Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools?

Alex Dekhtyar
Jane Huffman Hayes
Jody Larsen
University of Kentucky
{dekhtyar,hayes}@cs.uky.edu, jody@dreamfrog.com

Abstract

Several recent studies employed traditional information retrieval (IR) methods to assist in the mapping of elements of software engineering artifacts to each other. This activity is referred to as candidate link generation because the final say in determining the final mapping belongs to the human analyst. Feedback techniques that utilize information from the analyst (on whether the candidate links are correct or not) have been shown to improve the quality of the mappings. Yet the analyst is making an investment of time in providing the feedback. This leads to the question of whether or not guidance can be provided to the analyst on how to best utilize that time. This paper simulates a number of approaches an analyst might take to evaluating the same candidate link list, and discovers that more structured and organized approaches appear to save time/effort of the analyst.

1. Introduction

There are many tasks or activities that are undertaken as a routine part of the software development lifecycle and require a human software engineer or analyst to make decisions or judgments based on the output of an automated tool. For example, a software architect may examine the outputs of a cost benefit analysis tool to decide on a particular architecture; a project manager may examine the output of a risk assessment tool to determine what risks to mitigate; a software maintainer may examine the code modules retrieved by a tool as relevant to a given bug report, etc. When the process of making judgments or decisions is repetitive and/or mundane, it is important to make the best possible use of the human's time. Predictor models have long been used to predict how things will behave, such as project cost, complexity, risk, change, etc. Can predictor models be used to predict how human decisions or human approaches to a mundane task of providing judgments may impact the quality of the final results? This idea is examined here, using the specific case of an analyst interacting with a tool that is helping to build a mapping between software engineering artifacts.

Much of the information in a software artifact repository is expressed as free form text. Information retrieval (IR) techniques have demonstrated usefulness in building models of the relationships between such textual artifacts, such as requirements traceability matrices (RTMs) or documentation dependencies, i.e., what portions of a user's manual need to be changed when certain source code methods are modified, for example [7,8,12,1,14,4].[1] In a nutshell, these techniques examine the elements of a high level artifact (such as a concept specification) and return the elements deemed relevant from a lower level artifact (such as source code). When the notion of feedback (similar to the "more URLs like this" feature in Google) is added to the IR methods, the result is better candidate RTMs. That is to say that a higher number of true links are found while the number of false positives is lowered.

While the overarching goal is to generate the best possible RTM, it is clearly desirable to require the least amount of analyst effort possible to achieve this. In fact, even if an analyst is willing to expend a maximal amount of effort, *E*, it is still important to make the best possible use of *E*.

[1] RTMs are the backbone of many important activities performed by Verification and Validation (V&V) and Independent Verification and Validation (IV&V) analysts, including ensuring that requirements are satisfied by the design, ensuring that the requirements are implemented in the code, etc.

In this paper, we report on an initial study that compares the influence of different *simulated* analyst behaviors when presented with the same list of candidate links retrieved by an automated method. In a number of previous studies [9,10], our research group raised the issue of the importance of studying analyst behavior. We separated the study of traceability into two categories: the study of methods and the study of analysts. The former category includes empirical studies concentrating on the analysis of the results provided by the automated traceability methods *without interaction with an actual analyst.* The latter category encompasses empirical studies that involve an actual analyst performing traceability tasks. We have also noted that we believe that the study of the analyst is often a *garbage-in—garbage-out endeavor*, i.e., if the analyst sees "bad" results, it is very hard for the analyst to improve upon them.

The study reported here belongs to the realm of the *study of methods*, as in this study we simulate perfect analyst behavior, i.e., we assume that analysts always correctly determine the status of candidate links provided to them. This work is a necessary precursor to the *study of the analyst*, which will concentrate on the same questions.

The paper is organized as follows. Section 2 briefly describes the mathematics behind the automated tracing method used in this study. Section 3 describes the simulation method and the proposed analyst scenarios. Section 4 presents the results of our experiment as well as the dataset used for the validation (in the PROMISE repository). Section 5 presents related work in mapping and traceability, with emphasis on prior studies of the analyst. We provide final analysis of the results in Section 6.

.

2. Automated Tracing and Feedback Processing

Our group reported on a number of Information Retrieval techniques used to generate candidate link lists [12, 13]. For this study, we selected one method, vector space retrieval using tf-idf (term frequency-inverse document frequency) term weighting, completed with standard Rochio feedback processing method [3]. We briefly describe how these methods are applied to the problem of traceability.

The tracing tasks we consider involve two textual artifacts of the software lifecycle, e.g., a requirements document and a design document. Both artifacts are split into individual *elements*. The tracing task is to build a mapping *from* the elements of one artifact, referred to as *high-level document,* to the elements of the second artifact, which we call *low-level document*.

The IR method we use, vector space retrieval, converts each textual element into a vector of *keyword weights*. If $V=\{t_1,\ldots,t_N\}$ is the list of all keywords found in the artifacts, then an element d is represented as a vector $d=\{w_1,\ldots,w_N\}$ of keyword weights, where each keyword weight w_i is computed as the product $wi=tf_i{*}idf_i$. Here, tf_i, called *term frequency* of the keyword, is the normalized frequency of the occurrences of the keyword w_i in our element d. idf_i, known as the *inverse document frequency* of w_i is computed as $idf_i = log(M/M_i)$ [3], where M is the total number of elements in the document and M_i is the number of elements that contain w_i.

For each high-level document element, the vector space retrieval method provides a *ranking* of low-level document elements based on the similarity score between them. The similarity between two vectors d and q constructed as described above is computed as the cosine of the angle between the vectors:

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

The quality of the rankings obtained this way is measured through *precision* and *recall*. Precision, measuring the accuracy of the ranking, is the percentage of retrieved links that are correct. Recall, measuring the coverage of the ranking is the percentage of correct links that were retrieved.

After the ranked candidate link list is built, it can be improved via the feedback processing mechanism. Feedback processing involves examining a subset of the links in the candidate link list, and determining whether each link is correct or not. Let q be a high-level requirement and D_q be the set of all low-level requirements retrieved by our IR method. Suppose a subset of D_q was examined and broken into two sets: R_q and I_q, of relevant (correct) and irrelevant (false positive) links. Standard Rochio feedback processing, the method we use in this study, uses this information to change the vector q as follows:

$$q_{new} = \alpha q + \frac{\beta}{|R_q|} \sum_{d \in R_q} d - \frac{\gamma}{|I_q|} \sum_{d \in I_q} d.$$

Here, α, β, and γ are normalizing constants, which indicate the relative importance of the original vector (α), positive information (β), and negative information

(γ). We note here that the presence of positive feedback may affect the recall of the candidate link list (new relevant links may be retrieved), while the presence of negative feedback may affect its precision (other false positives can potentially be removed from the list).

3. Simulating Analyst

In our study we have *simulated* analyst behavior described below in this section. While in our future work we are planning on studying the work of analysts doing tracing *in-vitro* as well as *in-vivo*, this study did not use live analysts. Rather, we have designed a number of analyst behaviors, and implemented each behavior as a program that takes as input the candidate RTM, and interacts with our feedback processor at appropriate stages. Because this study is a *simulation* for analysts, we only accounted for differences in the simulated behavior.

For this study, we assumed the following paradigm. A task of *after-the-fact* tracing[2] the elements of one textual artifact to another is given to a human analyst. The analyst has at his disposal a software tool which can prepare candidate link lists and modify them through the feedback processing mechanism, as described in Section 3. The variable part of our simulations is the "front end" of the software. In our study, we simulate four different ways in which the analyst can interact with the software in order to complete the task. We document these four approaches and our assumptions about the analyst below.

For the analyst interaction with the software, we use two parameters: the order in which the analyst is shown the candidate links, and the use of feedback. The four cases we consider are:

Global ordering without feedback. The result of the work of the IR method is a collection of candidate link lists: one for each high-level element. Each link comes with a similarity score. In this method, the software first merges all the candidate link lists into a single list, and sorts it in descending order based on the similarity score. The software front-end displays candidate links (i.e., the text of both high- and low-level elements) one-at-a-time in descending order of similarity, and asks the analyst to either accept or reject the link. No feedback is used.

[2] I.e., we assume that the two artifacts represent the final versions of the documents, and do not change over time.

Local ordering without feedback. In this approach, the analyst is also shown links one-at-a-time, but in a different order. For each high-level requirement, the analyst is first shown the top candidate link (the high-level requirements are sorted in the document order). After that, the analyst is shown the second highest-scoring link for each requirement, and so on. There is no feedback. However, if the analyst sees the last correct link retrieved for a specific high-level element, the analyst conveys this information to the software, and the software stops showing links for this element from that point on.

Global ordering with feedback. This approach works exactly the same way as global ordering without feedback, except that after each choice is made by the analyst, the feedback is run for the high-level element that the analyst was observing. Then, the resulting new list of candidate links obtained from the feedback method is incorporated into the globally sorted list again (thus, the order in which links are shown to the analyst is affected on each stage).

Local ordering with feedback. This method is similar to local ordering without feedback, except that the feedback method is run after each analyst decision.

In setting up our simulations, we make two assumptions about our analysts:

1. The analyst always correctly identifies the nature of the observed link, and
2. The analyst is able to determine when a high-level element is completely satisfied (all children elements have been found).

While in practice these two assumptions may not always hold, our reasons for using them are quite straightforward. Automated tracing methods and techniques must be built and tested assuming perfect feedback from the analyst. That is, if our methods cannot provide good results with perfect feedback, the results will certainly not improve when the feedback is imperfect. In a similar vein, we view our *"perfect"* analyst as being able to determine when a specific requirement is completely satisfied.

Measures. We are interested in establishing the amount of *analyst effort* spent on a tracing task. As the direct measure of analyst effort, we use the *number of observed candidate links* that the analyst has to study and accept or reject during the run of the method. We use precision and *confirmed recall* (i.e., recall within the observed set of candidate links) to establish the quality of the final mapping produced by the analyst. We also use *selectivity* to measure the relative effort of the analyst. Selectivity is computed as:

$$selectivity = \frac{n_{observed}}{m \cdot n},$$

where $n_{observed}$ is the total number of links observed by the analyst, and m and n are the number of high-level and low-level elements, respectively. Thus, selectivity measures the percentage of all possible links that the analyst has examined.

4. Experiment

4.1. Experimental Design

We have conducted two simulation studies of the analyst effort. The first study compared the (simulated) analyst effort required to achieve a predefined recall level for the four methods described in Section 4. The second study fixed analyst effort and compared the accuracy (recall and precision) achieved for this effort. The second study used the four methods described above and a random simulation. Some may wonder why a random simulation was examined. Menzies et al. [15] note that software analysis should "start with random methods because they are so cheap, moving to the more complex methods only when random methods fail."

Table 1. CM-1 dataset overview.

Dataset Name	**CM-1**
# elements in requirements document (high-level)	235
# elements in design document (low-level)	220
# correct links	361
Total # of retrieved candidate links	36,556
Total # of correct links retrieved	358
Recall	0.99
Precision	0.001
Selectivity	0.707

Table 2. Results of the first experiment.

Method	Confirmed True Links	Observed Links	Precision	Recall	Selectivity
Local, Feedback	321	1595	0.20	0.89	0.03
Local, No Feedback	321	1713	0.19	0.89	0.03
Global, Feedback	321	5399	0.06	0.89	0.10
Global, No Feedback	321	6149	0.05	0.89	0.12

Table 3. Results of the second experiment.

Method	Observed Links	Confirmed True Links	Precision	Recall
Local, Feedback	1595	321	0.20	0.89
Local, No Feedback	1595	326	0.20	0.875
Global, Feedback	1595	236	0.15	0.65
Global, No Feedback	1595	227	0.14	0.63
Local, Random	1595	65	0.04	0.18
Global, Random	1595	58	0.036	0.16

Dataset. For this study, we used CM-1 [6], a sanitized dataset for a NASA scientific instrument. The CM-1 dataset from NASA comprises many artifacts: source code, requirements specification, test cases, design specification, etc. In prior maintenance-related work, static metrics from CM-1 were extracted and placed that in the PROMISE repository and used in a number of studies. **We did not use these static metrics (now commonly referred to as CM-1 dataset) for this study.** Instead, we have used two original CM-1 artifacts: the requirements specification and the design specification and the RTM of the relationships between these two artifacts that we have built and submitted it to the PROMISE repository in conjunction with [16]. Table 1 lists the basic characteristics of the CM-1 dataset and the properties of the candidate link list retrieved by the vector space retrieval method.

As can be seen from this table, the unfiltered candidate link list returned by the automated method captures almost all of the correct links, but introduces an enormous amount of noise. The four ways of simulating analyst behavior are designed to give the analyst the opportunity to *not consider* every single retrieved candidate link.

Study 1. Analyst effort for fixed recall. For the first experiment, we elected to compare the effort of the simulated analyst at a fixed recall level for the four methods described in Section 4. Our original intention was to fix our test recall level at 90% (325 out of 361 correct links retrieved). However, our experiments established that some methods retrieve only 321 correct links for recall of 88.9%. We used this latter level to compare the analyst effort.

Study 2. Recall for fixed analyst effort. After looking at the results of our first experiment, we asked ourselves whether we could get better insight into *where* the analyst effort is *wasted* by comparing the recall obtained from all four methods with fixed effort. We considered the recall for our four methods at roughly the same effort level. We took *1595 observed candidate links*, the smallest effort from the first study, as our benchmark effort for this study.

In addition to comparing the four methods described in Section 4 to each other at fixed effort level, we also simulated two *random selection* methods and compared the recall at the fixed level from our four methods to the random simulation. The two random simulations are described below:

Global Filtered Random Selection. In this method, we ordered all candidate links in the candidate link lists by their similarity score and filtered out the bottom 75%. The simulation selected one-at-a-time, without return, a link from the remaining top 25% of the links. Each link was selected with equal probability and 1595 links were selected.

Local Filtered Random Selection. In this method, the candidate link list for each high-level element was pruned at the 25% level, and then an attempt was made to draw 7 or 8 links (1595/220 = 7.25), without return, from the pruned list.

We ran each random simulation 1000 times and use medians (which proved to be very close to the averages in both cases, see Table 4) for comparison.

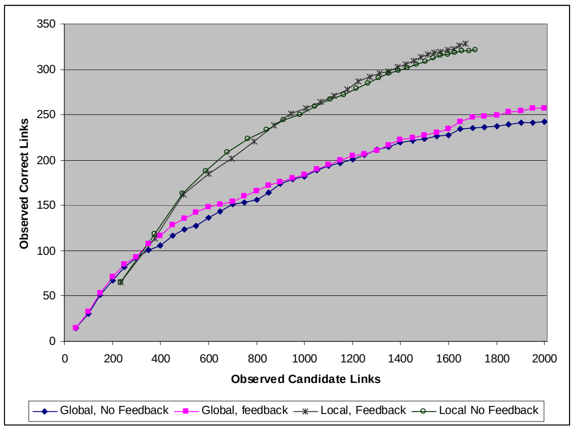

Figure 1. Simulation internals: observed correct links vs. all observed links during the four simulations

4.2. Results

We present the results of our experiments below. In Table 2, we show the results of the first experiment. As can be seen from this table, the biggest factor in determining the analyst effort was whether or not the method used local or global sorting. To be more precise, our "local" methods simulate the analyst providing the software the *"I see all the children links for this high level link, proceed to the next link"* command**.** Table 2 shows that the analyst's ability to stop at the "right place" has a drastic impact on the total effort. In addition, we observe that there is a benefit to feedback. The savings in effort are 7%

for the “local” methods when feedback is used and 13% for the “global” methods.

Table 3 shows how each method compares at the level of 1595 observed links. We can see, that, as expected, random methods cannot produce any meaningful results (for the sake of completeness, we present some standard descriptive statistics for the two random simulations in Table 4). Figure 1 shows the internals for the four non-random simulations we ran, plotting the number of correct links observed versus the total number of observed candidate links. As can be seen from the figure, the local ordering methods, in which we simulated the “I see all the children” command, find the majority of the correct links relatively quickly: at around 1000 observed links. The local simulations discovered around 250 correct links (for the confirmed precision of 25%). For the global ordering methods, correct links are observed much more uniformly throughout the first 2000 observed links, with the rate of observation of new correct links tapering off.

Table 4. Statistics from the random simulations.

Method	Local, Random	Global, Random
# of simulation runs	1000	1000
# observed links	1595	1595
Min # correct links	45	39
Max # correct links	89	88
Mean # correct links	64.867	58.67
Median # correct links	65	58
St. dev. # correct links	7.72	6.66

4.3. Analysis

Looking at our simulations, we can make a number of observations.

Knowing when to stop is crucial. The largest difference between the simulated analyst effort arises in simulations based on local sorting versus simulations based on global sorting of the links. The key reason for this is our decision to simulate the analyst correctly determining when (s)he has seen all links for a specific high-level requirement for the methods based on traversing individual candidate link lists without merging them[3].

Feedback helps, to a degree. In our implementation, feedback for an individual high-level requirement was processed almost instantaneously. Methods that used feedback showed an overall improvement of 7% to 13% in the simulated analyst method. While this improvement is not drastic, it is visible.

Have a system. All four non-random methods were designed to simulate a specific *structured* analyst behavior. Our random simulations represent *unstructured* analyst behavior. It is clear from the results in Table 3 that unstructured analyst behavior is highly inefficient.

This study is still preliminary, and should be viewed as a prelude to the study of the behavior of *real* (not simulated) analysts in tracing tasks. The results we observed provide guidelines to assist us in building user interfaces for the tracing software. They also suggest specific types of analyst-software interactions that need to be studied *in-vivo.*

5. Related Work

Though the generation of mappings is a general problem, it has mainly been investigated under the guise of “requirements traceability” or requirements tracing. Requirements tracing is defined, by Gotel and Finkelstein [5], as “the ability to follow the life of a requirement in a forward and backward direction.” All work to date has concentrated on the recovery or generation of traceability links between software engineering artifacts (structured as well as non-structured artifacts).

Antoniol et al. [1] applied the vector space model (also known as term frequency-inverse document frequency [3]) to the problem of recovering traceability links between a textual user’s manual and source code and between textual functional requirements and source code. They were able to achieve high levels of recall (93 – 100%), but were only able to achieve 13 – 18% precision. Antoniol et al. [2] also applied a probabilistic method to the problem of recovering links between source code and documentation. Though high precision was achieved (83%), it was done at the price of recall (39%). Note

[3] Our intuition is simple: an analyst looking at a sequence of links for the same high-level requirement is likely to be in a good position to call it quits at some point. The analyst looking at a merged list of candidate links may not be able to easily do so.

that for the purposes of V&V and IV&V, recall needs to be high (90% or higher).

Marcus and Maletic [14] applied latent semantic indexing (LSI) to the problem of recovering traceability links between documentation and source code (using the same dataset as Antoniol et al. [1]) and found that LSI performs at least as well as the vector space model while requiring less pre-processing of the artifacts. They achieved recall of 91 – 100% and precision of 13 – 18%. When they relaxed recall to 71%, they achieved precision of 43%. Cleland-Huang et al. developed a method for dynamically generating traceability data in a speculative manner for performance models that may be affected by a proposed change [4]. Links were established and maintained between the performance models and key requirements data that had been derived from the performance models.

In prior work, we found that simple keyword-matching methods, applied to the problem of tracing textual requirements to textual lower level requirements or to design elements, could achieve recall of 63% and precision of 39% [7]. This required much work on the analyst's part though, such as the building of a keyword ontology and/or the manual assignment of keywords to all low and high level elements. Examining the same problem, when a thesaurus was added to the vector space model, recall was 85% and precision was 40%. In later work, we introduced the notion of analyst feedback. Here, the analyst provides feedback on the top N elements of each candidate link list (yes this is a link, no this is not a link) and the feedback is used to modify the vectors for the high level elements before re-executing the matching algorithm. By adding feedback, we were able to improve recall to close to 90% with precision close to 80% (for the MODIS dataset) [12].

The current study focuses on the role of the analyst in requirements tracing. In earlier work [7], we undertook a study to compare an analyst: 1) performing tracing manually, 2) using a keyword-based tool, 3) using the output from a keyword-based tool, and 4) using our IR tool. The results showed that the analyst and the keyword-based tool achieved 63% recall and 39% precision, while using our tool achieved 85% recall and 40% precision. Next, we developed the requirements for a requirements tracing tool [8] and found that a number of the requirements had to do with the analyst, specifically the *utility* sub-requirement of *believability* (the analyst feels that the tool is useful), the *communicability* sub-requirement of *discernability* (the tool provides information, the process flow, and the results to the analyst in an understandable way), and the final requirement of *endurability* (the tool makes the tracing task as pleasant as possible).

Our first study of the analyst was undertaken using a small dataset (MODIS) and a number of requirements traceability matrices [10]. Some RTMs had low recall but high precision, some had low precision but high recall, and varied combinations in between. It was our belief that analysts would make better feedback decisions if given a high quality RTM (high recall and high precision), but would make bad decisions if given poor quality RTMs. What was observed was that all analysts made bad decisions (threw away true links and kept false positives). The study was very small, however, and general conclusions could not be drawn. We undertook a small study with graduate students and found that those using our IR tool (called Requirements Tracing On-target or RETRO) achieved statistically significantly higher recall (70.1% versus 33% for the group using no tool) but lower precision (12.8% versus 24.2%), but took far less time to complete the tracing task (three times less minutes (41.8 minutes versus 120.66) [11]**.** A small usability survey was also conducted. Students found the RETRO features that they used to be very useful. Students used most all of the features available to them. But there seemed to be a misunderstanding of the feedback feature.

To our knowledge, our work is the only that has been done to examine the role of the user in tracing work. The current work differs from the work mentioned above in several ways. First, it specifically focuses on the information that the analyst will see. Second, it examines a number of scenarios under which the analyst navigates the generated traceability links, without requiring a real analyst. Finally, it provides guidance on how to best work with feedback-driven information retrieval tracing methods.

6. Conclusions and future work

The analyst has the final say in many applications, such as a maintainer deciding which tool-retrieved code components are related to a bug report that is being addressed. In V&V and IV&V, analysts often work with tracing tools that retrieve pairs of textual artifacts that are deemed relevant. The analyst can save time and reduce errors by using such tools to assist with tracing tasks. But the analyst has the final say and must render decisions on the proposed links. As it is difficult to perform studies using real analysts, we have explored the use of predicting analyst behavior and then examining the impact on quality of the final trace and on analyst effort. The study suggests which facilities the analyst should have when working with the software, e.g., "I have seen all the links for this high-level requirement" command. It also suggests that the tracing software should structure its interactions with analysts, providing them with specific requests in specific order, rather than relying on analysts to wade their way through large fields of candidate links.

As mentioned above, this is a preliminary study. We have two basic avenues to explore: additional simulations to better understand the impact of specific tracing methods used to prepare candidate links, and *in-vivo* study of analyst interaction with tracing software. The results of the study reported in this paper will be used to build analyst guidance into user interfaces of the tracing software.

7. Acknowledgments

Our thanks go to Stephanie Ferguson and Marcus Fisher. We thank Senthil Sundaram for his assistance. We thank Mike Chapman and the Metrics Data Program (MDP) for access to the CM-1 dataset. This work is sponsored by NASA under grant NNX06AD02G.

7. References

[1] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering Traceability Links between Code and Documentation. *IEEE Transactions on Software Engineering*, Volume 28, No. 10, October 2002, 970-983.

[2] Giuliano Antoniol, Gerardo Canfora, Andrea De Lucia, Ettore Merlo: Recovering Code to Documentation Links in OO Systems. *WCRE 1999*: 136-144.

[3] Ricardo A. Baeza-Yates, Berthier A. Ribeiro-Neto: *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[4] J. Cleland-Huang, C.K. Chang, G. Sethi, K. Javvaji, H. Hu, and J Xia. Automating speculative queries through event-based requirements traceability. *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02*), Essex, Germany, Sept. 2002.

[5] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," *Proceedings of the First International Conference on Requirements Engineering*, pp. 94 – 101, 1994.

[6] MDP Website, CM-1 Project, *http://mdp.ivv.nasa.gov/mdp_glossary.html#CM1*.

[7] J. Huffman Hayes; A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval," in *Proceedings of the International Conference on Requirements Engineering (RE),* Monterey, California, September 2003.

[8] Jane Huffman Hayes, Alexander Dekhtyar, Senthil Sundaram, Sarah Howard, "Helping Analysts Trace Requirements: An Objective Look," in *Proceedings of IEEE Requirements Engineering Conference (RE)* 2004, Kyoto, Japan, September 2004, pp. 249-261.

[9] Jane Huffman Hayes, Alex Dekhtyar, Senthil Sundaram, "Text Mining for Software Engineering: How Analyst Feedback Impacts Final Results," *Proceedings of Workshop on Mining of Software Repositories (MSR),* associated with ICSE 2005, St. Louis, MO, May 2005, pp. 58 - 62.

[10] Jane Huffman Hayes, Alex Dekhtyar, Senthil Sundaram, "Humans in the Traceability Loop: Can't Live With 'Em, Can't Live Without 'Em,", in *Proc. Workshop on Traceability of Emerging Forms of Software Engineering (TEFSE'05),* Long Beach, CA, November 2005.

[11] Jane Huffman Hayes , Alex Dekhtyar, Senthil Sundaram, Ashlee Holbrook, Sravanthi Vadlamudi, Alain April, REquirements Tracing On target (RETRO): Improving Maintenance through Traceability Recovery, *University of Kentucky Technical Report UK* CS454-06, March 2006.

[12] J. Huffman Hayes, A. Dekhtyar, and S. Sundaram. Advancing Requirements Tracing: An Objective Look, *IEEE Transactions on Software Engineering*, Volume 32, No. 1, (January 2006), 4-19.

[13] Jane Huffman Hayes, Alex Dekhtyar, Ashlee Holbrook, Olga Dekhtyar, Senthil Sundaram, "Will Johnny/Joanie Make a Good Software Engineer?: Are Course Grades Showing the Whole Picture?," in *Proceedings of the Conference on Software Engineering Education and Training* (CSEET), Oahu, Hawaii, April 2006, pp. 175 - 182.

[14] A. Marcus, and J. Maletic. "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] Menzies, T., Owen, D., and Richardson, J. The Strangest Thing about Software. *IEEE Computer*, January 2007, pp. 54 – 60.

[16] Senthil Karthikeyan Sundaram, Jane Huffman Hayes and Alex Dekhtyar, Baselines in Requirements Tracing, in *Proceedings, PROMISE'2005: International Workshop on Predictor Models in Software Engineering* , St. Louis, MO, May 2005.