Trace-by-Classification: A Machine Learning Approach to Generate Trace Links for Frequently Occurring Software Artifacts

Mateusz Wieloch, Sorawit Amornborvornwong, Jane Cleland-Huang
School of Computing
DePaul University
Chicago, IL, 60604, USA
mateuszwieloch@outlook.com, sorawitamorn@windowslive.com, jhuang@cs.depaul.edu

***Abstract*—Over the past decade the traceability research community has focused upon developing and improving trace retrieval techniques in order to retrieve trace links between a source artifact, such as a requirement, and set of target artifacts, such as a set of java classes. In this Trace Challenge paper we present a previously published technique that uses machine learning to trace software artifacts that recur is similar forms across across multiple projects. Examples include quality concerns related to non-functional requirements such as security, performance, and usability; regulatory codes that are applied across multiple systems; and architectural-decisions that are found in many different solutions. The purpose of this paper is to release a publicly available TraceLab experiment including reusable and modifiable components as well as associated datasets, and to establish baseline results that would encourage further experimentation.**

***Index Terms*—traceability, machine learning, challenge**

I. Introduction

This paper presents *Trace by Classification* (TBC), a machine learning approach in which a classifier is trained to generate trace links for requirements and/or other kinds of software artifacts which occur relatively frequently across different projects. Unlike more traditional trace retrieval techniques, such as the Vector Space Model (VSM) [12], probabilistic approaches [5], [9], Latent Semantic Indexing [2], [8], [10] and Latent Dirichlet Allocation (LDA) [3], [14] in which similarity scores are computed between a *trace query* and each document in a set of target artifacts, the classification approach learns a set of weighted indicator terms that can then be used to classify target documents into specific types. Where a sufficient training set is available, this approach tends to outperform the basic trace retrieval techniques.

The *TBC* technique described in this paper was originally presented in our prior work on detecting and classifying non-functional requirements (NFR) [6], [7]. The classifier has since been used to trace regulatory codes to requirements [4], [11] and to trace quality concerns via architectural tactics to code [15]. The purpose of this paper is to describe the approach, implement the (TBC) tool as a reusable TraceLab workflow, reproduce previous results from the original NFR experiments, and release the datasets, components, and functioning experiments for public dissemination. TBC can be downloaded from http://coest.org/mt/14/150.

II. Traceability Challenge

The traceability challenge involves creating a technique that returns high-precision results for tracing commonly occurring source artifacts to a set of project-specific target artifacts. We present results from using our approach in two specific applications, namely (1) tracing quality goals such as "security", "performance", and "usability" to project-level requirements across a variety of projects, and (2) tracing HIPAA (Health Insurance Portability and Accountability Act (HIPAA) of 1996) technical safeguards to product level requirements in health care related products.

This work specifically addresses the *Ubiquitous* goal of the Grand Challenges of Traceability which states that "Near zero (or acceptable) stakeholder effort is required to establish and make use of traceability, with no (or minimum) impact on their primary task," and specifically research task 2 which focuses on "Total automation of (or one-click) traceability creation and trace maintenance, with quality and performance levels superior to manual efforts."

Our solution, TBC, leverages the fact that because NFRs and certain regulatory standards occur frequently across projects, there is value in investing the time and effort needed to train a classifier to recognize them. Our prior applications of this approach [4], [6], [7], [11], [15] have shown that TBC returns significant improvements over standard tracing approaches such as VSM for tracing frequently occurring requirements.

III. Trace by Classification

TBC takes as input a training set of manually classified artifacts. For example, in Table I we show a small selection of requirements classified by quality type; while in Table II we show a selection of requirements that address the automatic logoff requirement stipulated in the US Health Insurance Portability and Accountability Act (HIPAA). This, and several other technical safeguards, must be addressed in all software systems which handle a patient's personal

978-1-4799-0495-2/13/$31.00 © 2013 IEEE

TEFSE 2013, San Francisco, CA, USA

TABLE I
SAMPLE REQUIREMENTS CLASSIFIED BY QUALITY TYPE

Requirement	Type
The system shall display the local and exercise time in separate clocks	Functional
The system shall offer the ability to pause and resume the refresh of data.	Functional
The application shall match the color of the schema set forth by Department of Homeland Security	Look-and-feel
The product shall be available during normal business hours. As long as the user has access to the client PC, the system will be available 99If projected, the data must be understandable. On a 10'x10' projection screen, 90The product shall ensure that it can only be accessed by authorized users. The product will be able to distinguish between authorized and unauthorized users in all access attempts	Security
The product shall be intuitive and self-explanatory. : 90The product shall respond fast to keep up-to-date data in the display.	Performance
The CMA report shall be returned no later 60 seconds after the user has entered the CMA report criteria.	Performance
The system shall be capable of processing 100The product shall be able to process 10,000 transactions per hour within two years of its launch. This number will increase to 20,000 by Release 2.	Scalability

TABLE II
SAMPLE REQUIREMENTS FROM ACROSS MULTIPLES SYSTEMS THAT ADDRESS THE HIPAA TECHNICAL SAFEGUARD TO "IMPLEMENT ELECTRONIC PROCEDURES THAT TERMINATE AN ELECTRONIC SESSION AFTER A PREDETERMINED TIME OF INACTIVITY"

Automatic timeout can be specified by location.
Automatic timeout can be specified by role.
Electronic session must terminate after a pre-determined period of inactivity.
System has a timer allowing automatic logoff
System will implement session time outs and use cookies to terminate an electronic session
The system must time out if the user has been away from the system for some time.
The system shall ask the user if the user wants to continue using the system before timing out.
The system shall timeout after a period of inactivity.
The system upon detection of inactivity of an interactive session shall prevent further viewing and access to the system by that session by terminating the session, or by initiating a session lock that remains in effect until the user reestablish the connection.
When the system timeouts, the user is returned to the login page to sign in again.

healthcare information. Both of these datasets provide training data needed by our approach.

A. Preprocessing

Before training the classifier, all data in the training set (and subsequently all artifacts that are are to be traced using the *TBC* approach) are preprocessed to eliminate common stop words that do not provide any relevant information on the documents lexical content (for example conjunctions and prepositions),and to reduce the remaining words to their morphological roots.

B. Training Phase

During the training phase a set of indicator terms is identified for each NFR category. This step assumes the existence of a set of correctly pre-classified requirements that can be used for training. The requirements in the training set are used to compute a probabilistic weight for each potential indicator term in respect to each NFR type. The weight measures how strongly an indicator term represents a specific type (i.e. in the provided examples, either an NFR type or a HIPAA technical safeguard).

For example, terms such as "authenticate" and "access" that occur frequently in security requirements and infrequently in other types of requirements, represent strong indicator terms for security NFRs, while other terms such as "ensure" that occur less frequently in security requirements or are found in several different requirement types, represent much weaker indicator terms.

The formula for computing weighted indicator terms was described in our prior work [6], [7] and is repeated here for explanatory purposes.

Let Q represent a requirement type (e.g. a specific NFR type, such as *security* or *performance*, or a HIPAA regulatory code such as the previously described *Automated Logoff*). Indicator terms of type Q are found by considering the set S_Q of all type Q NFRs in the training set. The cardinality of S_Q is defined as N_Q. The frequency $freq(d_{Q,i}, t)$ of occurrence of term t in document $d_{Q,i}$ is computed for each document in S_Q. Each term is assigned a weight score that measures how well the term helps identify a requirement of type Q. So for each type Q and keyword t, the weight score $Pr_Q(t)$ is defined as a probability value computed as follows:

$$Pr_Q(t) = \frac{1}{N_Q} \sum_{i=1}^{N_Q} \frac{freq(d_{Q,i}, t)}{|d_{Q,i}|} \cdot \frac{N_Q(t)}{N(t)} \cdot \frac{NP_Q(t)}{NP_Q} \quad (1)$$

The first factor $\frac{1}{N_Q} \sum_{i=1}^{N_Q} \frac{freq(d_{Q,i},t)}{|d_{Q,i}|}$ in the expression above computes the average frequency of occurrences of term t in type Q artifacts normalized over the document's size $d_{Q,i}$. The factor increases if a term appears more frequently in type Q documents, and therefore it can be considered as a potential indicator term for type Q artifacts. The second factor $\frac{N_Q(t)}{N(t)}$ is the percentage of Q type documents in S_Q containing t with respect to all requirements in the training set containing t, whose number is denoted by $N(t)$. This factor decreases if the indicator term t is used broadly throughout the requirements specification. If the term is only used in Q type requirements, it will evaluate to 1 for that type. The third factor $\frac{NP_Q(t)}{NP_Q}$ is the ratio between the number of $NP_Q(t)$ of system projects containing type Q documents with term r and the number NP_Q of all projects in the training set with type Q artifacts.

The purpose of this rescaling factor is to decrease the weight PrQ(t) for terms that are project specific, and to increase the weight for terms that appear in several projects containing type Q documents. For each term t, a probability score $Pr_Q(t)$ is computed. Terms are ranked by decreasing order according to $Pr_Q(t)$. The top K terms are identified as indicator terms for the type Q.

C. Classification Phase

Once indicator terms are mined and weighted, they can be used in a second step to classify additional artifacts. A probability value that represents the likelihood that the new requirement belongs to a certain artifact type is computed as a function of the occurrence of indicator terms of that type in

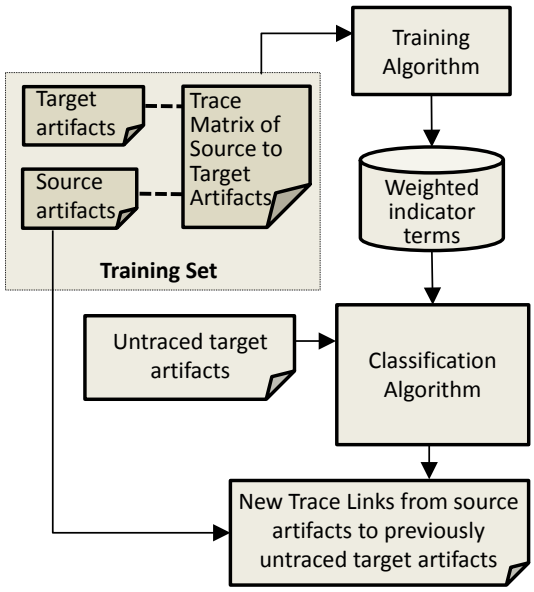

Fig. 1. The Trace by Classification Process (modified from [6], [7]

the requirement. A requirement is then classified according to a certain artifact type if it contains several indicator terms representative of that type. Requirements receiving classification scores above a certain threshold for a given type will be classified into that type, and all unclassified requirements will be assumed to be functional requirements. Because classification results can only be considered successful if a high percentage of the target artifacts are detected for a specific type, in all our experiments the threshold is established with the objective of achieving high recall results.

Artifacts are classified by computing a probability score $PrQ(R)$ that estimates the probability that a certain artifact R is classified as type Q. Let I_Q be the set of indicator terms for a quality type Q. We assume that the weighted indicator terms in I_Q are identified and their weights computed from a training set that contains correctly pre-categorized artifacts. The indicator terms are mined using the expression in Formula (1). The classification score that an unclassified artifact R belongs to a type Q is defined as follows:

$$Pr_Q(R) = \frac{\sum_{t \in RcapI_Q} Pr_Q(t)}{\sum_{t \in I_Q} PR_Q(t)} \quad (2)$$

The numerator is computed as the sum of the term weights of all type Q indicator terms that are contained in R, and the denominator is the sum of the term weights for all type Q indicator terms. The probabilistic classifier for a given type Q will assign higher score $Pr_Q(R)$ to an artifact R that contains several strong indicator terms for Q.

IV. Experimental Design

We evaluated our approach against datasets from two different domains. The first dataset represented 1449 requirements written by Masters Students in a Requirements Engineering Courses at DePaul university. Students each chose a project, elicited requirements from real stakeholders, and then specified part of the system. Given the classroom environment there is a higher than normal ratio of non-functional requirements and to functional ones. 30 projects are included, covering a total of 24 distinct NFR types. The second dataset includes a total of 10 HIPAA technical safeguards, traced to a total of 1891 requirements across ten different health-care projects.

Experiments were conducted to evaluate TBC using a standard leave-one-out cross validation technique. In each case the data was divided into 10 equally sized buckets. Nine buckets were used for training purposes to produce indicator terms and the remaining bucket was used for training. In previous experiments we considered three techniques for selecting indicator terms. These included top k terms, all terms, and all terms above a given threshold t. The optimal results were realized through use of a threshold, and therefore in this paper we select all terms weighted at scores greater than t. The computed indicator terms were then used to classify artifacts in the remaining bucket. This process was repeated 10 times until artifacts in each bucket had been classified once.

In previous experiments we considered two alternate approaches for classification purposes. The first approach, which was found to underperform, simply selected the top scoring type as the type for each requirement (i.e. Functional, Security, etc). The second approach, which was found to return better results, used a multiple classification scheme in which each artifact was classified according to any type for which it scored over a predefined threshold value. For example if a given requirement R scored above the threshold for both *security* and *performance* it was classified as both types, even though in our dataset each artifact belonged to one, and only one type. This approach would however, be useful if requirements were non-atomic and did in fact represent more than one type at the same time.

V. TraceLab Implementation

This paper does not present any new algorithms or experiments. Its contribution is purely to release TBC into the public domain and to summarize the results reported in our prior work in the form of a baseline. The complete experiment including all components for importing data, training the classifier, classifying artifacts, conducting the leave-one-out cross validation experiment, and displaying results in a confusion matrix are therefore released for public use. The workflow of our experiment is shown in Figure 2. The experiment proceeds as follows. First a set of preclassified requirements and a list of stopwords are imported. The requirements are then preprocessed to remove stopwords and to stem them to their morphological roots. The preprocessor component shown in Figure 2 is shown with sharp corners, meaning that it is actually a composite component made up of a number of lower level parts. Once the data is proprocessed, the *Scatter To Buckets* component randomly divides the data into n equally sized parts (referred to as buckets). The main experimental loop is then executed. In each iteration, one bucket is selected for testing (classification) and the remaining

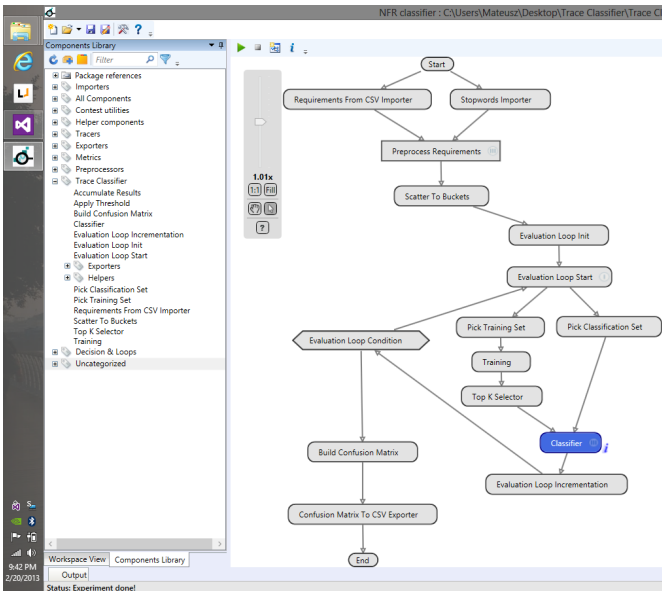

Fig. 2. TraceLab Workflow for the Trace By Classification experiment

buckets for training. The classifier is then trained, a set of indicator terms created, and the the top k selected. These terms are then used to classify the test set. The looping continues until each bucket has been tested once. The *Classifier* saves its results so that once the loop is exited, the data is available to build a confusion matrix. This is then exported to an external csv file.

The TraceLab package implementing TCB can be downloaded from http://coest.org/mt/14/150. To execute the TraceLab experiment it is also necessary to download and install TraceLab (also available for free from http://coest.org).

VI. BaseLine Results

Results, generated by the TraceLab experiment, are displayed and analyzed using a confusion matrix for the NFRs in Figure 3 and for the HIPAA technical safeguards in Figure 4. These results are fully reproducible through executing the provided TraceLab experiment. The confusion matrix presents results in a visual format and depicts both true and false positives as well as true and false negatives. The quality types (i.e. security, performance etc) are shown both as rows and columns, and the true positives which are depicted on the diagonal have been highlighted in the diagram. The NFR-type matrix in Figure 3 shows, for example, that the availability requirement (labeled "A") had 37 instances in the dataset. Of these, 32 were correctly classified as availability, 9 were incorrectly classified as scalability (SC), and 1 as maintenance (MN) etc. The confusion matrix allows us to compute several key metrics including recall, precision, F2-measure, and specificity, each of which is defined below.

Recall measures the fraction of artifacts of each type that are correctly classified:

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|} \quad (3)$$

Precision measures the fraction of retrieved links that are relevant and is computed as:

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|} \quad (4)$$

The F2-measure computes the harmonic mean of recall and precision weighted towards recall. This measure is commonly used to evaluate experiments in the traceability domain where high recall values are essential. The F2 Measure is computed as follows (where $\beta = 2$):

$$F_\beta Measure = \frac{(1+\beta^2) * Precision * Recall}{\beta^2 * Precision) + Recall} \quad (5)$$

Finally, specificity measures the True positive rate and is computed as:

$$Specificity = \frac{|TrueNegative|}{|TrueNegative| + |FalsePositive|} \quad (6)$$

The results shown in the two confusion matrices suggest that when a sufficiently large sized training set is available, the classification approach is generally able to achieve high recall and precision. However, the results are also impacted by a second factor which is discussed in our prior work [4], [6], [7], [11]. This factor is related to the homogeneity of artifacts assigned to each type. If a specific type is homogenous in nature, then it is easier to train the classifier. In contrast, artifacts from categories such as IC (Integrity Controls) are more heterogenous in nature and therefore require a far larger training set in order to be effectively classified.

VII. Conclusion

This paper presented the challenge to develop traceability techniques that improve the accuracy achieved for tracing requirements which occur frequently across multiple projects and/or domains. In fact, following the publication of our initial paper on the NFR-classifier [6] we made the dataset containing the original 15 student projects available to the PROMISE repository [1]. A team of students in Dr. Tim Menzies data mining course at the University of West Virginia, conducted an extensive set of experiments utilizing WEKA to compare standard classification techniques against the performance of the NFR-classifier algorithms. The results from naive bayes classifier, standard decision tree algorithm (J48), feature subset selection (FSS), correlation-based feature subset selection (CFS), and various combinations of the above were evaluated; however they were unable to globally improve on the results obtained by the NFR classifier [13]. The combined approach with decision trees and feature subset selection procedure in Jalaili et al. produced the best recall and precision values among the standard machine learning tools used in the paper. Its recall value was still lower than the one achieved by the NFR classifier, while the precision values were similar.

The purpose of creating a research challenges is to present results and then to stand back and expect other researchers to improve them. We therefore encourage researchers to address this challenge and to post their published results to http://coest.org/mt/14/150.

Type	Initial	FT	SC	MN	O	PE	A	US	SE	LF	L	F	CP	I	P	Recall	Precision	Specificity	F2-Measure
FT	10	**1**	0	2	5	0	0	4	3	0	0	3	1	0	0	0.100	0.048	0.985	0.082
SC	28	1	**19**	4	11	5	6	12	10	0	0	16	16	0	1	0.679	0.311	0.969	0.549
MN	47	5	7	**24**	20	1	13	17	8	1	0	19	7	1	0	0.511	0.348	0.966	0.467
O	95	9	2	15	**62**	4	10	31	21	11	0	57	9	3	10	0.653	0.106	0.587	0.321
PE	76	1	2	0	27	**50**	22	27	28	4	0	42	5	0	6	0.658	0.350	0.928	0.559
A	37	0	9	1	18	14	**32**	11	7	0	0	18	6	0	0	0.865	0.186	0.895	0.500
US	106	0	1	4	42	13	13	**88**	57	3	0	75	7	4	7	0.830	0.166	0.648	0.461
SE	109	1	2	4	43	3	4	62	**79**	2	0	81	4	2	20	0.725	0.171	0.695	0.440
LF	66	1	0	1	24	0	0	25	20	**36**	9	30	2	1	0	0.545	0.486	0.971	0.533
L	28	1	0	0	7	0	0	2	2	7	**21**	8	1	0	6	0.750	0.636	0.991	0.724
F	731	1	14	14	307	47	67	240	209	9	0	**438**	28	2	53	0.599	0.541	0.414	0.587
CP	11	0	5	0	8	3	1	6	5	0	0	9	**1**	0	1	0.091	0.011	0.934	0.037
I	11	0	0	0	6	3	4	1	5	0	0	7	2	**3**	3	0.273	0.176	0.990	0.246
P	11	0	0	0	7	0	0	5	8	1	3	7	1	1	**9**	0.818	0.078	0.921	0.281

Fig. 3. Confusion Matrix Showing Results for the Classification of Non-Functional Requirements where FT=Fault Tolerance, SC=Scalability, MN=Maintenance, O=Operationality, PE=Performance, A=Availability, US=Usability, SE=Security, LF=Look-and-feel, L=Legal,F=Non-Classified, I=Integrity, and P=Portability

Type	Initial	AC	PA	AUD	AL	F	IC	UUI	Recall	Precision	Specificity	F2-Measure
AC	54	**47**	6	8	0	50	0	4	0.870	0.109	0.788	0.362
PA	36	16	**28**	4	0	30	0	3	0.778	0.491	0.984	0.697
AUD	84	31	7	**79**	2	73	4	4	0.940	0.274	0.883	0.633
AL	10	5	1	2	**7**	7	0	1	0.700	0.583	0.997	0.673
F	1660	328	13	187	3	**1485**	51	67	0.895	0.891	0.142	0.894
IC	17	2	1	5	0	13	**3**	4	0.176	0.052	0.970	0.119
UUI	10	4	1	3	0	8	0	**7**	0.700	0.078	0.955	0.269

Fig. 4. Confusion Matrix Showing Results for the Classification of HIPAA Technical Safeguards where AC=Access Control, PA=Person/Entity Authentication, AUD=Audit Control, AL=Automatic Logoff, F=Non-Classified, IC=Integrity Controls, and UUI=Unique User Identification

ACKNOWLEDGMENTS

This work was supported by USA National Science Foundation grants CCF-0959924 and CCF-1265178.

REFERENCES

[1] PROMISE Software Engineering Repository. http://promise.site.uottawa.ca/SERepository/. [Online; accessed 1-April-2013].

[2] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002.

[3] H. U. Asuncion, A. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *International Conference on Software Engineering*, pages 95–104, 2010.

[4] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *ICSE (1)*, pages 155–164, 2010.

[5] J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *RE*, pages 135–144, 2005.

[6] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *RE*, pages 36–45, 2006.

[7] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc. Automated classification of non-functional requirements. *Requir. Eng.*, 12(2):103–120, 2007.

[8] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. Enhancing an artefact management system with traceability recovery features. In *ICSM '04: Proceedings of the 20th IEEE International Conference on Software Maintenance*, pages 306–315, Washington, DC, USA, 2004. IEEE Computer Society.

[9] C. Duan and J. Cleland-Huang. Clustering support for automated tracing. In *ASE*, pages 244–253, 2007.

[10] M. Eaddy, A. V. Aho, G. Antoniol, and Y.-G. Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *ICPC*, pages 53–62, 2008.

[11] M. Gibiec, A. Czauderna, and J. Cleland-Huang. Towards mining replacement queries for hard-to-retrieve traces. In *ASE*, pages 245–254, 2010.

[12] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Softw. Eng.*, 32(1):4–19, 2006.

[13] A. Jalaji, R. Goff, J. Jackson, N. Jones, and T. Menzies. Making sense of text: identifying non functional requirements early. In *West Virginia University CSEE technical report*, 2006.

[14] A. Marcus, J. I. Maletic, and A. Sergeyev. Recovery of traceability links between software documentation and source code. *Intn'l Journal of Software Engineering and Knowledge Engineering*, 15(5):811–836, 2005.

[15] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Çinar. A tactic-centric approach for automating traceability of quality concerns. In *ICSE*, pages 639–649, 2012.