Towards a Model of Analyst Effort for Traceability Research

Alex Dekhtyar
CalPoly
San Luis Obispo
California
001 + 1 805 756 2387
dekhtyar@calpoly.edu

Jane Huffman Hayes, Matt Smith
University of Kentucky
Lexington
Kentucky
001 +1 859 257 3171
hayes@cs.uky.edu, matt40413@aol.com

ABSTRACT
This paper posits that a theoretical model of analyst effort in tracing tasks is necessary to assist with study of the analyst. Specifically, it is clear from prior work by numerous research groups that the important factors in such a model are: the amount of time it takes for an analyst to vet a given candidate link and the amount of time it takes an analyst to find a missing link. This paper introduces a theoretical model of analyst effort as well as a simplified model. A number of simulations were undertaken in order to build effort curves to assist in evaluating numerous tracing scenarios, such as determining at what point in time an analyst should switch from vetting candidate links to manually searching for links not in the candidate list.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**]: Tools.

General Terms
Measurement, Economics, Experimentation, Human Factors.

Keywords
Traceability, study of the analyst, effort, false positives, Type I error, Type II error.

1. INTRODUCTION
Traceability research concentrated on automating the tracing process has been proceeding in two distinct and complimentary directions. The first direction, the study of methods for automating tracing tasks [1][2][3][4], concentrates on automated analysis of textual artifacts in search of traceability links. The second direction, the study of the analyst interaction with automated tools, is emerging because industry tracing processes require the involvement of human analysts and places responsibility for the final traces on them. Preliminary results [5,6] indicate that there is a complex relationship between the quality of the candidate traces produced by automated methods and the quality of the final traces produced by the analysts.

The main goal of the research on automating tracing can be succinctly captured as follows: *reduce analyst effort while keeping traceability matrix accuracy high.* The majority of the traceability research [1][2][3][4][5] in this area concentrates on the latter part of the goal: preserving accuracy. Effort estimation in tracing tasks remains an understudied topic, although recently some important studies in this area have been conducted [6].

One explanation of why effort estimation is understudied relates to the relative ease with which accuracy can be evaluated and the relative difficulty with which effort can be evaluated in tracing tasks. Specifically, there are well-defined, community-accepted measures of accuracy such as *recall, precision, failure rate, f-measure,* and *average expected precision* which appeared concomitantly in traceability studies using information retrieval and text mining methods for automatic trace recovery. Analyst effort evaluation and estimation in tracing tasks, however, lacks a common effort model that looks beyond the time spent performing the tasks and addresses the structure of the tracing process. In this paper, we propose a family of such effort models. We partition the analyst work into specific individual subtasks and estimate the total analyst effort by controlling the relationship between the subtask analyst effort.

The paper is organized as follows. In Section 2, we present some background on traceability research and practical challenges that motivate the need for the model (why?). Our position is also stated. Section 3 describes the theoretical model (what?). Section 4 presents/discusses some simulations illustrating how the model could be used to guide analyst work. Section 5 discusses the challenges and questions that the proposed research will address in the future.

2. BACKGROUND
Much research has focused on the effectiveness of trace recovery techniques [1][2][3][4]. As this research became more mature and empirical validation ensued, it became clear that an uncontrolled (and possibly uncontrollable) variable in the tracing process is the human analyst, who has the final say on whether or not a link recovered by a tracing technique is correct [7]. Assuming that tracing techniques are as effective as possible, it is important to study the analyst role in tracing to ensure that techniques are efficient and optimally apply analyst effort.

Toward that end, the authors examined a number of possible ways that an analyst might interact with an automated method [8]. We found that the most important factor in reducing analyst effort was for the analyst to know "when to stop" examining items retrieved by the tool. The authors have examined the accuracy of feedback provided by analysts (this is a link, this is not a link; similar to

ICSE'11, May 21-28, 2011, Waikiki, Honolulu, HI, USA

Google's "more hits like this" feature) and whether or not it is influenced by the quality of the tool-provided links [7][9].

In examining the efficiency of tracing techniques, it seems reasonable that researchers should first ensure that automated techniques are more efficient than manual tracing. Early work has established this to be true [1][10]. Recent work, however, has challenged the notion that manual tracing may be much more time consuming than automated methods [11]. It appears that for small tracing problems, where the target elements have been restricted to a matching subset, human analysts may be as accurate as, and possibly even faster than, an automated tool [11]. Tracing needs to be applied to the artifacts of **large** software systems too, though. It is highly unlikely that large system tracing will be a manual undertaking, so it is important to ensure that automated techniques can assist a human analyst with the tracing task.

In most settings where an analyst is working with an automated tracing tool, a list of retrieved links between the high level or source artifact and the low level or target artifact is presented (e.g., tracing from a requirement specification to test cases), called candidate links (so called until a human confirms them). The analyst vets the candidate links, marking them as links or rejecting them as false positives. The analyst may also perform searches into the target artifact to look for links not retrieved by the tool.

We are interested in studying the analyst interaction with the tracing tool in an attempt to understand where most of the analyst effort is expended. Why? We do this so that we might either redesign/modify automated tools to take into account the least efficient aspects of the task and/or so that we can provide guidance to analysts on how to most efficiently interact with such a tool. Toward that end, we present a theoretical model of the analyst interaction with a tracing tool. We further examine ways to simplify the model so that it can be more easily simulated/studied. We illustrate the use of the model in a number of tracing scenarios[1].

POSITION: It is our position that a model of analyst effort is needed in order to study analyst behavior, in order to advise analysts on how to make the best use of their time, and to permit better comparison of manual and automated techniques.

3. MODEL OF ANALYST EFFORT

We propose a model of analyst effort in tracing tasks based on the specific types of analyst activities that occur during tracing. In particular, we model the effort in the following scenario.

An analyst is asked to perform a tracing task for a pair of artifacts. The high-level (source) artifact has N elements, the low-level (target) artifact has M elements. The analyst is using a software tool which produces a candidate traceability matrix (candidate TM). During the tracing process, the analyst essentially repeatedly performs the following three subtasks: **Confirmation of a link.** The analyst examines a candidate link from a software-generated candidate TM and **confirms** that this is a true link. **Rejection of a link.** The analyst examines a candidate link from a software-generated candidate TM and **establishes that it is a false positive.** The analyst then **rejects** the candidate link**. Addition of a link.** The analyst examines the low-level and the high-level artifacts and determines that **there exists a traceability link** between a pair of elements that was not captured in the software-generated candidate TM. The analyst searches for and **adds** the link. The analyst is done when (s)he has validated and/or discovered all true links between the source and target artifacts.

We propose to estimate the analyst effort when performing such a tracing task using the model described below[2]. We introduce the following notation. Let the true traceability matrix contain ***True*** number of links. Let the candidate traceability matrix contain ***Retrieved*** number of links, which we represent as

$$Retrieved = Hits + Misses,$$

where ***Hits*** is the number of true retrieved links and ***Misses*** is the number of retrieved *false positives.* In this situation, the analyst will **confirm *Hits*** links, will **reject *Misses*** links, and will have to **add *True – Hits*** links. We note that, in general, the effort an analyst spends on confirming or rejecting one link depends solely on the contents of the link itself (and is roughly proportional to the amount of text the analyst needs to read). At the same time, the more links the analyst has already examined, the less effort it will take for the analyst to ***add*** a new link to the trace, as the space of possible links to be added shrinks. As such, we introduce the following effort parameters. Let ***A*** denote the average effort it takes an analyst to **confirm** a link, ***B*** denote the average effort it takes an analyst to **reject** a link, and ***C*** denote the average effort it takes an analyst to **add** a link before the analyst has examined candidate links in the trace.

Using the notation described above, we estimate the analyst effort to complete the tracing process as follows:

$$Effort = A \cdot Hits + B \cdot Misses + C \cdot (True - Hits) \cdot \frac{M \cdot N - (Hits + Misses)}{M \cdot N}$$

That is, the analyst effort is estimated as the combination of: the analyst effort to confirm all true links from the candidate TM, the analyst effort to reject all the false positives from the candidate TM, and the effort to discover and add all the true links that were missing from the candidate TM. Notice that we estimate the latter effort (add links) as follows: we assume that the analyst *first evaluates the links from the candidate TM*. We then scale the effort it takes to discover each of the remaining links by a percentage of the potential links that have not been observed yet. For example, if a candidate TM retrieves 25% of all possible element pairs (links), we estimate that the effort it would take an analyst to add one undiscovered link to the final TM is 0.75 of the analyst's expected effort in a fully unobserved task.

Simplified Model. In the model above, we used different estimates for the effort it takes to reject and the effort it takes to confirm a link from a candidate TM. In practice though, *both tasks involve the same set of steps:* examining the text of the source and target elements and rendering a **yes/no** decision. We posit that we can, under normal circumstances, treat the effort of confirmation and the effort of rejection as being the same:

$$\boldsymbol{A = B.}$$

If this is the case, we can simplify our effort model:

$$Effort = A \cdot Retrieved + C \cdot (True - Hits) \cdot \frac{M \cdot N - Retrieved}{M \cdot N}.$$

[1] To keep this paper focused on the model, we use synthetic tracing scenarios here. Our forthcoming work will apply the model as an evaluation technique for effort estimation using real data.

[2] We note that while the model we propose assumes that a candidate TM is produced by a software tool, it can be used to estimate analyst effort in a pure manual tracing process by setting the candidate TM as empty. As such, the model is applicable to a wide range of tracing scenarios.

In general, effort is measured in time, or person-time. However, our effort estimates so far are unit-less. As such, we can simplify the model even further, by introducing the model parameter $\alpha = C/A$: the ratio of effort between dealing with a reported candidate link and discovery of an unreported true link. We can then represent our effort estimate in a way that only makes it dependent on a single parameter:

$$Effort = \text{Retrieved} + \alpha \cdot (\text{True} - \text{Hits}) \cdot \frac{M \cdot N - \text{Retrieved}}{M \cdot N}.$$

Discussion. The proposed parameter α has been used informally in past traceability studies [1][10][7]. It represents the intuition that it takes an analyst α times as much effort to discover an error of omission (find an unreported true link) than to correct an error of commission (reject a false positive). It is important to note that there is no *unique value* for α. In practice, the specific value of parameter α will vary from tracing task to tracing task and from analyst to analyst. The value of the model is in the fact that we can substitute multiple values of α and produce multiple observations (see Section 4 for specific examples).

Given a value for a parameter α and a tracing scenario, we can estimate the analyst effort to recover the correct trace, based on the quality of the candidate TM (number of true links retrieved and number of false positives) and analyst behavior (how many links the analyst has inspected). Next, we examine a number of scenarios under which the model has been applied.

4. USE OF THE MODEL

To emphasize position of this paper (the effort model itself), we illustrate the variety of possible uses for the proposed effort estimation model using a number of *hypothetical examples.* We plan to provide analysis of the actual tracing experiments using the proposed effort model in a number of forthcoming works.

Example 1. Consider a tracing scenario where two automated methods, *TraceIt* and *AutoTrace,* are applied to a tracing problem. For illustrative purposes, let the source document contain 100 elements and the target document contain 200 elements for a total of 20,000 possible links between the two documents. Let there be 500 *true links* in the actual TM. Let *TraceIt* retrieve 5000 candidate links, of which 200 are true links (for 4% precision and 40% recall). Let *AutoTrace* retrieve 6000 candidate links of which 300 are true links (for 5% precision and 60% recall).

Notice that *AutoTrace* produces a candidate TM with ***both*** higher precision and higher recall. So, according to the traditional accuracy-based evaluation techniques, *AutoTrace* appears to be preferable to *TraceIt.*

Our effort model allows us to introduce some nuances to such an evaluation. We note that, given the total number of possible links, the number of retrieved links, and the number of true links and true retrieved links, our effort model becomes a linear function in α. The effort estimates for the two methods are:

$$\text{TraceIt: } Effort = 5000 + \alpha \cdot 300 \cdot \frac{15000}{20000} = 225 \cdot \alpha + 5000$$

$$\text{AutoTrace: } Effort = 5000 + \alpha \cdot 200 \cdot \frac{14000}{20000} = 140 \cdot \alpha + 5000$$

These estimates are made under the assumption that the analyst will examine all retrieved links and then will spend time finding all missing links in each case (all links will eventually be found). Figure 1 shows the graph of the effort estimate functions.

As seen from this graph, the method accounting for the lesser expected effort depends on the value of the parameter α. For smaller values of α (from 1 to 11), *TraceIt* (dashed line on the graph) accounts for lesser effort, despite producing a less accurate candidate TM. For larger values of α, *AutoTrace* leads to better estimated effort.

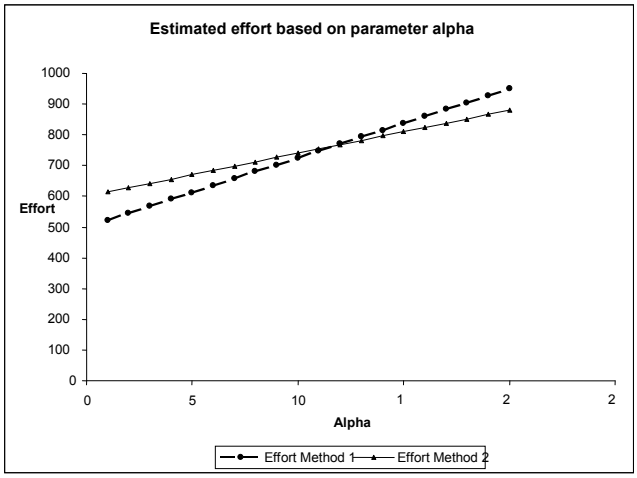

Figure 1. Effort Estimate Functions, Example 1

Example 2. In this example, we show a more involved method analysis scenario, which estimates effort at each step of the tracing process. Consider a task of tracing a source document with 50 elements to a target document with 100 elements for a total of 5000 combinatorial links. The TM contains 50 true links. Two methods, *RuleTrace* and *iTrace* are applied to the task. Both *RuleTrace* and *iTrace* retrieve 225 links, and both retrieve 45 true links among them for 90% *recall* with 20% *precision.* Both methods order candidate links by descending similarity score.

Suppose an analyst works using an environment in which (s)he is checking links in the order of their similarity score (examine highest first). *RuleTrace* orders candidate links in such a way that a true link shows up every five links. That is, there is one true link in the top five links, there is one more in the next five, and so on. When examined in this order, recall increases in an evenhanded manner, and precision is kept around the 20% level throughout the entire process of examining the retrieved candidate links. *iTrace* has 25 true links in the top 50 links distributed as follows: the first five links are all true, three out of the next five are true, and 2 to 3 links out of each of the following five-tuple of candidate links are true. After that, the following 130 links are *false positives*. Finally, the last 45 candidate links contain the remaining 20 true links roughly evenly distributed.

Figures 2 and 3 show the estimated effort throughout the trace validation process for each of the methods. We used five different values of the parameter α: 2, 5, 10, 15, and 20. The estimated effort is plotted against the number of observed links. The graphs simulate the following observation:

> *For as long as there are candidate links from the automatically generated candidate trace, at each step the analyst has a choice: examine the next candidate link* ***or stop link examination*** *and begin retrieving missing links.*

The estimated effort in the graphs is plotted against the number of candidate links confirmed/rejected by the analyst. For *iTrace,* assuming that **α**= 10, if the analyst observes the first 50 links and then switches to recovery of missing links, her estimated effort is 297.5; if the analyst chooses instead to examine 200 links, her effort estimate is 324.8. Of these two choices, it is preferable for the analyst to stop after examining the first 50 links.

In Figures 4 and 5, we compare the effort estimation curves for the same values of **α** (5, 10) for *RuleTrace* and *iTrace.* From these curves, we can see two things. **Observation 1**: at which point the absolute minimum estimate for each method comes: 246.875 for *RuleTrace* for **α**=5 at 50 observed links (Figure 4), 272.8 for RuleTrace at **α** =10 at 225 observed links (Figure 5); 173.75 for iTrace at **α** =5 at 50 observed links (Figure 4) and 272.75 for iTrace at **α** =10 at 225 observed links (Figure 5). **Observation 2**: which circumstances lead to (a) expected analyst effort being essentially independent of the number of observed candidate links; (b) expected analyst effort *increasing* with the increase in number of observed links; and (c) expected analyst effort *decreasing* in the number of observed candidate links. This is essentially controlled by the relationship between **α** and the precision of the method. Whenever ***precision ≈ 1/ α*****,** the search for a missing link takes about the same time as confirmation of a single true link (together with the rejection of nearby false positives), and thus the expected effort will remain roughly the same regardless of where the analyst chooses to switch from candidate link vetting to missing link search. When *precision* < 1/ **α,** it is faster to discover missing links than sort through the candidate TM, so the fewer candidate links that are vetted, the better the effort estimate. Finally, if *precision* > 1/ **α,** then link confirmation is more productive than searching for links: the more candidate links vetted, the lower the effort estimate.

From Observation 1, it is clear that lower values of **α** yield lower effort estimates. Also, the "winner" appears to be iTrace with **α** of 5 and effort of 173.75 (the lowest effort estimate). From Observation 2, it is likely that an analyst would prefer RuleTrace over iTrace because effort is constant: it does not matter when an analyst switches to looking for missing links. Note that this is in contrast to Observation 1 that iTrace has the lowest effort estimate.

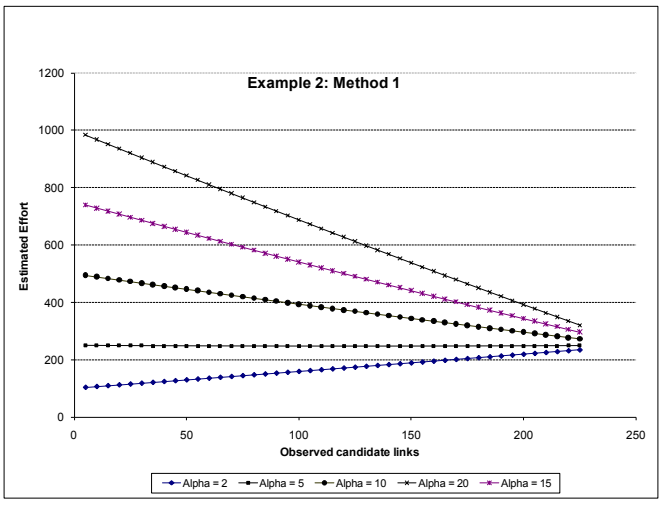

Figure 2.Effort Estimate for Alpha Values, RuleTrace, Example 2

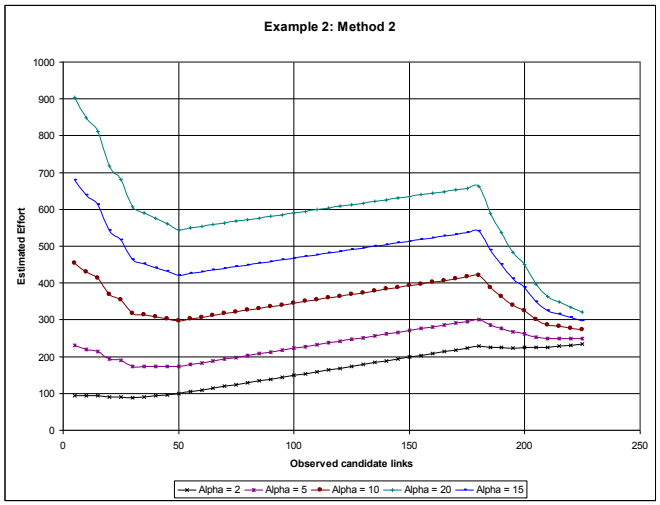

Figure 3. Effort Estimate for Alpha Values, iTrace, Example 2

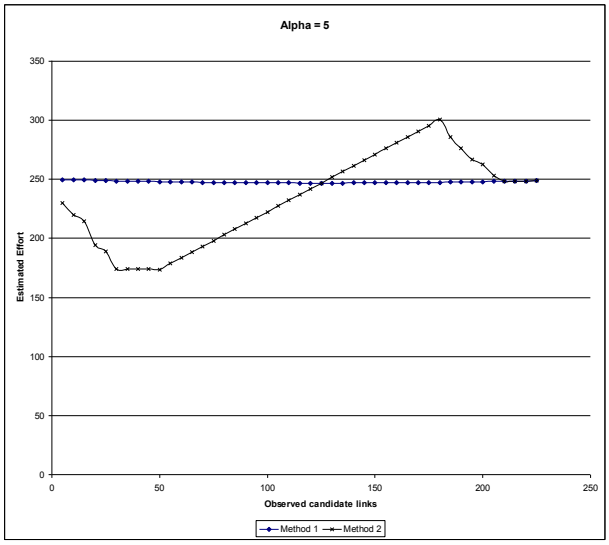

Figure 4. Effort Estimates for Alpha = 5, Example 2

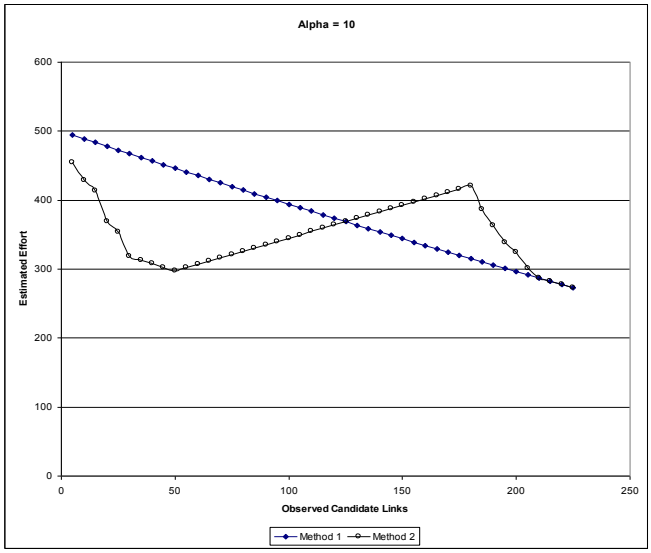

Figure 5. Effort Estimates for Alpha = 10, Example 2

It should be noted that the specifics of these examples are not that important. What is important is that we can use the proposed model to look at different tracing scenarios and compare them to each other in an apples-to-apples fashion. It is clear that the underlying issue is the comparison of proportions under specific assumptions about **α**. But **α** is important because it varies from task to task and from person to person. The model, therefore, is adaptable to the specifics of individual tasks.

Note also that the model is proposed for use in *traceability studies*, where we can observe key parameters of the model related to the true TM: number of true links, number of true links retrieved by an automated method, etc. It is possible to apply this model in practice to study the estimated ranges of analyst effort when the true TM is not available: we can fix **α**. Information about the true retrieved links will be available to us, but not information about the total number of true links. We can estimate the number of true links and construct estimated effort using the model for different values of the size of the true TM.

5. WHAT TO STUDY AND HOW TO STUDY IT?

We posited that an effort estimation model is necessary in order to study the role of the human analyst in traceability recovery and validation. We proposed a theoretical and a simplified model that depends on one parameter: the effort to confirm/reject a link, denoted **α.** We examined two examples using the model. It appears that such a model can help compare tracing scenarios.

We envision that a number of scenarios can be examined, such as: Is it better to have 50% recall and 40% precision or 60% recall and 30% precision? What is better: a method that retrieves very few links but with high precision or a method that retrieves many true links but with low precision? What is better: to have true links show up evenly, or to have them show up in chunks at the top and at the bottom of candidate link lists? With this, we could "rate" a tool as "use with datasets of type N." We would need to characterize datasets to "predict" the link distributions (what is meant by "type N").

We invite traceability researchers to examine, evaluate, validate, and comment on the proposed model. We also welcome suggestions on scenarios that can be examined using the model.

6. ACKNOWLEDGMENTS

This work is funded in part by the National Science Foundation under NSF grants CCF-0811140 and by a Lockheed Martin grant.

7. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, and E. Merlo, "Recovering Traceability Links between Code and Documentation," *IEEE Transactions on Software Engineering*, vol. 28, 2002, pp. 970-983.

[2] 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*, pp. 125-135.

[3] J. Cleland-Huang, C. Chang, G. Sethi, K. Javvaji, H. Hu, J. Xia,"Automating speculative queries through event-based requirements traceability," *Proceedings of the IEEE Requirements Engineering Conference* 2002, pp. 289-296.

[4] J.H. Hayes, A. Dekhtyar, and S. Sundaram, "Advancing Candidate Link Generation for Requirements Tracing: The Study of Methods," *IEEE Transactions on Software Engineering*, vol. 32, 2006, pp. 4 - 19.

[5] G. Spanoudakis, A. Zisman, E. Perez-Minana, P. Krause, "Rule-Based Generation of Requirements Traceability Relations,"*Journal of Sys.and S/W*, v72,2004, pp. 105-127.

[6] A. Egyed, F. Graf, and P. Grunbacher, "Effort and Quality of Recovering Requirements-to-Code Traces: Two Exploratory Experiments," *Requirements Engineering, IEEE International Conference on*, 2010, pp. 221-230.

[7] J.H. Hayes, A. Dekhtyar, "Humans in the traceability loop: can't live with 'em, can't live without 'em," *Proceedings of the 3rd international workshop on Traceability in emerging forms of software engineering*, 2005, pp. 20–23.

[8] A. Dekhtyar, J.H. Hayes, J. Larsen, "Make the Most of Your Time: How Should the Analyst Work with Automated Traceability Tools?," *Proceed. of 3rd Internat'l Workshop Predict. Models in Software Engineering*, 2007.

[9] D. Cuddeback, A. Dekhtyar, and J. Hayes, "Automated Requirements Traceability: The Study of Human Analysts," *Requirements Engineering, IEEE International Conference on*, 2010, pp. 231-240.

[10] J.H. Hayes, A. Dekhtyar, and J. Osborne, "Improving Requirements Tracing via Information Retrieval," *International Conference on Requirements Engineering, Monterey, California*, 2003, pp. 151-161.

[11] A. Egyed, F. Graf, and P. Grunbacher, "Effort and Quality of Recovering Requirements-to-Code Traces: Two Exploratory Experiments," *Requirements Engineering, IEEE International Conference on*, 2010, pp. 221-230.

[12] W. Kong, J.H. Hayes, A. Dekhtyar, J. Holden, "How Do We Trace Requirements? An Initial Study of Analyst Behavior in Trace Validation Tasks," *Proceedings of Cooperative and Human Aspects of Software Engineering*, May, 2011.