automated process.

	experienced analyst	inexperienced analyst
high level of effort expended	unknown	unknown
low level of effort expended	Excellent results (MODIS)	Poor results (CM-1)

Table 2: Building Thesaurus

Iteration 0:

Voc. Base:	**Low-level artifact**						**Low-level+High-level artifacts**					
Dataset	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	97.8	1.6	50.7	2.4	98.3	7.5	97.8	1.6	49.9	2.4	98.3	7.5
Modis	75.6	7.9	54.7	10.7	76.9	28.0	75.6	7.9	74.2	10.7	76.9	28.0
Waterloo-Avg	80.2	11.9	68.1	15.8	79.3	34.2	56.7	21.0	77.6	27.6	56.7	34.2
Waterloo-Med	81.6	11.7	66.2	13.7	80.7	35.2	58.8	18.3	80.5	22.7	58.8	35.0
Waterloo-Max	100.0	24.1	95.8	38.6	100.0	50.7	95.6	60.6	100.0	86.4	95.5	59.2
Waterloo-Min	47.0	3.6	36.5	5.7	42.2	15.2	9.1	7.4	38.4	11.8	10.1	10.6

Iteration 8:

Voc. Base:	**Low-level artifact**						**Low-level+High-level artifacts**					
Dataset	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	98.3	1.6	68.7	2.4	98.6	7.6	97.8	1.7	69.0	2.4	97.8	7.7
Modis	87.8	18.3	97.2	27.0	75.7	49.9	82.9	50.0	99.8	69.5	71.5	73.3
Waterloo-Avg	78.1	31.9	93.8	40.5	76.5	52.6	57.3	49.6	99.3	57.7	53.7	48.8
Waterloo-Med	81.4	24.0	95.3	32.5	79.6	51.4	58.4	50.6	100.0	57.0	56.0	53.8
Waterloo-Max	97.8	83.3	100.0	83.0	97.7	82.2	95.6	100.0	100.0	100.0	95.5	66.6
Waterloo-Min	43.9	7.1	79.8	7.2	33.9	22.3	11.8	15.8	92.9	24.6	11.8	14.3

Table 3: Effect of different vocabulary bases: Vector Space retrieval with tf-idf, iterations 0 and 8.

is computed as the cosine of the angle between q and d:

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

Okapi [22]. To test the robustness of tf-idf term weighting, we used two alternative term weighting schemes. One of them, proposed within a popular information retrieval system Okapi [22], uses the following formula for computing the weight of a keyword in an element vector [6]:

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

where $tf_i(d)$ is term frequency, df_i is document frequency, n is the total number of elements, dl is the length of current element, and avg_dl is the average length of an element in the collection.

LTU [17]. Our third weighting scheme is linear threshold unit, ltu:

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

Vocabulary bases

The following vocabulary bases have been used:

Low-level artifact only. This is a standard IR approach to building the list of keywords. This list is extracted from the elements of the low-level artifact, playing the role of the document collection.

Both low-level and high-level artifacts. In this approach we take advantage of the fact that we know our "queries," the elements of the high-level artifact, in advance. Extracting keywords from both high- and low-level elements has two effects on the final vectors of keyword weights. First, the vocabulary is enhanced with the keywords found only in the high-level artifact. For the most part, these keywords are useless as they do not match any of the keywords in low-level elements [7]. The second effect is the change in document frequencies of terms: we are guaranteed that any keyword found in both high- and low-level artifacts will have higher document frequency (and thus, lower idf) than in the first vocabulary construction method.

3.1.2 Latent Semantic Indexing (LSI)

Small sizes of datasets, as found in software engineering traceability applications, allow us to use Latent Semantic Indexing (LSI)[8, 10]. LSI is a dimension reduction technique based on Singular Value Decomposition (SVD) of the term-by-document matrix that can be constructed by putting tf-idf vectors of all documents in a single matrix. SVD transforms the original matrix into a product of two orthogonal matrices and a diagonal matrix of eigenvalues. By considering only the top k eigenvalues, we can obtain an approximation of the original matrix by a smaller matrix. Rows of the matrix can be compared to each other using the cosine similarity described above. For example, if L is a document-by-term weight matrix of dimension $A \times B$, its SVD is written as $L = TSD'$, where T is a matrix with orthogonal rows, D' is a matrix with orthogonal columns, and S is a diagonal matrix of eigenvalues of L. We can trim the list of eigenvalues of L from $rank(L)$ to a smaller number k and obtain a decomposition $L_k = TS_kD'$, where S_k is the diagonal matrix of size kxk with k largest eigenvalues of L on

[7] Unless a thesaurus of some sort is used.

the diagonal. Rows of the matrix TS_k^2D' can be compared to each other using the cosine similarity as defined above. Use of the matrix TS_k^2S' instead of the original matrix L reduces the dimensionality of the document vectors from B to k. As with vector space retrieval, we consider two ways to build the reduced-dimensionality matrix:

- *From low-level artifact only.* This is the standard approach to building the reduced matrix. Singular-Valued Decomposition (SVD) is applied to the element-by-keyword matrix consisting only of the low-level artifact vectors.
- *From both low-level and high-level artifacts.* Just as above, we can leverage the fact that we know our queries in advance. Instead of generating query vectors after performing LSI on the low-level element vectors, we can add high-level element vectors to the element-by-keyword matrix on which SVD is performed. Once SVD is complete and the new number of dimensions is selected, we extract the query vectors from the reduced matrix directly and compare them to the reduced vectors for the low-level documents.

In this study, we have run LSI twice for each dataset selected. The first run used a relatively small number of dimensions. The second run used more dimensions, but still significantly lower than the total number of dimensions in the original matrix. The numbers of dimensions are unique for each dataset. LSI was applied only to element vectors produced using standard $tf - idf$ weighting.

3.2 Keyword Extraction by χ^2

In [19], Matsuo and Ishizuka propose a method for determining (extracting) keywords using a single artifact (such as a design document) as a corpus. In a nutshell, the method works as follows. First, they cluster the vocabulary terms and discover the set F of most frequent $f\%$ of all (clustered) keywords. Then, pairwise occurrences in sentences of keywords from F and all keywords are discovered, and a matrix documenting the number of occurrences of each pair is built. Matsuo and Ishizuka assume that important keywords show a more irregular pattern of co-occurrence with different terms than generic words. They use χ^2 to measure the irregularity of such patterns.

We extend this method to information retrieval in a straightforward manner. We consider the ranked list of keywords in the descending order of the computed χ^2 measure. We choose some number $x \in [0, 100]$ and extract the top $x\%$ of the keywords from this list. We then collapse the element vectors of the artifacts by excluding all dimensions corresponding to non-selected keywords. After that, we compute cosine similarity between the reduced vectors. In our experiments, we considered $x = 30, 50$. Two differences exist between our implementation and that of [19]: (i) we consider co-occurrences within the same elements rather than sentences, and (ii) we did not cluster terms prior to building the frequency of co-occurrences matrix.

Clearly, this method cannot improve the % of correct links found (recall) outright as compared to vector space retrieval. However, by making sure that similarity is computed based only on the top tier of keywords, the % of relevant links retrieved (precision) will go up.

3.2.1 Keyword Extraction by *df*

This method is used as a
baseline check against the χ^2-based keyword extraction. Here, we order all keywords in ascending order by their document frequency (*df*). We then exclude all terms with *df* $= 1$ and extract the top $x\%$ of the keywords and reduce the element vectors to contain only the weights of extracted keywords. Cosine similarity between vectors of high- and low-level elements is computed.

In both keyword extraction methods, we use both low- and high-level artifacts to build the initial vocabulary and element vectors.

Feedback Processing. Each of the four methods is enhanced with user feedback processing. An analyst vets the candidate link list and indicates if links are true or false. If the analyst indicates TRUE, the keywords found in that document are given increased value in the query vector (the opposite occurs for FALSE). The IR method is then re-executed using the re-weighted query vectors. For Vector space retrieval as well as for both keyword extraction approaches, we use Standard Rochio Feedback [4] with equal weight assigned to the original query vector, positive feedback, and negative feedback. For LSI, we use the feedback technique from [9].

Each experiment is run for eight iterations. On iteration 0, the chosen IR method is run to build the initial list of candidate links. On each iteration, we automatically examine the top two (not yet examined) elements in the list for each high-level element and provide *correct* feedback.

3.3 Measures

For our study, we elected to track six measures.

Overall precision and overall recall. These measures track the expected outcomes closely: given the output of the method it is important for analysts to know how many of the true links have been captured, and how many of the captured links are false positives.

Average precision and average recall. We compute precision/recall numbers for each high-level element (query) in a tracing task and take the average.

Average expected precision. In our prior work [14], we have examined the effects of filtering the lists of candidate links using a set threshold α. Average expected precision allows for the effects of filtering to be incorporated into a single measurement of a candidate link list.

Given a list Q of high-level requirements, a list D of low-level requirments, and a list $L = \{\langle q, d\rangle, sim(q, d)\}$ of candidate links from D for Q, average expected precision of L is computed as follows. For each $q \in Q$ we consider the the ordered list $D_q = (d_1, \ldots, d_s)$ of candidate links, such that $\langle q, d_i, sim(q, d_i)\rangle \in L$ and $sim(q, d_1) \geq sim(q, d_2) \geq \ldots \geq sim(q, d_s)$. Some of $d_1, \ldots, d_s$ are *relevant* to q (or, in other words form a *true link*). The remaining elements of D_q are *false positives*. A *recall level* is any position $1 \leq i \leq s$ such that d_i is *relevant* (i.e., $\langle q, d_i\rangle$ is a true link).

An *expected precision* of D_q is the mean of precisions computed *at all recall levels of D_q and only at those*. The *average expected precision* of L is computed as the mean of expected precisions $Ep(q)$ for all $q \in Q$.

Consider, for example, candidate link lists for two requirements q_1 and q_2. Each list consists of five elements. In the list D_{q_1} recall levels are $1, 2$ and 3. In the list D_{q_2} recall levels are $2, 4, 5$. Then, for q_1, precisions at recall levels are $\frac{1}{1} = 1, \frac{2}{2} = 1$ and $\frac{3}{3} = 1$, and therefore $E_{q_1} = (1 + 1 + 1)/3 = 1$. At the same time, for q_2, precisions at recall levels are $\frac{1}{2}, \frac{2}{4} = \frac{1}{2}$ and $\frac{3}{5}$. Therefore, $E_{q_2} = \frac{\frac{1}{2}+\frac{1}{2}+\frac{3}{5}}{3} = \frac{8}{15} = 0.533(3)$. Therefore, the average expected precision for q_1, q_2 is $\frac{1+0.5333(3)}{2} = 0.766(6)$, i.e., around

Iteration 0:

Weighting formula:	Okapi						ltu					
Dataset	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM-1	98.1	1.6	49.7	2.5	99.0	7.5	97.8	1.6	51.2	2.4	98.3	7.5
MODIS	75.6	7.9	60.5	10.7	76.9	28.0	75.6	7.9	66.1	10.7	76.9	28.0
Waterloo-Avg	77.2	12.0	67.3	16.3	76.6	34.0	80.0	11.9	68.1	15.8	79.2	34.2
Waterloo-Med	80.7	11.7	68.0	14.4	79.3	35.2	81.6	11.7	67.2	13.7	80.7	35.2
Waterloo-Max	100.0	24.1	97.9	45.3	100.0	50.7	100.0	24.1	95.8	38.6	100.0	50.7
Waterloo-Min	38.2	3.6	36.8	5.7	38.2	15.2	47.0	3.6	35.7	5.7	42.2	15.2

Iteration 8:

Weighting formula:	Okapi						ltu					
Dataset	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM-1	98.6	1.6	69.3	2.3	99.3	7.6	98.3	1.6	70.1	2.3	98.6	7.6
MODIS	87.8	15.9	97.9	33.8	75.7	46.2	87.8	17.1	97.4	29.0	75.7	48.0
Waterloo-Avg	72.8	23.6	90.4	29.6	70.4	44.2	78.1	31.9	93.8	40.5	76.5	52.6
Waterloo-Med	73.5	18.0	93.0	30.8	75.2	45.0	81.4	24.0	95.3	32.5	79.6	51.4
Waterloo-Max	95.6	71.4	100.0	82.8	95.5	78.9	97.8	83.3	100.0	83.0	97.7	82.2
Waterloo-Min	31.8	4.4	34.2	2.6	31.9	14.2	43.9	7.1	79.8	7.2	33.9	22.3

Table 4: Effect of different term weighting schemas for vector space retrieval: Okapi and ltu, low-level artifact as vocabulary base, iterations 0 and 8.

Iteration 0:

Voc. Base:	Low-level artifact							Low-level + high-level artifacts						
Dataset	Dim	Rec	Pr	AEPr	AvPr	AvRec	f_2	Dim	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	100	99.2	0.9	48.7	1.4	99.6	4.3	200	98.9	0.9	51.3	1.4	99.4	4.5
	25	99.7	0.9	40.9	1.3	99.9	4.2	25	98.6	1.0	45.1	1.5	98.9	4.6
Modis	10	95.1	5.4	51.0	7.9	89.6	21.9	10	92.7	6.4	49.9	9.2	97.2	25.0
	25	87.8	5.9	55.8	8.4	88.4	23.1	30	75.6	5.2	50.6	8.0	81.8	20.3
Waterloo-Avg	5	93.8	7.0	48.2	7.9	92.6	25.9	25	89.5	7.3	67.9	8.5	89.9	26.6
	10	93.0	7.2	65.3	8.3	92.3	26.6	40	90.0	7.1	69.0	8.3	89.7	26.0
Waterloo-Med	5	94.6	5.6	48.4	7.7	95.3	22.6	25	89.3	5.9	67.9	7.7	91.7	23.4
	10	93.8	5.8	68.4	8.0	95.3	23.1	40	91.7	5.4	66.3	7.9	93.3	21.6
Waterloo-Max	5	100.0	16.6	60.6	16.5	100.0	48.2	25	100.0	16.1	86.8	17.3	100.0	46.4
	10	100.0	16.6	83.5	17.0	100.0	48.1	40	97.8	15.6	85.4	16.5	97.7	46.4
Waterloo-Min	5	84.6	3.6	40.9	4.4	81.0	15.4	25	79.5	4.1	45.2	4.9	76.2	16.9
	10	82.4	4.0	45.1	4.7	82.4	16.7	40	79.5	3.9	47.5	4.6	76.2	16.1

Iteration 8:

Voc. Base:	Low-level artifact							Low-level + high-level artifacts						
Dataset	Dim	Rec	Pr	AEPr	AvPr	AvRec	f_2	Dim	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	100	99.4	1.0	65.5	1.4	99.7	4.7	200	98.6	1.0	67.8	1.4	99.2	4.9
	25	97.8	0.9	56.5	1.3	97.0	4.5	25	99.2	1.0	61.3	1.5	99.2	4.9
Modis	10	80.5	16.0	73.6	13.9	62.5	44.6	10	85.4	10.8	77.0	17.7	80.2	35.8
	25	80.5	11.2	67.2	16.3	75.3	36.0	30	87.8	12.1	98.0	23.3	75.7	39.0
Waterloo-Avg	5	85.5	10.7	63.8	12.0	83.2	33.9	25	89.6	17.2	91.7	23.1	88.0	44.4
	10	90.1	13.7	83.1	16.6	87.8	39.0	40	89.6	19.0	92.6	26.1	87.4	46.7
Waterloo-Med	5	84.0	9.1	63.4	10.5	83.3	32.5	25	90.0	12.1	92.9	17.2	86.3	38.4
	10	92.2	10.1	83.6	11.1	90.6	35.0	40	90.2	13.8	93.9	18.3	90.7	43.1
Waterloo-Max	5	96.9	24.8	86.4	25.1	96.0	56.9	25	98.5	42.6	100.0	55.6	98.0	72.5
	10	96.9	33.5	98.1	39.1	98.0	68.3	40	97.9	45.2	100.0	59.1	97.7	79.4
Waterloo-Min	5	73.3	5.2	50.4	6.0	66.7	20.7	25	83.9	7.6	74.1	10.7	71.4	28.9
	10	79.4	5.5	68.9	6.3	71.4	22.2	40	78.9	7.8	75.8	11.2	66.7	29.4

Table 5: Latent Semantic Indexing: Effect of vocabulary base, iterations 0 and 8.

76.6%.

f-measure. In [23], we argue that achieving high recall is more important in tracing tasks than achieving highest possible precision. This is due to the fact that low recall means abundance of errors of omission in the candidate link list, while low precision means abundance of errors of comission. To fix an error of comission, the analyst must examine the explicitly provided link. To fix an error of omission, the analyst must notice that there is a missing link and then must search for it manually in the low-level artifact. Clearly, fixing errors of comission is easier for an analyst than fixing errors of omission.

F-measure is defined as a harmonic mean of precision and recall:

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

This formula puts equal preference to both measures. To ensure that high recall is valued more, we skew the f-measure with the parameter $\beta = 2$:

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

4. STUDY

4.1 Datasets

We have considered three different datasets.

MODIS [11, 12]. The dataset consists of 19 requirements and 49 design elements selected from the requirements and design specification (respectively) for the NASA Moderate Resolution Imaging Spectroradiometer (MODIS) space instrument. We consider the forward tracing task, i.e. tracing requirements to design elements. The answer set, containing 41 links, had been manually constructed by the authors and verified by a number of senior analysts with significant tracing experience.

CM-1 [20]. The dataset consists of a complete requirements and a complete design document for a NASA space instrument. We have manually extracted individual requirements (235) and design elements (220) from the documents. We consider the forward tracing task. The answer set, containing 361 links, has been constructed by a team of graduate students and junior analysts, and has been verified by a senior analyst and the authors.

Waterloo [5]. This dataset consists of 22 projects completed by students in the graduate-level Software Engineering course taught by Dan Berry at the Department of Computer Science, University of Waterloo. The students were given the task of designing a voice-over-IP management software system. For the purpose of this study, we have used two documents from each of the 22 projects: the requirements specification and the use cases description. The requirements specification contained the same core functionality of the system for all groups as well as a number of personalized requirements. The use cases have been designed by students to match the functional requirements found in the specification. We chose to use the 22 (out of a total of 36) groups whose project submissions included an RTM in softcopy format for the requirements-to-use cases trace. We have spotchecked the submitted RTMs, but otherwise used them without change. Each requirements document contained anywhere between 17 and 80 functional requirements, with the average of 48 requirements per document (we ignored non-functional requirements in this experiment). Each use cases document also contained between 5 and 30 use cases, with about 17 use cases per document. The answer sets contained between 19 and 143 links, with an average of about 57 links per answer set. It was not uncommon for functional requirements in this dataset to go unsatisfied by the use cases.

4.2 Results

We have implemented all IR methods mentioned in Section 3 as a part of a requirements tracing tool called REquirements TRacing On-target (RETRO) [14]. The results of running each experiment were collected and analyzed against existing answer sets. As mentioned above, we have run each experiment for eight iterations. In this paper, we show only the results from iteration 0 (prior to feedback) and iteration 8. In Tables 3 — 7, we summarize the obtained results[8]. In each table, we show the results obtained for the three datasets. Results for CM-1 and MODIS are shown outright. For the Waterloo dataset, we show four results: the average, the median, the max and the min of each computer measure. Note that the measures are computed independently - e.g., maximum recall and maximum precision can be reached on different cases in the dataset.

Experiments with vector space retrieval. Tables 3 and 4 contain the results obtained when using vector space model. In Table 3, we compare the results of using *tf-idf* term weighting for two different vocabulary bases: low-level artifact and both low-level and high-level artifacts. In Table 4, we compare the results of using *Okapi* and *ltu* term weighting with the low-level artifact as the vocabulary base.

Looking at these tables, we observe the following. All methods exhibit high recall at iteration 0. For the MODIS dataset, recall improves by 8–15% by iteration 8. For the CM-1 dataset, recall remains generally stable, while for the Waterloo dataset recall tended to remain the same for about 40% of the cases, improve for about 50%, and drop slightly for about 10% of the cases. Precision on smaller datasets (MODIS, Waterloo) tends to be between 5 and 15% for individual cases (with the exception of a couple of outliers in the Waterloo dataset) at iteration 0, improving by 8-10% by iteration 8. However, it is just 1.6% for CM-1 at iteration 0 and does not improve significantly. At the same time, we note that all methods and all datasets produce decent-to-good average expected precision at iteration 0 and show significant improvement by iteration 8. For small datasets, average expected precision ends up above 90%, while for CM-1, it improves from around 50% to 68-70%.

Comparing the methods to each other, we note the following. First, the use of both high- and low-level artifacts for vocabulary base could bring significant payoff, but was risky, as overall it tended to decrease the accuracy. For the MODIS dataset, using both artifacts lead to a signficantly better overall result at iteration 8, as evidenced by the f-measure of 0.733 vs. 0.499 when only the low-level artifact was used (recall was less by 5% but precision was higher by 21.7%). However, within the Waterloo dataset, vector space retrieval based on low-level artifact was much more stable and predictable. When we compare the use of different weighting schemes, we notice the following. First, *tf-idf* and *ltu* show almost identical behavior. At the same time, retrieval using *Okapi* scheme

[8]The colmn names in the tables are: "Rec" - overall recall, "Pr" - overall precision, "AEPr" - average expected precision, "AvRec", "AvPr" - average recall and precision, f_2 - f-measure with $\beta = 2$, "Dim" - number of dimensions (for LSI), "Top" - percentage of keywords used (for keyword extraction methods).

Iteration 0:

Method:		chi^2-based extraction						DF-based extraction					
Dataset	Top	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	30	96.7	1.6	44.3	2.4	97.4	7.4	4.7	12.2	25.0	17.0	5.5	5.4
	50	97.5	1.6	46.2	2.4	98.1	7.5	15.2	12.9	37.2	21.5	18.0	14.7
Modis	30	73.2	7.8	51.5	10.6	76.5	27.4	7.3	42.9	100.0	50.0	11.5	8.8
	50	75.6	7.9	72.4	10.7	76.9	28.0	17.1	28.0	80.0	49.3	22.3	18.5
Waterloo-Avg	30	52.4	20.5	76.0	25.8	52.2	32.0	14.4	48.6	96.9	55.6	15.2	16.3
	50	55.7	21.0	77.1	27.4	55.7	33.9	23.3	40.4	94.0	50.9	24.8	24.6
Waterloo-Med	30	48.6	17.7	76.9	21.1	51.1	32.3	13.3	47.6	100.0	54.5	13.3	15.3
	50	57.4	18.3	79.8	22.6	58.6	35.0	20.0	38.9	95.2	51.8	24.8	22.7
Waterloo-Max	30	87.5	60.7	100.0	84.2	90.6	51.8	40.6	91.7	100.0	95.5	41.7	43.9
	50	95.6	60.6	100.0	86.4	95.5	59.2	46.9	61.1	100.0	81.8	46.7	46.9
Waterloo-Min	30	9.1	7.0	37.1	11.8	10.1	10.6	1.9	16.7	80.6	16.7	0.4	2.3
	50	9.1	7.4	38.8	11.8	10.1	10.6	1.9	10.8	79.8	16.1	0.4	2.3

Table 6: Keyword extraction: χ^2-based vs. df-based, tf-idf weighting, iteration 0.

showed some unpredictable behavior on a number of cases in the Waterloo dataset, achieving recall significantly lower than *tf-idf*.

Experiments with LSI. Table 5 shows the results of experiments using LSI. At the time of writing this paper, experiments for 12 of the 22 Waterloo cases were complete, and the results in the table are listed for them. For each experiment, we show the number of dimensions used. For each dataset, we show the results for two different reductions: one using very few dimensions (around 90% reduction rate) and one using a "moderate" (typically between 50 and 70%) number of dimensions. As with vector space retrieval, we ran the experiments for two vocabulary bases: low-level artifact only and both artifacts.

We observe that recall is very high for all experiments: LSI almost always captures more than 90% of the true links. This, however, comes at the cost of precision, that starts at 5-7% for small datasets, improves to 10-15% by iteration 8, and hovers around 1% for CM-1 throughout the feedback process. Average expected precision shows definite improvement for all datasets. For the MODIS dataset, smaller number of dimensions appears to produce marginally better results, while for all other datasets, larger number of dimensions produces somewhat better results for vocabulary based on the low-level artifact.

More importantly, we observe that, unlike vector space retrieval, the quality of LSI is positively affected by the use of both artifacts as vocabulary base. Indeed, while overall precision and recall numbers differ only slightly, we notice significant improvement in average expected precision, meaning that the use of both artifacts supports better separation between true links and false positives in LSI.

Experiments with keyword extraction methods. Tables 6, 7 compare the results of running two keyword extraction methods: χ^2-based and df-based. For both methods, we used both high-level and low-level artifacts to determine the keywords. We used $tf-idf$ as the weighting scheme for the element vectors. For each dataset, we show the results obtained by using top 30% and top 50% of the keywords. Our goal was to test the χ^2-based extraction method. The df-based method was suggested as a baseline, "cheap" keyword extraction procedure (unlike χ^2-based method, it does not require any extra information above what is collected by standard vector space retrieval). As can be seen, these two methods show drastic difference in behavior.

In short, df-based retrieval fails to deliver any stable performance. The recall achieved by this method is unacceptably low for a requirements tracing application. This makes high precision numbers irrelevant — in the context of requirements tracing the method must guarantee successful retrieval of at least a majority of the true links before we start considering its precision. At the same time, χ^2-based retrieval shows good behavior. For CM-1 and MODIS, the "50%" run starts around the same precision and recall as vector space retrieval for the entire vocabulary (thus showing that the bottom half of the list is not very useful for retrieval purposes). For MODIS, it ends with the same recall and higher precision than vector space retrieval with the entire vocabulary. For CM-1, the recall is about 1% smaller, with the same precision. For the Waterloo dataset, χ^2-based retrieval yields lower recall but significantly higher precision, and shows very good average expected precision.

Method comparison - Eliminate Manual Analyst Steps. Figure 1 presents a graphical comparison of the methods for MODIS dataset, iteration 8, top 2 feedback. Reading from the right to the upper left, each point for a method represents a filter value. The tf-idf plus thesaurus bottom-most point of 1.0,0.19 has no filtering applied (all candidate links were used to determine recall and precision). The next point for tf-idf plus thesaurus (0.97, 0.42) was filtered at the 0.05 level (meaning that only candidate links with similarity measures of 0.05 or above were used for the calculation). The filtering increases by 0.05 for each point so that tf-idf plus thesaurus point of 0.5, 1.0 had filtering of 0.5 applied. Our goal is to have points in the upper left corner of the graph: high recall and high precision. The best results shown are for tf-idf plus thesaurus at 0.9, 0.77 and for keyword extraction at .78, .91. And note that the bext non-thesaurus results are for tf-idf at 0.7, 0.82. So chi-square-based keyword extraction outperforms all non-thesaurus methods and performs on par with the thesaurus enhanced tf-idf. This result indicates that manual Step 3, thesaurus building, can be eliminated without undue loss of accuracy.

In addition, it should be noted that the chi-square-based keyword extraction method was on par with the probabilistic results achieved by Cleland-Huang et al. [7]. They achieved recall of 87% with 51% precision as compared to our result of 78% recall with 91% precision (both studies used filtering). This lends evidence to the notion of eliminating manual Step 2, build probabilistic model, without undue loss of accuracy.

Vocabulary Base . Out of all the methods surveyed, vector space retrieval using tf-idf and just low-level artifact as vocabulary

Iteration 8:

Method:	chi^2-based extraction							DF-based extraction					
Dataset	Top	Rec	Pr	AEPr	AvPr	AvRec	f_2	Rec	Pr	AEPr	AvPr	AvRec	f_2
CM1	30	97.2	1.6	60.4	2.4	97.8	7.6	5.3	26.0	50.0	35.5	5.7	6.3
	50	97.8	1.6	62.9	2.4	98.2	7.7	15.2	12.6	48.2	27.2	17.9	14.6
Modis	30	68.3	19.6	98.2	36.3	59.9	45.6	7.3	100.0	100.0	100.0	11.5	9.0
	50	87.8	24.7	98.8	35.6	75.7	58.1	17.1	100.0	100.0	100.0	22.3	20.5
Waterloo-Avg	30	52.2	37.0	97.7	41.1	49.4	40.8	15.6	97.6	100.0	98.6	15.5	18.4
	50	57.2	45.8	98.2	54.7	53.8	46.3	25.3	87.4	100.0	91.4	25.3	28.8
Waterloo-Med	30	55.4	31.2	99.8	32.3	48.8	42.3	15.1	100.0	100.0	100.0	13.3	18.2
	50	61.2	39.6	99.6	50.4	57.4	47.9	22.7	94.2	100.0	94.9	24.8	26.7
Waterloo-Max	30	87.7	87.5	100.0	87.7	88.0	64.7	40.6	100.0	100.0	100.0	41.7	45.8
	50	95.6	86.8	100.0	92.3	95.5	66.0	48.7	100.0	100.0	100.0	46.6	53.7
Waterloo-Min	30	7.8	9.9	86.1	11.0	8.3	8.8	1.8	80.0	100.0	87.5	2.9	2.2
	50	9.8	14.9	86.1	20.7	10.4	11.8	3.0	52.6	100.0	58.3	2.9	3.8

Table 7: Keyword extraction: χ^2-based vs. df-based, tf-idf weighting, iteration 8.

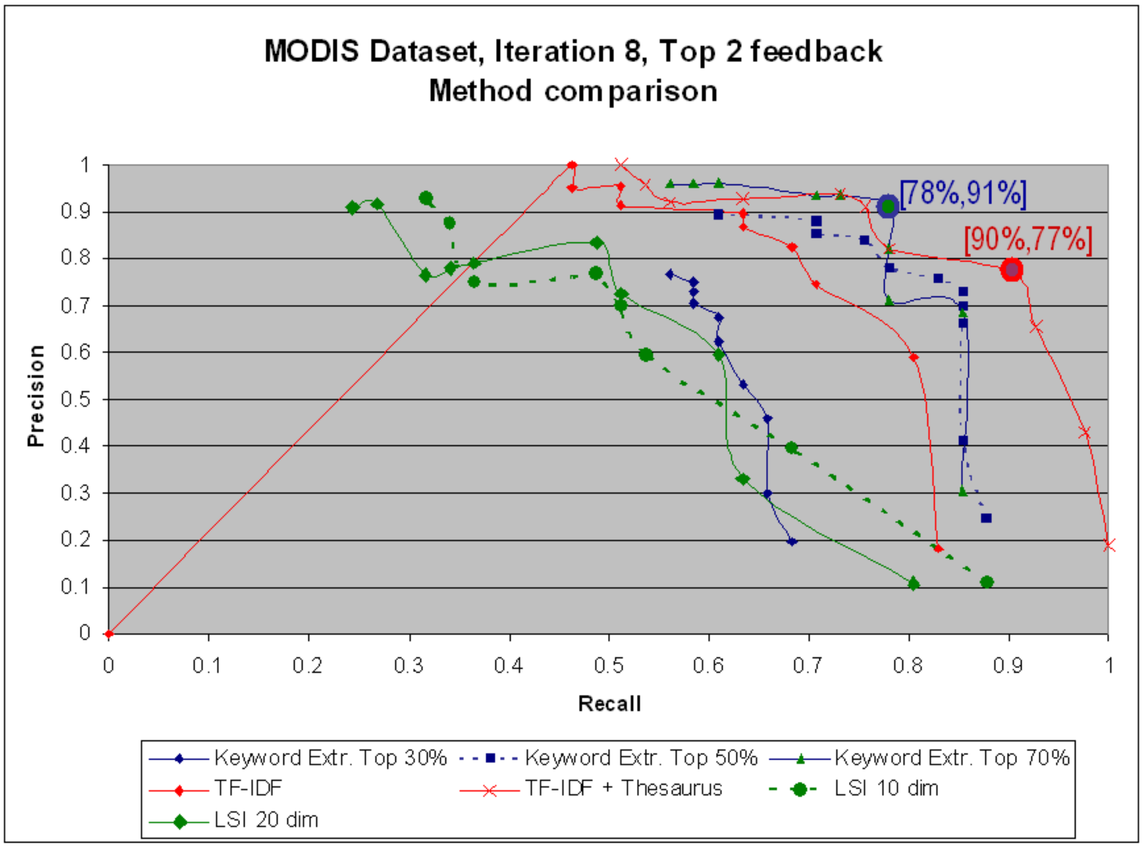

Figure 1: Precision-vs-recall plots for filtered IR methods for the MODIS dataset.

base showed the most consistent results from dataset to dataset. The use of both artifacts for the vocabulary base appears justified for LSI, but not for vector space retrieval. The χ^2-based keyword extraction method appears to be capable of providing high recall-good precision results, but improvement over the use of the entire vocabulary is not guaranteed.

5. CONCLUSIONS AND FUTURE WORK

Traceability links and the ubiquitous RTM are important elements of a software engineering enterprise. Developing RTMs presents challenges for the human analyst, and tool support is lacking. We have applied information retrieval techniques to the problem of tracing and building RTMs and have achieved good success. Specifically, chi-square-based keyword extraction generates traceability links with accuracy on par with methods requiring the building of thesauri or probabilistic models. It appears promising that adoption of this IR method may lead to the elimination of two manual steps of the tracing process, thus reducing analyst workload. We also found that the use of both artifacts for the vocabulary base only produced improved results for LSI.

There are still some datasets for which accurate, consistent results were not seen (for example, CM-1). There is a need to investigate more complex techniques to address this potential scalability issue. The examination of the manual steps of the tracing process identified several areas for potential future study, such as using an experienced analyst for a short period of time to build a probabilistic model. Also, our work has assumed perfect feedback from an analyst. We need to investigate the effect of poor feedback on the methods. We need to examine the impact of the quality of the returned candidate link lists on analyst decision making. This represents subjective study that needs to be undertaken.

6. ACKNOWLEDGMENTS

Our work is funded by NASA under grant NAG5-11732. We thank Stephanie Ferguson and Ken McGill. We thank Mike Chapman and the Metrics Data Program. We thank the MODIS Program. We thank Dan Berry, Davor Svetinovic, and the students of the Software Engineering Seminar at the University of Waterloo. We thank Liming Zhao for his assistance.

7. REFERENCES

[1] Level 1a (l1a) and geolocation processing software requirements specification, September 1997.

[2] MODIS science data processing software requirements specification version 2, November 1997.

[3] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

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

[5] D. Berry. Software requirements and specification course (cs445 / cs645 / ece451 / se463) - University of Waterloo. http://www.student.cs.uwaterloo.ca/ cs445/calendar.shtml.

[6] C. Buckley, A. Singhal, and M. Mitra. New retrieval approaches using SMART. In *the Fourth Text Retrieval Conference (TREC-4)*, 1996.

[7] J. Cleland-Huang, R. Settimi, O. BenKhadra, E. Berezhan, and S. Christina. Goal-centric traceability for managing non-functional requirements. In *International Conference on Software Engineering(ICSE'05)*, 2005.

[8] S. Deerwester, S. Dumais, T. Landauer, G. Furnas, and R. Harshman. Indexing by latent semantic analysis. *Journal of the Society for Information Science*, 41(6):391–407, 1990.

[9] S. Dumais. Enhancing performance in latent semantic indexing retrieval. Technical Report TM-ARH-017527, Bellcore, 1990.

[10] S. Dumais, G. Furnas, T. Landauer, and S. Deerwester. Using latent semantic analysis to improve information retrieval. In *Proceedings of CHI'88: Conference on Human Factors in Computing*, pages 281–285, 1988.

[11] GSFC SBRS. Level 1a and geolocation processing software requirements specification.

[12] GSFC SBRS. MODIS science data processing software requirements specification version 2.

[13] J. H. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. In *International Conference on Requirements Engineering, Monterey, California*, pages 151–161, 2003.

[14] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. In *International Conference on Requirements Engineering (RE'2004)*, 2004.

[15] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard. Helping analysts trace requirements: An objective look. Technical Report TR 392-04, Department of Computer Science, University of Kentucky, January 2004.

[16] H. J. Huffman. Risk reduction through requirements tracing. In *Proc., Software Quality Week*, May 1990.

[17] K. S. Jones and P. Willet. *Reading in Information Retrieval*. Morgan Kaufmann Publishers, 1997.

[18] A. Marcus and J. Maletic. Recovering documentation-to-source code traceability links using latent semantic indexing. In *Proceedings of the Twenty-Fifth International Conference on Software Engineering, ICSE 2003*, 2003.

[19] Y. Matsuo and M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. *International Journal on Artificial Intelligence Tools*, 13(1):157–169, 2004.

[20] MDP Website. CM-1 project. http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1.

[21] M. Porter. An algorithm for suffix stripping. *PROGRAM*, 14(3):130–137, 1980.

[22] S. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Proc., Eighth Text Retrieval Conference (TREC-8)*, 2000.

[23] S. Sundaram, J. H. Hayes, and A. Dekhtyar. Baselines in requirements tracing. In *IEEE International Workshop on Predictive Models for Software Engineering (PROMISE) 2005*, 2005.

[24] M. T. and H. F. Requirements analysis using supertrace pc. In *Proceedings of the American Society of Mechanical Engineers (ASME) for the Computers in Engineering Symposium at the Energy & Environmental Expo*, 1995.