Improving After-the-Fact Tracing and Mapping:

Supporting Software Quality Predictions

Jane Huffman Hayes, Alex Dekhtyar, and Senthil Sundaram, *University of Kentucky*

The requirements traceability matrix has proven useful in assisting with the prediction of software product quality before code is written. Automated information retrieval techniques could lead to its widespread adoption.

The software engineering industry undertakes many activities that require generating and using mappings. Companies develop knowledge bases to capture corporate expertise and possibly proprietary information. Software developers build traceability matrices to demonstrate that their designs satisfy the requirements. Proposal managers map customers' statements of work to individual sections of companies' proposals to prove compliance. Systems engineers authoring interface specifications record design rationales as they make relevant decisions. In some cases, you can generate mappings or traceability information as a process unfolds—for example, developers working on design. But many cases require mapping preexisting information—for example, when you must build a knowledge base, or when a verification and validation (V&V) agent or independent verification and validation (IV&V) agent must determine a trace. We refer to this latter practice as *after-the-fact tracing*.

The *requirements traceability matrix*, mapping elements of a high-level artifact such as requirements to elements of a low-level artifact such as design, forms the basis of good V&V. Software engineers can use the RTM to predict a software system's quality as it's being built, well before any code is written. However, despite the RTM's advantages, its development process requires that analysts manually discover and vet links between artifact levels. Our work focuses on the challenge of automatically identifying potential or candidate links. We developed an approach to tracing and mapping that aims to use fully automated information retrieval techniques, and we implemented our approach in a tool, called RETRO (Requirements Tracing on Target). Backed by empirical results, we discuss the advantages our approach has over others in the industry.

The requirements traceability matrix

Software engineers can use the RTM for such tasks as:

- performing traceability analysis (Do all low-level elements have parents?),

0740-7459/05/$20.00 © 2005 IEEE

- performing completeness analysis (Have all high-level requirements been fully satisfied?), and
- assessing test coverage (Do test cases exist for each requirement?).

To comprehend the importance of the RTM, you need only look at just one headline-grabbing incident. When NASA lost its Mars Climate Orbiter, US taxpayers lost a $125 million spacecraft, and the space program lost the single communication link that was to exist between the Mars Polar Lander and Earth.[1] The loss was due to a software issue: navigation information was specified in English rather than metric units in a module. The earliest opportunity to discover this anomaly was during requirements tracing and RTM development.

Benefits

As Barry Boehm's work[2]—recently reconfirmed by Stephen Schach and his colleagues[3]—indicates, issues identified early in software system development are much less costly and time consuming to repair than if left undetected until later. Using the RTM to perform analyses such as traceability analysis and completeness analysis provides quantitative results that can be used to predict software quality. For example, traceability analysis measures the percentage of elements without parents. Completeness analysis measures the percentage of parents not fully satisfied by their children elements. If the percentage of unsatisfied parent elements is high, we can predict with high probability that a poor-quality software system will be developed (incomplete system). If the percentage of elements without parents is high, then we can predict with high probability that a poor quality software system or unsecure system will be developed or cost escalations and schedule delays because of unintended functions in the system will occur.

Obstacles

Accepting the RTM's importance often isn't enough to encourage its development, however, due to its tedious development process. The analyst performing this unenviable task must examine each high-level element one by one and find matches among the low-level elements. RTM development doesn't differ much from other processes involving information retrieval, such as Internet searching. In tracing from a high-level requirements document to a design document, you can view the high-level requirements as queries (such as the ones we write in a search field) into the design elements that play the same document collection role as Web-page collection on the Internet.

The key difference between RTM construction and searching the Web becomes apparent when you consider the role of the human in each process. The state of the art in RTM generation has the human analyst

- manually assign keywords to all elements of all artifact levels or build detailed keyword thesauri,
- manually or semiautomatically perform all the searches for low-level elements that map to high-level elements, and
- render a decision considering each discovered candidate link.

In contrast, when searching the Web, the user must only specify her information need and examine the results for relevant Web pages. Specially designed software performs the actual search. At the core of this software lie different information-retrieval techniques for determining the relevance between the documents in the collection (Web pages) and user queries. Long before the user's search, search-engine software automatically indexes all the documents.

The difference between the amount of manual labor in requirements tracing as compared with Web search is stark. Is it any wonder then that tracing is tedious, error prone, and boring? Is it any wonder that V&V analysts want an improved process?

The solution: automation

In our opinion, the best way to improve the RTM generation process is to model it on how search engines operate. The analyst's role would change from a human search engine to a verifier who checks the automatically generated candidate RTMs. Software would assume the responsibility of indexing the high-level and low-level elements and determining (at the outset) pairs of similar elements.

To achieve this goal, the system must be able to automatically identify potential or candidate links. The new approach to this challenge is to use fully-automated information retrieval techniques. Researchers have begun

Accepting the importance of the requirements traceability matrix often isn't enough to encourage its development.

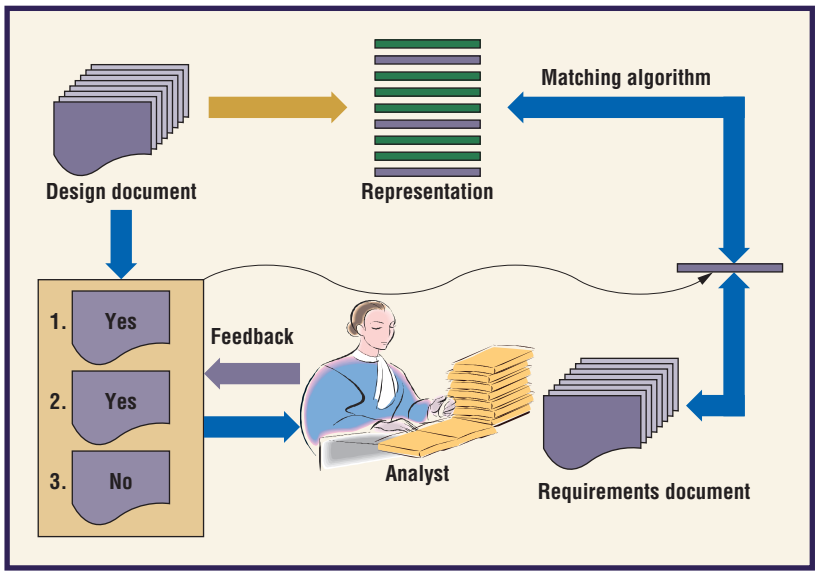

Figure 1. Requirements tracing scenario as an information retrieval problem.

investigating the application of such techniques to this problem with encouraging results.[4–7] For example, we have observed that manual tracing and keyword matching require more effort, yet provide less accuracy.[6]

To illustrate our approach to candidate-link generation for after-the-fact tracing, we examine the following scenario: a V&V analyst has been asked to determine if a collection of design specifications fully satisfies a requirement specification. This task is often called *forward tracing* (see figure 1). We frame forward tracing as an IR problem. We'll step requirement-by-requirement through the specification (shown at the lower right of the figure), treating each requirement as a query into the collection of design elements ("the document collection" in information retrieval terms).

Information retrieval methods for requirements tracing

Information retrieval works in two stages. First, the IR system analyzes and indexes incoming document collection. As a result of this process, a representation of each document is constructed and archived. Second, the system analyzes and represents incoming queries, as with the documents, and uses a matching or ranking algorithm to determine which document representations are similar to the query representation.

In the next sections, we describe the results of using two different IR approaches: *vector space model with term-frequency-inverse document frequency (tf-idf) term weighting*[8] and *latent semantic indexing*.[9] To enhance retrieval, we used a simple thesaurus. We also simulated the analyst's work by incorporating user feedback in the retrieval process. Finally, we considered the effects of producing filtered candidate-link lists. We have used a number of typical IR measures to evaluate the performance of these methods. We'll discuss each of these.

However, it is appropriate to first mention some preprocessing that occurs regardless of the IR technique used: stopword removal and stemming. We use a standard list of stop words such as "the," "and," and "of," and remove these terms from consideration. We use Porter's stemming algorithm to remove suffixes of words to extract their stems—for example, "documenting," "documents," and "documentation" all become the stem "document."[10]

Vector space retrieval

Vector space retrieval methods represent each document and each query as a vector of keyword (term) weights. IR researchers have proposed different term weighting schemes. We use tf-idf, the standard term-weighting computation method. To compute term weights, this method uses *term frequency* (tf), the (normalized) number of times that each term (or word) occurs in a given query or document, and *inverse document frequency* (idf), which measures how rarely the term is found in the entire collection. Term frequency helps capture keywords that occur often in a given document. Inverse document frequency captures the rare terms with high distinguishing ability. Once the system builds the document and query vector, it computes the similarity between them as the cosine of the angle between them.[8]

Latent semantic indexing

Latent semantic indexing is a dimensionality reduction technique. Vector space retrieval methods construct a document-by-keyword matrix, which can be viewed as a mapping between the keywords and documents. The larger the vocabulary—the list of unique words or terms found in the queries and the document collection—the larger the matrix becomes. Also, the matrix can become sparse—most keywords will not occur in most queries and documents. LSI employs singular-valued decomposition of the document-by-term matrix to represent it as a product of two ortho-

normal matrices and a diagonal matrix of its eigenvalues. By reducing the number of eigenvalues considered, we can construct reduced-dimensionality approximations of the original matrix. These reduced-dimensionality matrices might encompass some underlying (latent) concepts or domains of interest within the document collection. The similarity comparison between document vectors and query vectors is then performed on the reduced space.

Thesaurus

Artifacts being traced to each other often are written by different individuals or organizations, or use different "lingo." For example, one artifact might use the word "error" whereas another artifact uses the word "failure." Standard IR techniques will not recognize these two words as being relevant to each other. One way to address such situations is to build a simple thesaurus. Our thesaurus is a set of triples of the form (v, a, w) where v and w are terms and a indicates the degree of relevance between the two. Many standard thesauri are free or relatively inexpensive. We found that it's possible to build a simple thesaurus in a short period, often using preexisting artifacts as a basis (such as an appendix of acronyms or a data dictionary). For example, we built the thesaurus for the Moderate Resolution Imaging Spectroradiometer (MODIS) dataset in just 20 minutes. The thesaurus is then used in conjunction with either tf-idf or LSI as follows: if the system finds that terms in the vector of the query or document are from the thesaurus, it adds the product of their weights and a to the similarity measure.

Feedback

Constructing an RTM using automated tools is an interactive process. The software provides the analyst its best guess, and the analyst examines the lists of candidate links and updates them. But what happens if the analyst communicates her decisions back to the software? Research in IR shows that the use of such analyst feedback (think clicking on "More links like this" in a Web search engine) provides improved results. Thus, we introduce the notion of analyst feedback to improve tracing. Here, tracing tool shows the analyst a candidate link list and asks the analyst to vet the links. Suppose that an analyst is tracing from a requirements specification (query) to a design document (document collection). If the analyst indicates that a candidate link is true, the keywords found in that document increase in value in the query vector. If the analyst indicates that the link is false, the keywords found in that document decrease in value in the query vector. After the analyst completes a round of feedback, the tracing tool reruns the appropriate IR method using the reweighted query vectors. We used several feedback approaches, and Standard Rochio feedback performed best.[8]

Filtering

Just as in Web search, a long list of irrelevant items that have been retrieved can easily overwhelm an analyst or user. One way to alleviate this is to use filtering. The idea is to only display items that have relevance above a certain level. For example, filtering at 0.1 would display to the analyst only candidate links with relevance of 0.1 or higher. We applied filtering at various levels: 0.05, 0.1, 0.25, 0.3, and so on.

Measures of success

We applied many common IR measures to evaluate how well the methods perform. We address recall, precision, and selectivity here.

Recall. Recall is a coverage measure. Given the theoretical "true trace" or "answerset," it measures the percentage of true links retrieved. Recall is the number of correct retrieved links (C) divided by C plus the number of correct missed links (M). We want recall to be as high as possible for tracing tasks. But note that you could achieve 100-percent recall by merely retrieving all elements for each query. The result, though, would be very low precision.

Precision. Precision is a signal-to-noise ratio. It examines how much "junk" an analyst is made to examine. It's measured as the number of correct retrieved links (C) divided by C plus the number of retrieved false positives (F). We want this measure to be as high as possible.

Selectivity. The tracing activity could theoretically require an analyst to manually perform $M \times N$ comparisons, where M is the number of high-level elements and N is the number of low-level elements. A tracing method that solicits feedback from the analyst should result in the problem space (that starts as $M \times N$) becoming smaller with each iteration. We'd like the problem space to become small as quickly

It's possible to build a simple thesaurus in a short period, often using preexisting artifacts as a basis.

Related Links

Holagent Corporation's RDD-100: www.holagent.com/products/product1.html
Metrics Data Program: http://mdp.ivv.nasa.gov
Metrics Data Program's CM-1 Project: http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1

Table 1

Results of information retrieval methods on CM-1 and Modis datasets (no feedback, no filtering)*

	Modis			CM-1		
Method	Precision (%)	Recall (%)	Selectivity (%)	Precision (%)	Recall (%)	Selectivity (%)
tf-idf	7.9	75.6	41.9	1.5	97.7	42.8
tf-idf + thesaurus	**10.1**	**100.0**	43.1	1.5	97.7	42.8
LSI (10/100)	6.3	92.6	64.1	0.9	98.6	71.5
LSI + thesaurus (10/100)	6.5	95.1	63.7	0.9	98.6	71.5
LSI (19/200)	4.2	63.4	65.2	0.9	98.8	73.9
LSI + thesaurus (29/200)	5.4	80.4	65.8	0.9	98.8	73.9

*Bold denotes best results.

as possible. Selectivity helps to measure this. It is measured as the number of correct candidate links (C) plus the number of false positives (F) divided by the product of M and N.

Retro: Requirements Tracing on Target

We implemented our approach to tracing and mapping in our Retro tool. The tracing process using Retro is a multistep process. First, the analyst selects the documents for tracing through the GUI. The GUI is written in Java and the input documents are flat files. Next, the Build module, written in C++, builds the corpus (the matrix of the occurrence of terms) for the selected artifacts. Each element of each artifact is output in its vector representation. The vectors are represented as XML and passed to the IR toolkit, written in C++. After the tool applies the selected IR method (tf-idf or LSI), the results (candidate-link lists with relevance values) are passed in XML to the filtering module (written in C++). The filtering module applies a filtering technique and sends the results to the GUI. The analyst then assesses the candidate-link lists and makes choices ("yes, this is a link," "no, this is not a link"). The tool sends the choices to the feedback module that reweights the vectors for the high-level artifact. Then, the process starts again.

Results to date

We applied our toolkit to two datasets. The first, Modis, is a NASA science instrument.[11] It comprises 19 high-level elements extracted from a larger top-level requirement specification and 49 low-level elements extracted from a software requirement specification. The two levels have 41 correct links between them. The information on the correct links is stored in the answer set (the "theoretical true trace"). The second dataset is called CM-1 and is also from a NASA science instrument. The data has been provided by the Metrics Data Program (see the Related Links sidebar for this and other useful URLs). MDP sanitized the data to hide the project's identity. There are 235 high-level elements from a requirement specification, 220 low-level elements from a design specification, and 361 correct links.

In our experiments with these datasets, we used the following user-feedback strategy. On each iteration and for each high-level requirement, we examined two previously unseen candidate links with highest relevance and specified, according to our answer set, whether or not they were correct. In addition, for each iteration, we considered the accuracy of both the unfiltered list of candidate links and the filtered lists. We used filter threshold values of 0.05, 0.1, 0.15 and 0.2. We ran experiments with and without a thesaurus. For LSI, we ran experiments at differing matrix dimensionality.

Perhaps the easiest way to depict the methods' accuracy is through the two primary measures of recall and precision. As mentioned earlier, we are most interested in high recall, but we also want acceptable precision. In addition, we want selectivity to be as low as possible. In table 1, we show the recall, precision, and selectivity values obtained for experiments run with Retro on Modis and CM-1 datasets with no filtering or feedback. Note that the best result we have achieved is 10.1 percent precision, 100 percent recall, and 43.1 percent selectivity for Modis using tf-idf plus thesaurus. Though recall is excellent, precision and selectivity are unacceptable.

Table 2 shows that filtering and feedback make a tremendous difference. Even with filtering at a very low level of 0.05 (show all

Table 2

Effects of relevance feedback and filtering on the results of information retrieval methods*

Method	Filter / Iteration	0.05 Recall (%)	0.05 Precision (%)	0.1 Recall (%)	0.1 Precision (%)	0.15 Recall (%)	0.15 Precision (%)	0.2 Recall (%)	0.2 Precision (%)
MODIS, tf-idf	0	48.8	7.8	29.3	11.8	24.4	17.2	19.5	21.6
	1	48.8	8.1	29.3	12.6	24.4	20.0	22.0	36.0
	2	48.8	8.9	31.7	17.6	24.4	31.3	24.4	47.6
	3	53.7	11.1	31.7	21.7	31.7	38.2	31.7	61.9
	4	65.9	14.9	46.3	33.9	36.6	51.7	31.7	65.0
	5	68.3	19.7	53.7	51.2	46.3	70.4	41.5	77.3
	6	**70.7**	**33.7**	**65.9**	**64.3**	48.8	74.1	48.8	80.0
	7	**75.6**	**50.0**	**68.3**	**70.0**	63.4	78.8	51.2	80.8
	8	**80.5**	**58.9**	**70.7**	**74.4**	**68.3**	**82.4**	**63.4**	**86.7**
MODIS, tf-idf + thesaurus	0	78.0	12.1	65.9	22.3	46.3	25.7	39.0	33.3
	1	78.0	12.1	61.0	21.6	51.2	33.3	41.5	44.7
	2	78.0	12.7	61.0	25.0	51.2	42.9	46.3	61.3
	3	78.0	14.1	63.4	29.5	56.1	50.0	53.7	68.8
	4	**90.2**	**18.9**	**75.6**	**37.3**	56.1	52.3	56.1	69.7
	5	**95.1**	**22.4**	**75.6**	**46.3**	58.5	57.1	56.1	67.6
	6	**97.6**	**30.3**	**78.0**	**52.5**	**65.9**	**69.2**	58.5	70.6
	7	**97.6**	**39.2**	**92.7**	**60.3**	**78.0**	**74.4**	**75.6**	**77.5**
	8	**97.6**	**43.0**	**92.7**	**65.5**	**90.2**	**77.1**	**78.0**	**82.1**
CM-1, tf-idf	0	92.2	4.4	***76.5***	***10.8***	53.7	19.1	32.7	27.1
	1	91.7	4.3	***77.0***	***10.9***	55.4	19.8	38.0	31.6
	2	91.4	4.3	***76.2***	***10.8***	59.0	20.8	42.9	34.4
	3	91.7	4.4	***77.6***	***10.9***	63.2	22.0	45.4	34.8
	4	92.0	4.4	***78.9***	***11.1***	66.5	23.1	48.8	35.6
	5	92.0	4.4	***81.4***	***11.5***	67.6	23.6	52.6	37.6
	6	92.2	4.4	***82.8***	***11.7***	**70.1**	**24.3**	55.4	39.1
	7	92.2	4.4	***84.2***	***11.9***	**70.9**	**24.5**	57.6	39.6
	8	92.2	4.5	***84.8***	***12.0***	**74.0**	**24.8**	**61.2**	**40.9**

*Bold denotes best results.

links that have relevance above 0.05), we can achieve recall of 80.5 percent and precision of 70.7 percent for tf-idf on MODIS. Note that the results are achieved on the 8th iteration of feedback. Without feedback (but with filtering of 0.05), our best recall result occurs for CM-1 using tf-idf, with a recall of 92.2 percent. Precision, however, is only 4.4 percent. Our best precision result is 29.3 percent for MODIS with tf-idf, but recall is only 48.8 percent. When we take advantage of both filtering and feedback techniques, we achieve excellent results. For example, we achieve 90.2 percent recall and 77.1 percent precision for the 0.15 filter for MODIS with tf-idf plus thesaurus. Similarly, we obtain 86.7 percent recall and 82.4 percent precision for MODIS for tf-idf.

From table 2, we can see that adding a thesaurus for MODIS increased accuracy somewhat (recall went up to 97.6 percent on the 0.05 filter with thesaurus). But a thesaurus doesn't always improve results. In table 1, the results for CM-1 were the same regardless of whether a thesaurus was used.

In our work, it has become obvious that we needed to assign at least rough levels of "goodness" to determine if methods are appropriate for tracing. We defined acceptable, good, and excellent levels in table 3.

Table 3

"Goodness" levels

Measure	Acceptable	Good	Excellent
Recall	> 60%	>70%	> 80%
Precision	> 20%	>30%	> 50%

*Levels below these are unacceptable.

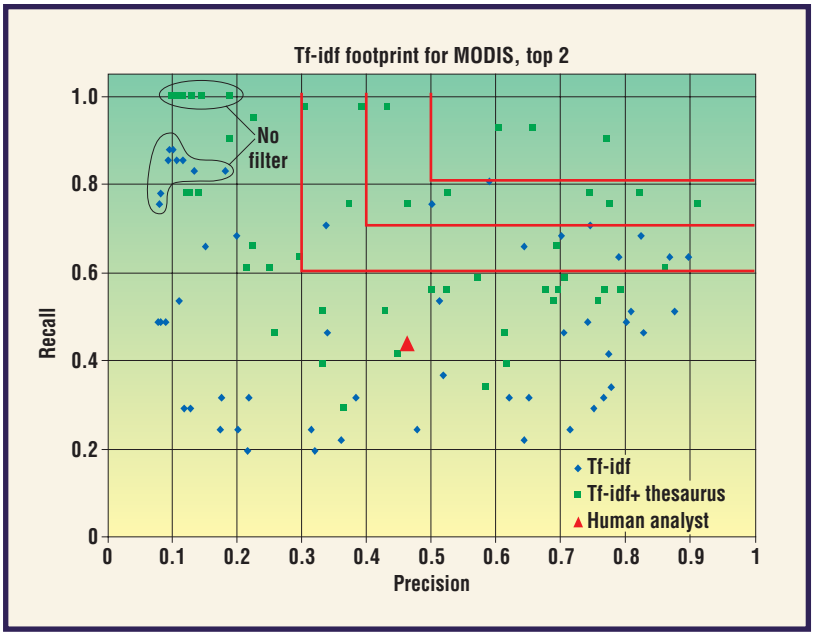

Figure 2. Footprint graph for Modis using term-frequency-inverse document frequency (tf-idf).

We developed a footprint graph to depict these goodness levels. Figure 2 shows the acceptable, good, and excellent levels of recall and precision in the upper right quadrants of the graph. Next, we took all the data points we have for various runs (points from the 0th iteration of tf-idf, points from the 1st iteration, and so on) for a dataset and showed them on a scatter plot. The concern isn't that all points must be acceptable, but that some acceptable (or better) points must exist.

Figure 2 depicts the footprint graph for the Modis dataset for the tf-idf method. Some of the data points fall in the acceptable, good, and excellent regions. For reference, we added a result we obtained in previous work of a human analyst working with this dataset.[6] Figure 3 shows the footprint graph for CM-1, for both tf-idf and LSI. This graph makes clear that we're not achieving the same success for the larger dataset as we are for the smaller one. We plan to address this by applying more advanced methods from IR and tailoring these methods to the tracing problem's characteristics. We can also see that filtering has a tremendous impact on the recall and precision. The upper-left group of points represents the highest recall but the lowest precision—this is without filtering. The next collection of points (below and to the right) is for filtering level 0.05. Recall decreases, but precision increases. As you continue to move down the graph and to the right, you can see that precision gradually increases to 0.67, but recall also decreases to 0.39. This effect is highly visible in figure 3 because the traces for runs with different filter values are highly separable. In figure 2, we highlighted the unfiltered points for the readers' benefit.

We show only a few exemplary results in this article. Many such figures and results are available in previous works.[6,12] We can state that, in general, high iterations lead to higher recall and precision values. Higher filter values increase precision, but at the price of recall.

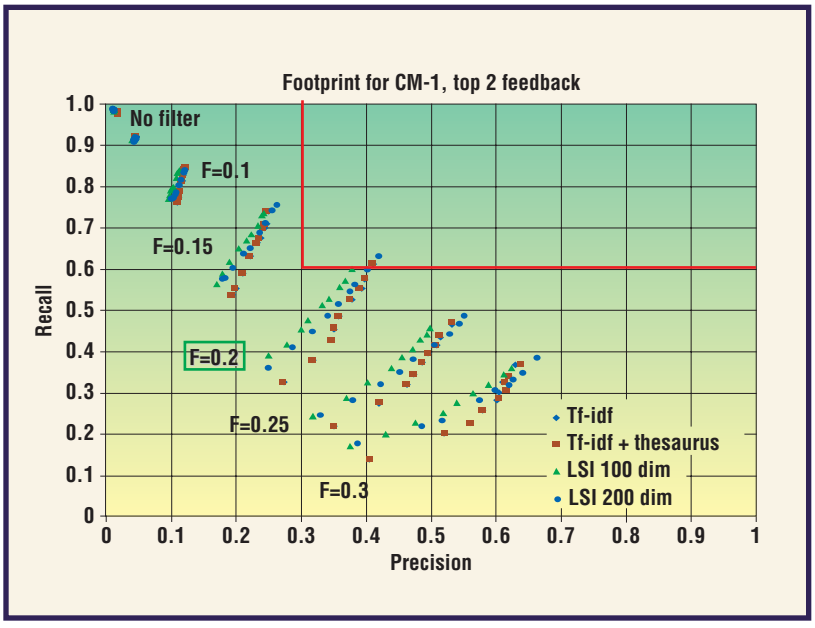

Figure 3. Footprint graph for CM-1 using tf-idf and LSI.

Our approach to prediction provides a significant advantage over other methods: We can provide predictive information before any code has been written. Other techniques use data from previous releases to build predictive models, an approach useful only for future developments. Our approach helps analysts make predictions early enough to allow improvements on current projects.

Our Retro tool has outperformed analysts, even those using state-of-the-art tools.[6] Specifically, in the pilot experiment[6] using the Modis dataset, the analyst working with the results from a proprietary toolkit achieved 46.15 percent overall precision and 43.9 percent overall recall. As can be seen from figure 2 and table 2, our results are much higher. Our hope is that the ease of RTM development with Retro will encourage wider-spread RTM develop-

ment, even if after the fact, to promote software quality predictions earlier in the lifecycle.

We already successfully integrated our methods into a requirements tracing tool used by an IV&V organization on NASA and US Navy mission-critical software systems. We've also made the RETRO tool available to several other IV&V organizations and one European systems engineering organization. You can also use the toolkit for purposes more general than tracing. Contact us if you're interested in learning more about RETRO or if you would like to volunteer to assist us with ongoing studies of tracing.

Acknowledgments

NASA funds our work under grant NAG5-11732. We thank Stephanie Ferguson and Ken McGill. We also thank the Metrics Data Program, Mike Chapman, and the MODIS program. We also thank our students who worked on RETRO: James Osborne, Sarah Howard, Ganapathy Chidambaram, and Sravanthi Vadlamudi.

References

1. M. O'Brien and Associated Press, "NASA: Human Error Caused Loss of Mars Orbiter," *CNN.com*, 10 Nov. 1999, www4.cnn.com/TECH/space/9911/10/orbiter.03.
2. B.W. Boehm, *Software Engineering Economics*. Prentice Hall, NY, 1981.
3. S. Schach et al., "Determining the Distribution of Maintenance Categories: Survey versus Empirical Study," *Kluwer's Empirical Software Eng.*, 2003; vol. 8, no. 4., pp. 351–365.
4. G. Antoniol et al., "Recovering Traceability Links between Code and Documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, 2002, pp. 970–983.
5. A. Marcus and J. Maletic, "Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing," *Proc. 25th Int'l Conf. Software Eng.* (ICSE 2003), IEEE CS Press, 2003, pp. 125–135.
6. J.H. Hayes et al., "Improving Requirements Tracing via Information Retrieval," *Proc. Int'l Conf. Requirements Engineering* (RE 2003), IEEE CS Press, 2003, pp. 138–147.
7. J.H. Hayes et al., "Helping Analysts Trace Requirements: An Objective Look (2004)," *Proc. 12th Int'l Requirements Eng. Conf.* (RE 2004), IEEE CS Press, 2004, pp. 249–261.
8. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval,* Addison-Wesley, 1999.
9. J. Haritsa, M. Carey, and M. Livny, "On Being Optimistic about Real-Time Constraints," *Proc. 9th ACM SIGACT-SIGMOD-SIGART Symp. Principals of Database Systems*, ACM Press, 1990, pp. 331–340.
10. M.F. Porter, "An Algorithm for Suffix Stripping," *Program*, vol. 14, no. 3, 1980, pp. 130–137.
11. *MODIS Science Data Processing Software Requirements Specification Version 2*, SDST-089, GSFC SBRS, Nov. 10, 1997, NASA.
12. J. Hayes et al., "Helping Analysts Trace Requirements: An Objective Look," tech. report TR 392-04, Department of Computer Science, Univ. of Kentucky, Jan. 2004.

About the Authors

Jane Huffman Hayes is an assistant professor in the Department of Computer Science at the University of Kentucky. Her research interests include requirements, software V&V, traceability, maintainability, and reliability. She received her PhD in information technology from George Mason University. She's a member of the IEEE Computer Society. Contact her at Univ. of Kentucky, 301 Rose St., Lexington 40506-0495; hayes@cs.uky.edu.

Alex Dekhtyar is an assistant professor in the Department of Computer Science at the University of Kentucky. His interests include traceability, management and reasoning with uncertain information, digital libraries, computing in humanities, and management of XML data. He received his PhD in computer science from the University of Maryland at College Park. He's a member of the ACM; the ACM's Special Interest Groups on Management Of Data, Information Retrieval, and Data Mining and Knowledge Discovery; the American Association for AI; and the Association for Logic Programming. Contact him at Univ. of Kentucky, 763G Anderson Hall, Lexington, KY 40506-0046; dekhtyar@cs.uky.edu.

Senthil Karthikeyan Sundaram is a PhD candidate in the Department of Computer Science at the University of Kentucky. His research interests include traceability, requirements engineering, software design, information retrieval, and data mining. He received his BE in computer science and engineering from Madras University. He's a member of the IEEE Computer Society. Contact him at 444 S Ashland Ave, Apt. B1, Lexington, KY 40502; skart2@uky.edu.