International Journal of Engineering Inventions
e-ISSN: 2278-7461, p-ISSN: 2319-6491
Volume 3, Issue 6 (January 2014) PP: 01-07

An Efficient Approach for Requirement Traceability Integrated With Software Repository

P.M.G.Jegathambal[1], P.Sheela Gowr[2]
[1]P.G Student, Tagore Engineering College, Chennai.
[2] Asst.Professor, Vels University, Chennai.

ABSTRACT: *Traceability links between requirements of a system and its source code are helpful in reducing system conception effort. During software updates and maintenance, the traceability links become invalid since the developers may modify or remove some features of the source code. Hence, to acquire trustable links from a system source code, a supervised link tracing approach is proposed here. In proposed approach, IR techniques are applied on source code and requirements document to generate baseline traceability links. Concurrently, software repositories are also mined to generate validating traceability links i.e. Histrace links which are then called as experts. Now a trust model named as DynWing is used to rank the different types of experts. DynWing dynamically assigns weights to different types of experts in ranking process. The top ranked experts are then fed to the trust model named as Trumo. Trumo validates the baseline links with top ranked experts and finds the trustable links from baseline links set. While validating the links, Trumo is capable of discarding or re-ranking the experts and finds most traceable links. The proposed approach is able to improve the precision and recall values of the traceability links.*

Index Terms: *Traceability, requirements, features, source code, repositories, experts.*

I. INTRODUCTION

Requirement traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)" [1]. Traceability links between requirements1 of a system and its source code are helpful in reducing system comprehension effort. They are also essential to ensure that a system's source code is consistent with its requirements and that all and only the specified requirements have been implemented by developers. Yet, during software maintenance and evolution, as developers add, remove, or modify features, requirement traceability links become obsolete because developers do not/cannot devote effort to update them [2]. Yet, recovering later these traceability links is a daunting and costly task for developers. Consequently, the literature proposed methods, techniques, and tools to recover these traceability links semi-automatically or automatically [3]. Requirements traceability has received much attention over the past decade in the scientific literature. Many researchers used information retrieval (IR) techniques, e.g., [2], [3], [4], to recover traceability links between high-level documents, e.g., requirements, manual pages, and design documents, and low-level documents, e.g., source code and UML diagrams [3], [4], [5], [6]. IR techniques assume that all software artifacts are/can be put in some textual format. Then, they computed the textual similarity between two software artifacts, e.g., the source code of a class and a requirement.

A high textual similarity means that the two artifacts probably share several concepts [3] and that, therefore, they are likely linked to one another. The effectiveness of IR techniques is measured using IR metrics: recall, precision, or some average of both, like the F_1 score [3], [5], [7]. For a given requirement, recall is the percentage of correct recovered links over the total number of pertinent, expected links while precision is the percentage of correct recovered links over the total number of recovered links. High recall could be achieved by linking each requirement to all source code entities (classes, structures, methods, and functions), but precision would be close to zero. High precision could be achieved by reporting only obvious links, but recall would be close to zero. Either extreme cases are undesirable because developers then would need to manually review numerous candidate links to remove false positives and-or study the source code to recover missing links [3]. Hence, the literature proposed IR-based techniques that sought a balance between precision and recall. During software maintenance and evolution, developers often evolve requirements and source code differently. Indeed, they often do not update requirements and requirement-traceability links with source code. Requirements and source code consequently diverge from each other, which decreases their textual similarity. Thus, IR techniques (or any combination thereof [8], [9]) typically produce links with low precision and-or recall because, due to their very nature, they depend on the textual similarity between requirements and source code [10]. Yet, while developers may not evolve requirements in synchronization with source code, they frequently update other sources of information, including CVS/SVN repositories,

bug-tracking systems, mailing lists, forums, and blogs. These other sources of information to build improved traceability-recovery approaches are exploited.

1. Mine software repositories, e.g., CVS/SVN repositories, to support the traceability recovery process and improve the precision and recall of IR-based traceability recovery approaches.
2. Heterogeneous sources of information is taken as experts whose opinions can combine using a trust model to discard/re-rank the traceability links provided by the IR techniques to improve accuracy.
3. An automatic, dynamic, per-link weighting technique is used rather than some global static weights to combine the opinions of experts.

A traceability-recovery approach uses heterogeneous sources of information to dynamically discard/re-rank the traceability links reported by an IR technique. The proposed system consists of three parts:

1) Histrace mines software repositories to create links between requirements and source code using information from the repositories. Histrace stores all the recovered links between requirements and software repositories in dedicated sets, e.g., Histrace commits and Histrace bugs, which are considered as experts whose opinions will be used to discard/re-rank baseline traceability links. For example, Histrace mines CVS/SVN repository to link requirements and source code using commit messages and provide the set of expert Histrace commits.

2) Trumo combines the requirement traceability links obtained from an IR technique and discards/reranks them using an expert's opinions and a trust model inspired by Web-trust models. It compares the similarity of the recovered links with those provided by the experts and with the number of times that the link appears in each expert's set. It is not tied to any specific IR-based traceability-recovery approach and can use any expert's opinion to adjust the ranking of recovered links. For example, the experts can be Histrace commits and–or Histrace bugs, taking advantage of CVS/SVN repositories and bug-tracking systems.

3) DynWing computes and assigns weights to the experts in the trust model dynamically, i.e., on a per link basis. Dynamic-weighting techniques are promising to assign weights for each link. DynWing analyses each expert's similarity value for each link and assigns weights according to these values.

The benefits of mining software repository and using trust models are

1. Use two experts, $\text{Histrace}_{\text{commits}}$ when mining CVS/SVN and $\text{Histrace}_{\text{bugs}}$. When mining bug reports.
2. DynWing is proposed, a dynamic weighting technique to automatically assign weights to different experts on a per-link basis.
3. DynWing with a PCA-based weighting is compared technique and with a static weighting technique to analyse the potential improvement of using different weighting techniques.
4. The impact of Trustrace on another IR technique, i.e., the Jensen-Shannon similarity model.

II. RELATED WORK

2.1 TRACEABILITY APPROACHES

TraceM based on techniques from open-hypermedia and information integration. TraceM manages traceability links between requirements and architecture.

Developers to keep source code identifiers and comments consistent with high-level artifacts. The approach computes textual similarity between source code and related high level artifacts, e.g., requirements. The textual similarity helps developers to improve their source code lexicon. An integrated approach to combine orthogonal IR techniques, which have been statistically shown to produce dissimilar results. Their approach combines VSM, JSM, and relational topic modeling. Their proposed approach uses each IR technique as an expert and uses a PCA-based weighting scheme to combine them.

The precision and the recall of the links recovered during traceability analyses are influenced by a variety of factors, including the conceptual distance between high-level documentation and low-level artifacts, the way in which queries are formulated, and the applied IR technique. Comparisons have been made between different IR techniques, their approach assumes that, if two or more files co-change in the system history, and then there is a possibility that they have a link between them. However, it is quite possible that two files are co-changing but that they do not have any semantic relationship. It is also likely that some documents evolve outside the system's version control repository and, in such a case; their approach cannot find a link from/to these documents. In addition, their approach does not analyze the contents of the CVS/SVN commit logs and files that were committed in CVS/SVN. More importantly, in co-change-based traceability, if two or more files have a link but they were not co-changed, then these approaches fail to find a link. It does not dependent on co-changes and overcomes these limitations.

2.2 WEB TRUST MODEL

There are two types of trust in e-commerce: first, a customer's initial trust when interacts with a Web site for the first time and, second, the Web site's reputation trust that develops over time and after repeated experiences. When customers hesitate to buy things online, they may ask their friends, family, and other buyers to make sure that they can trust a Web site. Many researchers investigated the problem of increasing

customers trust in a Web site. Some researchers suggested that using other sources of information can increase the trust in a Web site. Approach uses traceability from requirements to source code as initial trust. Then uses CVS/SVN commit logs and also uses bug reports, mailing lists, temporal information and so on, as reputation trust for a traceability link. As the reputation of a link increases, the trust in this link also increases.

III. TRUST BASED TRACEABILITY

Trustrace uses software repositories, e.g., CVS/SVN repositories and bug-tracking systems, as experts to trust more or less some baseline links recovered by an IR technique and, thus, to discard/reranks the links to improve the precision and recall of the IR-based techniques.

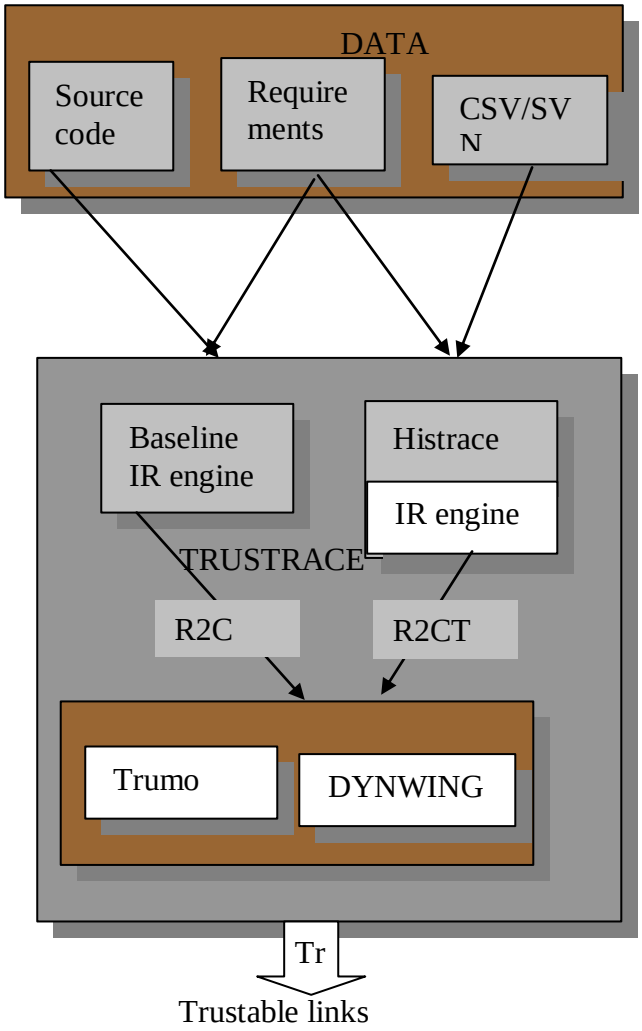

Fig 1. Trust based requirement traceability process

3.1 IR PROCESS

IR-based RTAs process is typically divided into three main steps.

1. All the textual information contained in the requirements and source code is extracted and pre-processed by splitting terms, removing stop words and remaining words are then stemmed to its grammatical root.
2. All the stemmed terms are weighted using a term weighting scheme, e.g., term frequency and inverse document frequency.
3. An IR technique computes the similarity between requirements and source code documents. Finally it generates a ranked list of potential traceability links.
   A high similarity between two documents shows a potential semantic link between them.

3.2 PRE PROCESSING

To create traceability links, extract all the identifiers from source code and terms from requirements. Some IR techniques as an engine to create links between requirements and source code. IR techniques assume that all documents are in textual format. To extract source code identifiers, use a source code parser, e.g., a Java parser. The parser discards extra information, e.g., primitive data types and language specific keywords, from the source code and provides only identifier names. The extraction of the identifiers and terms is followed by a filtering, stopper, and stemmer process.

1. A text normalization step converts all uppercase letters into lower-case letters. This step removes non-textual, i.e., some numbers, mathematical symbols, brackets, etc., information and extra white spaces, from the documents. Some identifiers/terms could be combined with some special characters, e.g., under score. Then split all the joined terms to make them separate terms. For example, SendMessage and send message are split into the terms "send message".
2. The input of this step is normalized text that could contain some common words, e.g., articles, punctuation, etc. These common words are considered as noise in the text because it does not represent semantics of a document. Thus, in this step, we use a stop word list to remove all the stop words. English language stop words list is used as all our documents are in English.

3. This step is stemming. An English stemmer, for example, would identify the terms "dog", "dogs" and-or "doggy" as based on the root "dog". An IR technique computes the similarity between two documents based on similar terms in both documents. However, due to different postfix, IR techniques would consider them, e.g., access, accessed, as two different terms and it would result into low similarity between two documents. Thus, it becomes important to perform morphological analysis to convert plural into singular and to bring back inflected forms to their morphemes.

3.3 TERM WEIGHTING SCHEMES

An IR technique converts all the documents into vectors to compute the similarities among them. To convert documents terms into vectors, each term is assigned a weight. Various schemes for weighting terms have been used Widely used weighting schemes are characterized in two main categories: probabilistic and algebraic models Probabilistic weighting models heavily depends on probability estimations of terms and algebraic weighting models depends on terms distribution in a document

3.4 TERM FREQUENCY (TF):

TF is often called local frequency. If a term appears multiple times in a document then it would be assigned higher TF than the others. TF is calculated as:

$$TF = {}^{n_{ij}}\Big/_{\sum_k n_{k,j}} \quad \text{.....(1)}$$

3.5 GLOBAL FREQUENCY (GF):

If a word appears in multiple documents then the term is not considered representative of documents content. The global frequency is also called inverse document frequency (IDF).
The IDF of a term is computed as

$$IDF = \log\left({}^{|D|}\Big/_{|d: t_i \in d|}\right) \quad \text{..... (2)}$$

If a term appears multiple times in a single or multiple documents then IR technique would recommend that document as relevant document to a query.

3.6 IR TECHNIQUE

To build sets of traceability links, we use some IR techniques, in particular VSM and JSM. JSM and VSM are used to recover traceability links. Both techniques essentially use term-by-document matrices. Consequently, we choose the well known T F/IDF weighting scheme for the VSM and the normalized term frequency measure for the JSM. These two measures and IR techniques are building blocks for traceability.

3.7 VECTOR SPACE MODEL

Many traceability links recovery techniques use VSM as baseline algorithm. In a VSM, documents are represented as vector in the space of all the terms. Various term weighting schemes can be used to construct these vectors. If a term belongs to a document then it gets a nonzero value in the VSM along the dimension corresponding

to the term. A document collection in VSM is represented by a term by document matrix, i.e., m ∈ n matrix, where m is the number of terms and n is the number of documents in the corpus.
Once documents are represented as vectors of terms in a VSM, traceability links are created between every two documents, e.g., a requirement and a source code class, with different similarity value depending on each pair of documents.

The similarity value between two documents is measured by the cosine of the angle between their corresponding vectors. Cosine values are in [−1, 1] but negative values are discarded and a link has thus a value in [0, 1] because similarity cannot be negative and zero between two documents. Finally,
the ranked list of recovered links and a similarity threshold are used to create a set of candidate links to be manually verified.
The angle between two vectors is used as a measure of divergence between the vectors. The similarity of requirement to source code can be calculated as follows

$$Sim(R,C) = R.C \Big/ ||R||.||C|| \quad \cdots\cdots\cdots\cdots\cdots\cdots (3)$$

Where R is a requirement vector and C is a source code vector.

3.9 JENSEN-SHANNON DIVERGENCE MODEL

The JSM is an IR technique proposed by It is driven by a probabilistic approach and hypothesis testing technique. JSM represents each document through a probability distribution, i.e., normalized term-by-document matrix. The probability distribution of a document is

$$pb_{i,j} = n(w,d) \Big/ T_d \quad \cdots\cdots\cdots\cdots\cdots\cdots (4)$$

Where n (w, d) is the number of times a word appears in a document d and T_d is the total number of words appearing in a document d. The empirical distribution can be modified to take into account the term's global weight, e.g., IDF. After considering the global weight, each document distribution must be normalized as

$$p_{i,j} = pb_{i,j}.IDF_{i,j} / \sum_{i=0}^{n} pb_{i,j}.IDF_{i,j} \quad \cdots\cdots\cdots\cdots\cdots (5)$$

Where $pb_{i,j}$ and $IDF_{i,j}$ is the probability distribution and inverse document frequency of i^{th} term in j^{th} document, respectively.

Once the documents are represented as probability distribution, JSM computes the distance between two documents' probability distribution and returns a ranked list of traceability links. JSM ranks source documents, e.g., requirements, via the "distance" of their probability distributions to that of the target documents, e.g., source code

HISTRACE

Histrace creates links between the set of requirements, R and the source code, C, using the software repositories T_i, i.e., in the following, T_1 stands for CVS/SVN commit messages and T_2 for bug reports. Histrace considers the requirements' textual descriptions, CVS/SVN commit messages, bug reports, and classes as separate documents. It uses these sources of information to produce two experts $Histrace_{commits}$ and $Histrace_{bugs.}$ $Histrace_{bugs}$ use CVS/SVN commit messages and bug reports respectively, to create traceability links between R and C through the T_i. Below, we discuss each step of Histrace in details.

DOCUMENT PRE-PROCESSING

Depending on the input information source (i.e., requirements, source code, CVS/SVN commit messages, or bug reports), specific pre-processing steps to re- move irrelevant details from the source, (e.g., CVS/SVN
commit number, source code punctuation or language keywords) is performed, split identifiers, and, finally, normalize the resulting text using stop-word removal and stemming.

REQUIREMENTS AND SOURCE CODE.

Histrace first processes source code files to extract all the identifiers and comments from each class. Histrace then uses underscore and the Camel Case convention [3] to split identifiers into terms, thus producing for each class a separate document.
Histrace then performs the following steps to normalize requirements and source code documents:

1. Converting all upper-case letters into lower-case and removing punctuation.
2. removing all stop words (such as articles, numbers, and so on);
3. Performing word stemming using the Porter Stemmer, bringing back inflected forms to their morphemes.

CVS/SVN COMMIT MESSAGES

To build Histracecommits, Histrace first extracts CVS/SVN commit logs and excludes those that

1. Are tagged as "delete" because they concern classes that have been deleted from the system and thus cannot take part in any traceability link.
2. Do not concern source code (e.g., if a commit only contains HTML or PNG files).
3. Have messages of length shorter or equal to one English word, because such short messages would not have enough semantics to participate in creating $R2CT_i$.

Histrace then extracts the CVS/SVN commit messages as well as the classes that

1. Are still part of the source code.

2. Source code has been part of the commits. Histrace applies the same normalization steps on CVS/SVN commit messages as those for requirements and source code.

BUG REPORTS.

To build $Histrace_{bugs}$, Histrace extracts all the bug reports from a bug-tracking system. Usually, bug reports donot contain explicit information about the source code files that developers updated to fix a bug. All the details about the updated, deleted, or added source code files are stored in some CVS/SVN repository. Therefore, Histrace must link CVS/SVN commit messages to bug reports before being able to exploit bug reports for traceability. Histrace uses regular expressions, i.e., a simple text matching approach but with reasonable results, to link CVS/SVN commit messages to bug reports. However, Histrace could also use more complex techniques.

Consequently, Histrace assumes that developers assigned to each bug a unique ID that is a sequence of digits recognizable via regular expressions. The same ID must be referred to by developers in the CVS/SVN commit messages.

Then, to link CVS/SVN commit messages to bug reports concretely, Histrace performs the following steps:

1. Extracts all CVS/SVN commit messages, along with commit status and committed files.
2. extracts all the bug reports, along with time/date and textual descriptions, and
3. Links each CVS/SVN commit message and bug reports.

3.3 TRUMO

Trumo assumes that different experts $Histrace_{commits}$ (also known as $R2CT_1$) and $Histrace_{bugs}$ ($R2CT_2$) know useful information to discard or re-rank the traceability links between two documents, e.g., requirements and source code classes. Trumo is thus similar to a Web model of the users' trust: the more users buy from a Web merchant, the higher the users' trust of this merchant.

Trumo combines the requirement traceability links obtained from an IR technique and discards/reranks them using an expert's opinions and a trust model inspired by Web-trust models. It compares the similarity of the recovered links with those provided by the experts and with the number of times that the link appears in each expert's set. It is not tied to any specific IR-based traceability-recovery approach and can use any expert's opinion to adjust the ranking of recovered links. For example, the experts can be Histrace commits and–or Histrace bugs, taking advantage of CVS/SVN repositories and bug-tracking systems.

3.4 DYNWING

To automatically decide the weights $\lambda_i(r_j , c_s)$ for each expert, we apply a dynamic weighting technique. Existing techniques [2], [8] to define weights use static weights for all the experts. Using the same static weight may not be beneficial for all the recovered links. Therefore, consider each link recovered by a baseline IR technique and by the different experts as an independent link and dynamically assign weights to baseline links and each expert. Choosing the right weight per link is a problem that we formulate as a maximization problem. Basically, we have different experts, i.e., CVS/SVN commits, bug reports, and others, to trust a link. Each expert has its own trust into the link. By maximizing the similarity value DynWing automatically identifies the experts that are most trustworthy and those that are less trustworthy.

DynWing computes and assigns weights to the experts in the trust model dynamically, i.e., on a per link basis. Dynamic weighting techniques are promising to assign weights for each link. DynWing analyses each expert's similarity value for each link and assigns weights according to these values.

IV. GOAL

The goal of is to study the accuracy of Trustrace when recovering traceability links against that of a single IR technique, JSM and VSM, using requirements, source code, CVS/SVN commits, and–or bug reports as experts. The quality focus is the accuracy of Trustrace in terms of precision and recall. It is also the improvement brought by the dynamic weighting technique, DynWing, with respect to a PCA based technique in

terms of F1 score. The main goal is to recover traceability links with greater precision and recall values than that of currently available traceability recovery IR based techniques.

V. DISCUSSION AND CONCLUSION

Trustrace has a better accuracy. Other factors that can impact the accuracy of traceability-recovery approaches. One of these factors is quality of source code identifiers. If there is a low similarity between the identifiers used by requirements and source code, then no matter how good an IR-based technique is, it would not yield results with high precision and recall values. To analyse the quality of the identifiers in our datasets and measure their similarity values which helps to identify poor semantic areas in source code. Compute the similarity value between the set of requirements R, all merged into a single document and the set of classes C all merged into the normalized term-by-document matrix to avoid any effect from the document lengths. Then, we use JSM and VSM to compute the similarity. The similarity between these source code and requirements shows how close two documents are in terms of semantics.

Trustrace is the topic which involves source code and requirements and software repositories it is done in order to provide consistency and accuracy to the links obtained. Previous methods implement JSM and VSM model but the main disadvantage of these models are poor accuracy and they requires manual pages and UML diagrams which has probability of low accuracy and leads to low precision and recall values. So by implementing software repositories to support traceability process which improves precision and recall values. It creates baseline links by mining source code document; it also implements two trust models namely DynWing and Trumo which is used to rank experts and also to find the trustable links with respect to experts correspondingly. The proposed system mines software repositories in addition to source code, which leads to high precision and recall values. It combines the mined results with IR techniques and achieves higher precision and recall values.

REFERENCES

[1.] O. C. Z. Gotel and C. W. Finkelstein, "An analysis of the requirements traceability problem," Requirements Engineering., Proceedings of the First International Conference on, pp. 94-101, April 1994.

[2.] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trust-based requirements traceability," in Proceedings of the 19th International Conference on Program Comprehension, S. E. Sim and F. Ricca, Eds. IEEE Computer Society Press, June 2011, 10 pages.

[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, vol. 28, no. 10, pp. 970-983, 2002.

[4.] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in Proceedings of 25th International Conference on Software Engineering. Portland Oregon USA: IEEE CS Press, 2003, pp. 125-135.

[5.] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, and Howard, "Helping analysts trace requirements: An objective look," in RE '04: Proceedings of the Requirements Engineering Conference, 12th IEEE International. Washington, DC, USA: IEEE Computer Society, 2004, pp. 249-259.

[6.] J. I. Maletic and M. L. Collard, "Tql: A query language to support traceability," in TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering. Washington, DC, USA: IEEE Computer Society, 2009, pp. 16-20.

[7.] J. H. Hayes, G. Antoniol, and Y.-G. Guéhéneuc, "Prereqir: Recovering pre-requirements via cluster analysis," in Reverse Engineering, 2008. WCRE '08. 15th Working Conference on, Oct 2008, pp. 165-174.

[8.] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," IEEE Transactions on Software Engineering, vol. 33, no. 6, pp. 420-432, 2007.

[9.] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in Software Maintenance (ICSM), 2011 27th IEEE International Conference on, sept. 2011, pp. 133 -142.

[10.] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, Factors Impacting the Inputs of Traceability Recovery Approaches, A. Zisman, J. Cleland-Huang, and O. Gotel, Eds. New York: Springer-Verlag, 2011.