When and How Using Structural Information to Improve IR-based Traceability Recovery

Annibale Panichella[1], Collin McMillan[2], Evan Moritz[4], Davide Palmieri[3],
Rocco Oliveto[3], Denys Poshyvanyk[4], Andrea De Lucia[1]
[1]*University of Salerno, Fisciano (SA), Italy*
[2]*University of Notre Dame, Notre Dame, USA*
[3]*University of Molise, Pesche (IS), Italy*
[4]*The College of William and Mary, Williamsburg, USA*
apanichella@unisa.it, cmc@nd.edu, eamoritz@cs.wm.edu, davice22it@gmail.com,
rocco.oliveto@unimol.it, denys@cs.wm.edu, adelucia@unisa.it

***Abstract*—Traceability recovery is a key software maintenance activity in which software engineers extract the relationships among software artifacts. Information Retrieval (IR) has been widely accepted as a method for automated traceability recovery based on the textual similarity among the software artifacts. However, a notorious difficulty for IR-based methods is that artifacts may be related even if they are not textually similar. A growing body of work addresses this challenge by combining IR-based methods with structural information from source code. Unfortunately, the accuracy of such methods is highly dependent on the IR methods. If IR methods perform poorly, the combined approaches may perform even worse.**

In this paper, we propose to use the feedback provided by software engineers when classifying candidate links to regulate the effect of using structural information. Specifically, our approach only considers structural information when the traceability links from the IR methods are verified by developers and classified as correct links. An empirical evaluation conducted on three systems suggests that our approach outperforms both a pure IR-based method and a simple approach for combining textual and structural information.

***Keywords*-Traceability Link Recovery, Empirical studies.**

I. Introduction

Traceability links are a valuable resource during software maintenance because they provide a connection from high-level software documents such as use cases to low-level implementation details, such as source code and test cases [3]. Unfortunately, traceability links are notoriously difficult to extract from software [3], [13], [28]. Software engineers must read and understand different artifacts to determine whether a link exists between two artifacts. Meanwhile, the artifacts are constantly being modified in the midst of an evolving software system. Maintaining a list of up-to-date traceability links inevitably becomes an overwhelming, error-prone task. Semi-automated tools for traceability recovery offer an opportunity to reduce this manual effort and increase productivity.

Information Retrieval (IR) [5] has gained wide-spread acceptance as a method for automating traceability recovery [3], [13], [21], [28]. The IR-based methods, such as those based on Vector Space Model (VSM) [5] or probabilistic Jensen and Shannon (JS) model [1], identify traceability links using the textual information from the software artifacts. For example, the keywords from documents describing use cases may match keywords in the comments of a source code file. Textual information has the advantage of being widely available, but it is unfortunately also highly subjective. Words may have multiple meanings, identifiers from software are often misleading if taken out of the context, and comments are frequently out of date [2]. Different strategies have been successful in improving IR-based methods, including text pre-processing (e.g., [37], [39]), smoothing filters [12], and combinations of these approaches [17]. Nevertheless, imprecision remains a major barrier to using IR for traceability link recovery in practice.

Structural information contained in source code (e.g., function calls or inheritance relationships) has been proposed in solutions to increase the precision of IR-based traceability recovery [31]. In general, a combined approach will use an IR-based method to locate a set of candidate links, and then either augment or filter the set of links based on the structural information. However, combined approaches tend to be sensitive to a given IR method. If the candidate links are correct, then the structural information can help locate additional correct links. Otherwise, the structural information offers little help, or will even pollute the results with incorrect links.

Our conjecture is that the traceability links recovered by IR-methods should be verified by software engineers prior to expanding the set of links with structural information. For example, consider the use case `Authorize User` from a hospital management system. The system contains a class called `User`, and there is a traceability link between the use case `Authorize User` and the class. This link is likely to be located by IR methods because of shared keywords such as "user." However, there are other links, which the IR methods do not recognize. The use case is also relevant to the class `Doctor`, which extends `User`. If a software engineer classifies the link from the class `User` to the use case `Authorize User` as correct, then we recommend a link from the use case to the class `Doctor`, because of the inheritance relationship between `Doctor` and `User`.

On the other hand, consider the use case `Bill User`. A link exists between this use case and the class `Patient`. However, because of the keyword matches, an IR method would locate a link between the use case and the class `User` in addition to `Patient` (because of the keyword "user"). Yet, this new link would be a false positive. A canonical application of structural information would aggravate the situation by recommending a link to `Doctor`, based on the link to `User`. In our approach, the links recovered via IR techniques are verified incrementally by software developers before using the structural information. As a result, only the correct link to `Patient` class would be identified.

In this paper, we present our approach to using feedback and structural information to improve IR-based traceability link recovery. We use IR methods to locate candidate links, which are certified by programmers. Then, we apply a "bonus" to additional links to artifacts, which are connected via the structural information. We present an empirical evaluation in which we found that our approach outperforms two IR-based methods as well as a naive combined approach by a statistically significant margin. We also found that adding a bonus to correct links is more effective than applying a penalty to incorrect links. The results in this paper are consistent across three different systems and provide practical guidelines to using programmer feedback for traceability recovery.

Paper structure. Section II provides background information on IR methods and traceability recovery. Section III presents the proposed approach, while Section IV reports the design and the results of the empirical evaluation of the proposed approach. Finally, Section V concludes the paper.

II. Background and Related work

This section provides a background on Information Retrieval methods as well as a survey of the related work.

A. *Information Retrieval Techniques*

Information Retrieval (IR) refers to a type of technique used to compute textual similarities among different documents. The textual similarity is computed using the occurrence of terms in the documents. If two documents share a large number of terms, those documents are considered to be similar. Different IR methods have been proposed, but in general they can be summarized in three steps [5]. First, a *corpus* is built from the documents after pre-processing such as stop word removal and stemming. Next, each document is represented as an entry in an index. One common index is the *term-by-document matrix*, where each row represents a document and each column is a term. The values in the matrix indicate the frequency of the term occurring in the document. Finally, the similarity among the index entries is calculated using a formula such as the cosine distance [28]. The representation of the index entries and the formula for calculating the similarity vary depending on the IR method. We use two different IR methods in this paper, and briefly describe each in the following paragraphs.

In the Vector Space Model (VSM) [5] each document is represented as a vector of terms. Each row in the term-by-document matrix can be considered as one document's vector in the space of terms that occur in all documents. The similarity of two documents is calculated based on the angle between each document's vector. For example, the similarity may be the cosine of the angle between the vectors. In general, the angle between two document vectors will decrease as the documents share more terms. Thus, the similarity of the documents will be higher.

The Jenson-Shannon (JS) model [1] differs from VSM in that each document is represented as a probability distribution. The distribution is based on the probability of a term occurring in each document. In other words, each document has some probability of containing each term. For example, the distribution can be derived from each column in the term-by-document matrix, where we have first normalized the columns of the matrix. The similarity of the documents is calculated from the Jenson-Shannon divergence (e.g., the "distance" between the probability distributions).

B. *IR-based Traceability Recovery Approaches*

Traceability recovery has long been recognized as a major maintenance task in software engineering [19]. Promising results have been achieved using IR methods (e.g., [3]), since pairs of source-target artifacts having higher textual similarities have a high probability to be linked.

These techniques extract textual similarities from high-level artifacts, such as requirements documents, to the implementation artifacts, such as source code. The terms from source code are typically the identifier names and comments. Unfortunately, there is a synonymy problem when using these terms, in that the authors of the high-level documents may not use the same terms to describe the same topics as the programmers writing source code. Efforts to address the synonymy problem relied on using advanced IR methods such as Latent Semantic Indexing (LSI) [15]. Despite these advancements, no single IR method has been shown to consistently have the highest performance for traceability [1], [17], [33].

Different improvements have been proposed to the application of the IR methods. Some improvements focus on the terms that are extracted from the artifacts, and some pre-processing techniques, such as identifier splitting, are now widely-accepted [16]. Capobianco *et al.* have suggested that domain-specific terms (e.g., jargon) best describe the concepts in the code. They propose to use only the nouns from the software artifacts [9]. Other work has also adapted

Algorithm 1 Optimistic Combination of Structural and Textual Information — O-CSTI($List$, $G(C,E)$, δ)

```
1: $i \leftarrow 1$
2: while not (end of $List$) do
3:     Get the link $(s, c_j)$ in position $i$ of $List$
4:     for all $c_t \in C$ do
5:         if $(c_j, c_t) \in E$ then
6:             $Sim(s, c_t) \leftarrow Sim(s, c_t) + \delta * Sim(s, c_t)$
7:         end if
8:     end for
9:     $i \leftarrow i + 1$
10: end while
11: Reorder $List$
12: The user classifies the links in $List$
```

the weights of the artifacts' terms depending on the length of the artifacts [37], a project glossary [39], or external dictionaries [20]. Recently, smoothing filters have been shown to improve the precision of IR-based traceability [12]. In addition to these technical improvements, other related work has concentrated on human factors in traceability, such as how to help programmers understand how to use the links for a specific task [11], [22], [26], [34].

Structural information has been shown to augment IR-based methods for traceability. The structural information generally refers to the relationships in the software's source code, such as function calls, inheritance, or realization relationships. These relationships are valuable for traceability recovery because the links among the source code are reflected as links among high-level artifacts (an idea known as *software reflexion* [32]). Approaches combining IR and structural information have been used to compute similarity among software artifacts in single software systems [8], [18], [23], [27], [31], [35], [38] and repositories of source code [6], [25], [29], [30]. While these approaches have substantially increased the performance of traceability recovery methods, they still rely on the accuracy of the IR method.

Our work differs from existing methods in that we propose to filter the links recovered by IR methods using feedback provided by software engineers. Relevance feedback has been suggested as an improvement to IR-based methods before [14], [21]. These approaches ask developers to classify links as correct or incorrect, and based on the classifications, modify the weights of the terms in the artifacts. Our work instead proposes to combine the advantages of using feedback with the increased accuracy from structural information.

III. Integrating Structural Information in IR-based Traceability Recovery

We conjecture that the structural links in source code are *transitive* for the purpose of traceability recovery. If a structural link exists between two source code artifacts, and a textual link (from an IR method) exists between a documentation artifact and one of the source code artifacts, then we conjecture that a link exists from the documentation artifact to both source code artifacts. In practice, an approach combining IR and structural information detects the transitive relationships and assigns a *bonus* to the textual similarity of those relationships.

Previously proposed approaches (e.g., [31]) use an IR method to locate a set of initial links, and then extend that set using the structural information. The software developer evaluates and classifies the final set of links only at the end. Formally, let $G(C,E)$ be the indirect graph of relationships between code classes, where $C = \{c_1, \ldots, c_n\}$ is the set of code classes and $E = \{(c_i, c_j),\ \textit{there is a relationship between } c_i \textit{ and } c_j\}$ is the set of indirect edges. Moreover, let S be the set of source artifacts (e.g. use cases) and let $List = \{(s, c),\ s \in S \textit{ and } c \in C\}$ be the list of candidate links computed by using an IR method. Starting from the link (s, c_1) in the first position of $List$, the similarity between all the pairs $(s, c_j) \in List$ such that $(c_1, c_j) \in E$ is increased by a adding bonus δ (constant or variable). The same process is applied to all the links in $List$. Once the list of candidate links is recomputed and reordered, the software engineer analyzes the candidate links and determines those that are correct links, and those that are false positives. In the following we refer to this approach as O-CSTI, *Optimistic Combination of Structural and Textual Information* (see Algorithm 1). O-CSTI is identical to that proposed by McMillan et. al [31] except that we use a different IR method to locate the initial links. We compare O-CSTI to our new approach in Section IV.

The approach proposed in this paper works by iterating through the list of links derived by using the IR method. At each iteration, we provide the first link from the ranked list to a software engineer. The software engineer classifies this link either as correct link or as false positive. Then, we recommend new links based on structural information if and only if the link is classified as correct. The ranked list is reordered and the classified links are not shown anymore in the following iterations while the new top links are classified by the engineer. The process is stopped when all the correct links are retrieved or when the number of false positives becomes too high if compared to the number of correct links (and thus the software engineer decides to stop the traceability recovery process [13]). We refer to this approach as UD-CSTI, *User-Driven Combination of Structural and Textual Information* (see Algorithm 2).

It is important to highlight that the proposed approach does not require recomputing textual similarities between each pair of source and target artifacts because our approach does not modify the initial term-by-document matrix. In other words, unlike the IR-based relevance feedback [36], our approach does not apply any re-weighting of terms within the software artifacts.

A crucial input for both approaches is the choice of the bonus δ. The simplest way to define the bonus is to fix it as a

Algorithm 2 User-Driven Combination of Structural and Textual Information — UD-CSTI($List,\ G(C,E),\ \delta$))

```
1:  while not (stopping criterion) do
2:      Get the link $(s, c_j)$ on top of $List$
3:      The user classifies $(s, c_j)$
4:      if $(s, c_j)$ is correct then
5:          for all $c_t \in C$ do
6:              if $(c_j, c_t) \in E$ then
7:                  $Sim(s, c_t) \leftarrow Sim(s, c_t) + \delta * Sim(s, c_t)$
8:              end if
9:          end for
10:     end if
11:     Reorder $List$
12:     Hide links already classified
13: end while
```

constant value in the range of [0, 1], i.e., the range of values of textual similarities. Alternatively, the similarity value can be increased by adding a bonus that is a percentage of the actual similarity value (variable bonus). However, defining a (constant or variable) bonus *a priori* is quite difficult and oftentimes, subjective. In fact, the size (measured as the difference between the max and min similarity values) of the ranked list can sensibly differ from one system to another, or when tracing different types of artifacts. In particular, there might be very concentrated ranked list, where all the similarity values are thicken in a small interval of values. On the other hand, there might be scattered ranked list, where the similarity values are highly spread in the interval [0, 1]. In the former case, a small bonus is enough, while in the latter case a quite high bonus is required.

For this reason, we propose an adaptive bonus that is proportional to the median variability of the similarity values computed for each software artifact. More precisely, we set the adaptive bonus as $\delta = median\{v_i, \ldots, v_n\}$ where a generic v_i value denotes the *variability* of the similarity values the $i-th$ artifact, i.e. $v_i = (max_i - min_i)/2$ where max_i and min_i are the maximum and minimum similarity values the $i-th$ source artifacts.

The benefits of the proposed approach are as the following. With respect to the constant bonus, the proposed approach takes into account the variability of the ranked list. The variability of the ranked list is also taken into account by the variable bonus. Nevertheless, the adaptive bonus is superior to the variable bonus because the former does not require providing any input to the software engineer, that is, the bonus definition is completely automatic. In Section IV we empirically analyze the benefits of such a heuristic with respect to this variable bonus.

IV. Empirical Evaluation

In this section we describe in detail the design of a case study carried out to evaluate the proposed approach. The description of the study follows the Goal−Question−Metric guidelines [7].

Table I
Characteristics of the systems used in the case study.

System	Description	KLOC	Source Artifact (#)	Target Artifact (#)	Correct links
EasyClinic	A system used to manage a doctor's office	20	UC (30)	CC (37)	93
			UML (20)	CC (37)	69
			TC (63)	CC (37)	204
eTour	An electronic touristic guide developed by students.	45	UC (58)	CC (174)	366
SMOS	A system used to monitor high school students	23	UC (67)	CC (100)	1,044

UC: Use case, TC: Test case, CC: Code class

A. Definition and Context

The *goal* of the experiment was to analyze whether the accuracy of IR-based traceability recovery methods improves when combining the textual similarity computed by the IR method with structural information. We also verified whether it is worthwhile to exploit classification of candidate links by software engineers by expanding the set of links with structural information.

The *context* of our study was represented by three software systems, namely EasyClinic, eTour, and SMOS. All the systems have been developed by final year Master's students at the University of Salerno (Italy). Use cases and code classes are available for eTour and SMOS, while for EasyClinic we use those two types of artifacts, as well as the descriptions of UML diagrams and test case scenarios.

Table I shows the characteristics of the considered software systems in terms of type and number of source and target artifacts, as well as Kilo Lines of Code (KLOC). The natural language of the artifacts for all the systems is Italian. The table also reports the number of links between different types of source and target artifacts. Such information (that we used as an oracle to evaluate the accuracy of the proposed traceability recovery methods) was derived from the traceability matrix provided by the original developers.

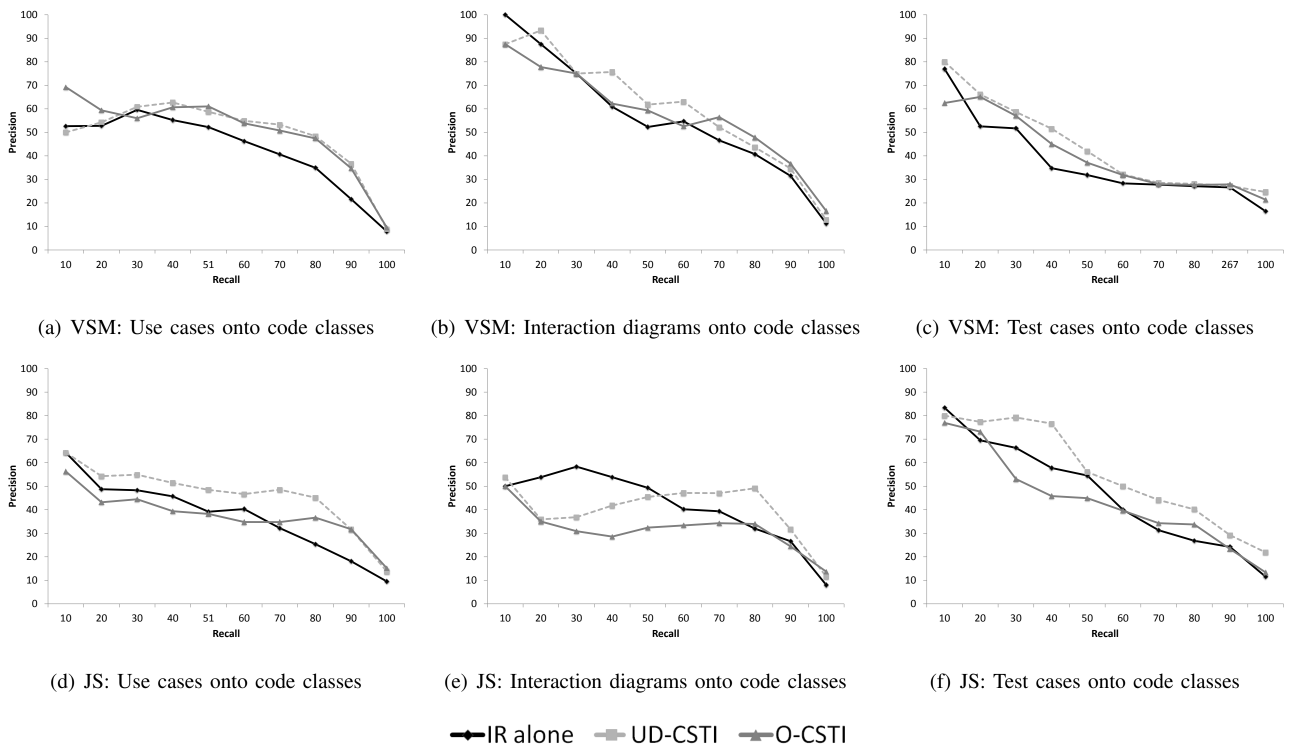
B. Research Questions

In the context of our study, we formulated the following research questions:

- **RQ**$_1$: *Can structural information be used to complement textual information and improve IR-based traceability recovery?*
- **RQ**$_2$: *Can we improve the use of structural information by filtering the IR-based links with feedback from software developers?*

To respond to our first research question, we compared the accuracy of UD-CSTI with the accuracy obtained by using canonical IR-based traceability recovery methods. For the second research question, we compared UD-CSTI with O-CSTI (see Section III).

To increase the generalizability of our results, we recovered traceability links between different types of software



(a) VSM: Use cases onto code classes (b) VSM: Interaction diagrams onto code classes (c) VSM: Test cases onto code classes

(d) JS: Use cases onto code classes (e) JS: Interaction diagrams onto code classes (f) JS: Test cases onto code classes

Figure 1. Precision/Recall curves achieved on the EasyClinic repository.

documentation artifacts and source code of the three different software systems. In addition, we analyzed the benefits of the proposed approach when using two different IR methods, namely the JS method and VSM (see Section II). To eliminate any bias towards either IR method, we used identical term-by-document matrices.

C. Metrics

To evaluate the different traceability recovery methods, we use two well-known IR metrics [5]:

$$recall = \frac{|cor \cap ret|}{|cor|}\% \qquad precision = \frac{|cor \cap ret|}{|ret|}\%$$

where cor represents the sets of correct links and ret is the set of links retrieved by the traceability recovery technique.

A common way to evaluate the performance of retrieval methods consists of comparing the precision values obtained at different recall levels. This result is a set of recall/precision points which are displayed in precision/recall graphs. In order to provide a single value that summarizes the performance, we use the *average precision*, that can be defined as the mean of the precision scores obtained for each correct link [5]. It can be mathematically expressed as

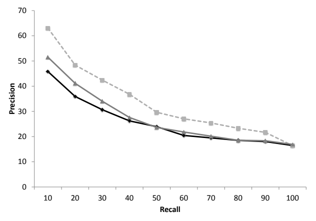
$$AP = \frac{\sum_{i=1}^{n} x_i \; prec_i}{\sum_{i=1}^{n} x_i}$$

where x_i represents the binary correctness of $i-th$ link (i.e. $x_i = 1$ if the $i-th$ is correct; $x_i = 0$ otherwise) while $prec_i$ denotes its precision value.

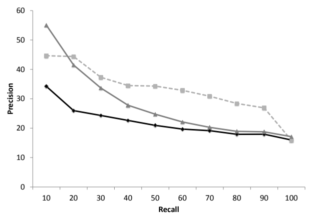
D. Analysis of the Results

This section discusses the results of our experiments, aimed at answering two research questions stated in Section IV. Figures 1 and 2 compare the precision/recall curves summarizing the performance of (i) the proposed approach (UD-CSTI), (ii) the canonical approach to combine structural and textual information (O-CSTI), and (iii) the pure IR-based traceability recovery. The results are grouped by software systems and IR methods.

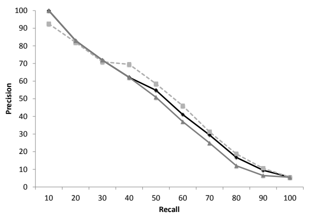
The results achieved indicate that structural information is useful to improve the performances of pure VSM- and JS-based traceability recovery methods. In addition, UD-CSTI generally outperforms O-CSTI, indicating that the classification of the software engineer is worthwhile to consider in order to regulate the effect of structural information. Specifically, for recall values lower than 100% the UD-CSTI provides an improvement—compared to the other methods—in terms of precision ranging between 10% and 20%. This result corresponds to a substantial reduction in false positives, ranging between 75% and 20%. From the software engineer's point of view who has to inspect the ranked list of candidate traceability links, this results represent a substantial improvement since such a drastic



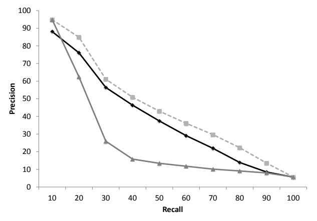
(a) Using VSM on SMOS



(b) Using JS on SMOS



(c) Using VSM on eTour



(d) Using JS on eTour

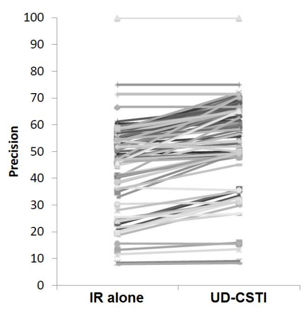
◆ IR alone ■ UD-CSTI ▲ O-CSTI

Figure 2. Precision/Recall curves achieved on SMOS and eTour.

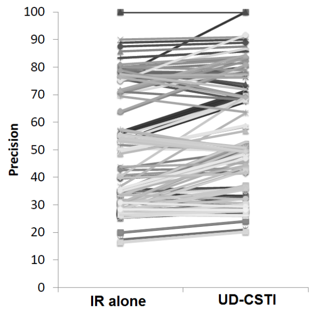
reduction of false positives mirrors a drastic reduction of time needed for the manual classification. Indeed, the number or false positives measures the waste of manual effort required before reaching a given recall value. For example, when tracing use cases to code classes of eTour using a pure JS-based traceability recovery method we can retrieve 314 correct links (about 85% of recall level) discarding 3,007 false positives (i.e., with a precision value about 9.5%). With UD-CSTI, we can obtain 17.2% precision for the same level of recall, discarding only 1,505 false positives (i.e., with a false positives reduction about 50%). Thus, the software engineer is able to retrieve the 85% of all the correct links saving the 50% of time she should uselessly spent for analyze irrelevant links when using the other methods.

When the goal is to recover all correct links (100% of recall), it is possible to achieve an improvement of the accuracy only in a few cases, i.e., when tracing UML diagrams onto code classes and when tracing test cases onto code classes for EasyClinic. For the other traceability activities, tracing all the correct links requires the same effort—intended as number of false positives to be discarded—for all the presented traceability techniques. This is probably due to the low textual similarity of the links in the lower part of the ranked list. If the textual similarity of the correct link is very small, then the usage of the bonus does not allow the link to effectively "climb" in the ranked list. For example, the last correct link on eTour has a textual similarity equivalent to 0.03%. When applying the structural feedback its textual similarity will be increased by only 0.007%, enabling the possibility for this link to increase its ranking.

Such a scenario is confirmed by the relation diagrams showed in Figure 3, that compares the ranking of each correct links in (i) a ranked list build using only VSM and (ii) a ranked list build using VSM with UD-CSTI. Each point in the diagram represents a correct link in the ranked list. Thus, the relation diagram is particularly useful to graphically measure the effect of UD-CSTI on the ranked list of candidate links. As we can see, structural information is generally useful to increase the ranking of correct links, which results in a better precision as compared



(a) Use cases onto code classes



(b) Test cases onto code classes

Figure 3. Relation diagram achieved on EasyClinic using VSM.

to a canonical stand-alone IR-based traceability recovery method. However, when the textual similarity between the two artifacts is too low, it is quite difficult to noticeably increase the similarity. In this case, we retrieve the links with the same precision.

Table II reports a typical scenario where the usage of the structural feedback allows to improve the recovery of correct links. More precisely, consider the link between the use case `Insert Laboratory Data` and the class `GUIData` extracted from EasyClinic. This link is correctly retrieved by the IR method since these artifacts have a high textual similarity (i.e., 42%). The use case is also relevant to the class `Laboratory`, which has a usage relationship with the class `GUIData`. If a software engineer classifies the link between the class `GUIData` and the use case `Insert Laboratory Data` as correct, then we increase the similarity between the use case and the class `Laboratory`, because of the usage relationships between the two classes. In addition, since the classes `GUIData` and `Laboratory` are both structurally related to the class `GUILaboratoryData` by usage relationships, the link between use case `Insert Laboratory Data` and the class `GUILaboratoryData` is increased as well as applying a bonus two times (one for each relationship). Thus, such a link—that turns to be a cor-

Table II
USEFULNESS OF STRUCTURAL INFORMATION ON EASYCLINIC.

Use Case	Code Class	Correct	Similarity	N. Bonus	Similarity + Bonus (10%)
Insert Laboratory Data	GUIData	YES	42%	0	42%
	Laboratory	YES	39%	1	43%
	GUILaboratoryData	YES	36%	2	41%
	LaboratoryManager	YES	27%	3	35%

Table III
DIFFERENCES BETWEEN UD-CSTI AND O-CSTI IN PRESENCE OF FALSE POSITIVES.

Use Case	Code Class	Correct	Sim.	UD-CSTI		O-CSTI	
				N. Bonus	Bonus (10%)	N. Bonus	Bonus (10%)
Modifiy Visit	GUIExamination	NO	49%	0	-	0	-
	Examination	NO	46%	0	-	1	+5%
	GUIExaminationResults	NO	31%	0	-	2	+6%

rect link—is able to "climb" the ranked list because its similarity values are increased by 5%. In the same way, the similarity between `Insert Laboratory Data` and `LaboratoryManager` is increased because of the usage relationships between `LaboratoryManager` and `GUIData`, `Laboratory` and `GUILaboratoryData`.

The previous example explains how the structural information can effectively improve the recovery of correct links. If the candidate link is correct, then the structural information helps to locate additional correct links. Unlike O-CSTI, the proposed approach is empirically shown to be very stable even when candidate links are false positives. Consider the scenario reported in Table III where the structural information is combined with textual similarity of false positives. Consider the link between the use case `Modify Visit` and the class `GUIExamination.java` extracted from EasyClinic. This link is retrieved by the IR method because these two artifacts have 49% textual similarity. However, this link is a false positive. Moreover, the class `GUIExamination` has an usage relationship with the class `Examination` and an inheritance relationship with the class `GUIExaminationResults`. If a software engineer classifies the link between the class `GUIExamination` and the use case `Modify Visit` as false positives, then the approach UD-CSTI does not change the similarity values of the candidate links. Differently, the O-CSTI will erroneously increase the similarity between the use case and the classes `Examination` and `GUIExaminationResults`, because of the structural relationships between them and the class `GUIExamination`. For instance, the textual similarity (computed using VSM) between `Modify Visit` and `Examination` will be increased by 5% allowing such false positive link to "climb" the ranked list. The same happens for the link between `Modify Visit` and `GUIExaminationResults`. Such a situation will have a negative impact on the accuracy of the traceability recovery method, since two false positives have increased their ranking in the list of candidate links.

E. Threats to validity

External validity concerns the generalization of these findings. An important threat is related to the repositories used in the empirical study, i.e. EasyClinic, SMOS and eTour. They are software projects implemented by the final year Master's students at the University of Salerno (Italy), thus they are hardly comparable to real industrial projects. However, they are comparable to repositories used by other researchers [3], [21], [28] and both EasyClinic and ETour have been used as benchmark repositories in the last two editions of the traceability recovery challenge organized at TEFSE'09 and TEFSE'11. In addition, EasyClinic was also used by other authors to evaluate the accuracy of IR-based traceability recovery methods [4]. Nevertheless, we are planning to replicate the experiment using other artefact repositories in order to corroborate our findings.

With the aim of making the achieved results more generalizable we recovered traceability links between different types of software documentation artifacts (i.e. use cases, UML diagrams and test cases) and source code on EasyClinic. In addition, we analyzed the benefits of the proposed approach when structural information is combined with textual information derived by using both vector space and probabilistic models, namely VSM and JS respectively. Our approach seems to be able to produce better results regardless of the underlying type of artifacts being traced and IR method.

Concerning the threats to the construct validity, we adopted two widely used metrics for assessing IR techniques, namely recall and precision. Moreover, the number of false positives retrieved by a traceability recovery tool for each correct link retrieved reflects well its retrieval accuracy.

Concerning the internal validity, several factors may affect our results. First, choosing the right values of bonus is a critical issue. We propose to use an adaptive bonus that is proportional to the size of the ranked list. Since other heuristic might be used to define a bonus, we compared the results achieved with an adaptive bonus with those achieved using a variable bonus (see Section III). We experimented with different percentage values of the variable bonus,

Table IV
COMPARISON BETWEEN THE AVERAGE PRECISION VALUES ACHIEVED USING VARIABLE AND ADAPTIVE BONUS.

System	Activity	Method	b=10%	b=20%	b=30%	b=40%	b=50%	b=60%	b=70%	b=80%	b=90%	b=100%	Adaptive Bonus
EasyClinic	UC→CC	VSM	53.54	52.63	51.27	50.31	49.23	49.11	48.53	48.12	47.80	46.58	**53.69**
		JS	**49.98**	48.29	47.20	46.19	46.19	46.07	46.07	46.07	46.07	46.07	49.65
	ID→CC	VSM	63.65	61.31	58.07	54.80	52.69	51.68	51.53	51.17	51.17	50.75	**64.44**
		JS	45.46	42.61	41.80	40.53	40.53	40.53	40.53	40.53	40.53	40.53	**49.13**
	TC→CC	VSM	**45.98**	43.98	41.88	40.20	39.83	39.67	39.48	39.15	38.41	38.40	45.79
		JS	55.13	**58.69**	50.91	49.59	49.59	49.59	49.59	49.59	49.59	49.59	57.40
SMOS	UC→CC	VSM	31.88	34.34	35.98	**36.89**	36.37	36.63	36.29	35.99	35.70	28.96	36.27
		JS	35.26	35.91	35.29	35.22	35.23	35.23	35.23	35.23	35.23	35.23	**36.08**
eTour	UC→CC	VSM	53.01	**53.40**	53.08	52.58	52.13	51.59	51.16	50.82	50.76	50.37	53.22
		JS	48.88	43.28	37.13	32.91	30.10	28.50	28.13	28.04	28.04	28.04	**49.03**

Table V
COMPARISON BETWEEN AVERAGE PRECISION VALUES ACHIEVED USING UD-CSTI (WITH MALUS) AND IR ALONE

System	Activity	Method	UD-CSTI	IR alone
EasyClinic	UC→CC	VSM	39.32	45.83
		JS	31.39	41.76
	ID→CC	VSM	53.45	59.40
		JS	43.64	46.50
	TC→CC	VSM	37.50	41.84
		JS	50.52	50.89
SMOS	UC→CC	VSM	29.17	28.96
		JS	23.30	23.29
eTour	UC→CC	VSM	50.87	51.68
		JS	38.97	42.19

ranging from 10% to 100% with a step of 10%. Table IV reports the average precision achieved by considering different heuristics and different percentage for the variable bonus. We decided to use the average precision since it returns a single value for each experimented method and facilitates the comparison [5]. As we can see, the adaptive bonus provides generally good results and frees the software engineer from tuning up the parameter required for the variable bonus (i.e., percentage of similarity).

Another important factor to be considered relates to the possibility of using not only a "bonus" for correct links, but also a "malus" for false positives. Specifically, when the software engineer classifies a candidate link between a documentation artifact and a class as a false positive, the structural information can be used to reduce (instead of increase) the textual similarity between the same documentation artifact and other structurally related classes. We experimented with such a variant of the proposed approach by using, also in this case, an adaptive and a variable bonus. Table V compares the best results (in terms of average precision) achieved with UD-CSTI and those achieved with IR alone. As we can see, this variant of UD-CSTI was not able to improve the performance of a pure IR-based recovery method, and often the obtained results were even worse.

F. TraceLab Implementation

We also implemented the experiment using the TraceLab framework [24], [10], which allowed us to use existing tools to quickly design and set up our approach. We extended the framework by designing two new components – which correspond to the two models used in the evaluation, O-CSTI and UD-CSTI – using the TraceLab software development kit (SDK) in C#.NET.

The two new components take a previously computed rank-list of similarities from an IR model and a set of relationships between code classes, which we implement as a similarity matrix of pairs with a score of 1. Additionally, the UD-CSTI component takes into account developer feedback. We use the oracle to simulate the process of selecting true or false positives described in in Algorithm 2.

The overall TraceLab implementation is designed in the following manner (Figure 4): a setup component reads the experiment meta-information and stores it to the Workspace. The experiment then enters the main loop, which iterates over each dataset under study. At the beginning of each iteration, an artifacts importer stores the source artifacts, target artifacts, oracle, code relationships, and dataset-specific stopwords to the Workspace. After preprocessing the artifacts, the two IR models, VSM and JS, are run. The similarity rank-lists from these models are then input to the O-CSTI and UD-CSTI components. Each model's results are analyzed and their metrics are collected into a single data structure and stored into the Workspace. Once each dataset has been analyzed, the collective data structure is converted into a format for viewing in a GUI component. Further components can be added to the graph to save the results to disk for offline viewing. The graph pictured here has been slightly truncated to preserve space while conveying the overall structure of the experiment.

The results of the TraceLab implementation are similar to those shown in the evaluation. The differences can be accounted for by slight variations in the way TraceLab's components compute their outputs, such as preprocessing or IR model internals. We provide the experiments and datasets for download at `http://www.cs.wm.edu/semeru/data/csmr13/`. We welcome the readers to reproduce and build on top of our results by rerunning our experiments in TraceLab.

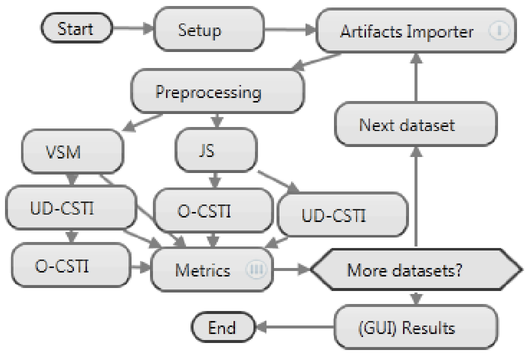


Figure 4. TraceLab experiment implementation

V. Conclusions and Future Work

Structural information contained in source code (e.g., function calls or inheritance relationships) has been repeatedly proposed in solutions to increase precision of IR-based traceability recovery [23], [27], [30]. However, such approaches are too optimistic, assuming that structural links in source code are always *transitive* for the purpose of traceability recovery. In this paper, we proposed an approach that considers structural information when the traceability links from IR methods are verified by the software developers and classified as correct links. An empirical evaluation conducted on three software systems suggests that our approach outperforms both stand-alone IR-based methods and an optimistic approach for combining textual and structural information.

Future work will be devoted to replicating our study in a larger context to corroborate these findings. Moreover, there are some potential improvements to our approach that extend into two areas. First, we plan to experiment with other heuristics for defining the bonus. While we used an adaptive bonus in this paper, it is possible that different formulae for choosing the bonus will further increase the performance of the proposed technique. The second area of improvement is related to the combination of the proposed approach with user feedback analysis [36] and other enhancing strategies such as smoothing filters [12]).

VI. Acknowledgements

This work is supported in part by NSF CCF-1016868, CNS-0959924, and NSF CCF-1218129 awards. Any opinions, findings and conclusions expressed here are the authors' and do not necessarily reflect those of the sponsors.

References

[1] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proceedings of the 16th IEEE International Conference on Program Comprehension*, 2008, pp. 103–112.

[2] N. Anquetil and T. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative research*, 1998, p. 4.

[3] G. Antoniol, G. Canfora, G. Casazza, A. De 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] H. U. Asuncion, A. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, 2010.

[5] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[6] S. K. Bajracharya, J. Ossher, and C. V. Lopes, "Leveraging usage similarity for effective retrieval of examples in code repositories," in *Proceedings of the 18th International Symposium on Foundations of Software Engineering*, 2010, pp. 157–166.

[7] V. Basili, G. Caldiera, and D. H. Rombach, *The Goal Question Metric Paradigm*. John Wiley and Sons, 1994.

[8] D. Binkley, N. Gold, M. Harman, Z. Li, and K. Mahdavi, "An empirical study of the relationship between the concepts expressed in source code and dependence," *Journal of Systems and Software*, vol. 81, no. 12, pp. 2287–2298, 2008.

[9] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the role of the nouns in IR-based traceability recovery," in *Proceedings of the 17th International Conference on Program Comprehension*, 2009, pp. 148–157.

[10] J. Cleland-Huang, A. Czauderna, A. Dekhtyar, O. Gotel, J. H. Hayes, E. Keenan, G. Leach, J. Maletic, D. Poshyvanyk, Y. Shin, A. Zisman, G. Antoniol, B. Berenbach, A. Egyed, and P. Maeder, "Grand challenges, benchmarks, and Tracelab: developing infrastructure for the software traceability research community," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*. New York, NY, USA: ACM, 2011, pp. 17–23.

[11] D. Cuddeback, A. Dekhtyar, and J. Hayes, "Automated requirements traceability: The study of human analysts," in *Proceedings of the 18th International Requirements Engineering Conference*, 2010, pp. 231 –240.

[12] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Improving ir-based traceability recovery using smoothing filters," in *Proceedings of the 19th International Conference on Program Comprehension*, 2011, pp. 21–30.

[13] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Transactions on Software Engineering and Methodology*, vol. 16, no. 4, p. 13, 2007.

[14] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 299–309.

[15] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[16] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *Proceedings of the 19th IEEE International Conference on Program Comprehension*, 2011, pp. 11–20.

[17] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, 2011, pp. 133–142.

[18] N. Gold, M. Harman, Z. Li, and K. Mahdavi, "Allowing overlapping boundaries in source code using a search based approach to concept binding," in *Proceedings of the 22nd International Conference on Software Maintenance*, 2006, pp. 310–319.

[19] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proc. of 1st International Conference on Requirements Engineering*, 1994, pp. 94–101.

[20] J. H. Hayes, A. Dekhtyar, and J. Osborne, "Improving requirements tracing via information retrieval," in *Proceedings of the 11th IEEE International Requirements Engineering Conference*, 2003, pp. 138–147.

[21] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods." *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.

[22] J. H. Hayes, A. Dekhtyar, S. Sundaram, and S. Howard, "Helping analysts trace requirements: an objective look," in *Proceedings of the 12th International Requirements Engineering Conference*, 2004, pp. 249–259.

[23] E. Hill, L. Pollock, and K. Vijay-Shanker, "Exploring the neighborhood with dora to expedite software maintenance," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007, pp. 14–23.

[24] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. H. Hayes, A. Dekhtyar, D. Manukian, S. Hussein, and D. Hearn, "Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions," in *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering*, 2012.

[25] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi, "Sourcerer: mining and searching internet-scale software repositories," *Data Mining and Knowledge Discovery*, vol. 18, pp. 300–336, 2009.

[26] P. Mader, O. Gotel, and I. Philippow, "Motivation matters in the traceability trenches," in *Proceedings of the 17th IEEE International Requirements Engineering Conference*, 2009, pp. 143–148.

[27] J. I. Maletic and A. Marcus, "Supporting program comprehension using semantic and structural information," in *Proceedings of the 23rd International Conference on Software Engineering*, 2001, pp. 103–112.

[28] 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*, 2003, pp. 125–135.

[29] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie, "Exemplar: A source code search engine for finding highly relevant applications," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.

[30] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usage," in *Proceedings of the 33rd International Conference on Software Engineering*, 2011, pp. 111–120.

[31] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Proceedings of the International Workshop on Traceability in Emerging Forms of Software Engineering*, 2009, pp. 41–48.

[32] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: Bridging the gap between design and implementation," *IEEE Transactions on Software Engineering*, vol. 27, no. 4, pp. 364–380, 2001.

[33] R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia, "On the equivalence of information retrieval methods for automated traceability link recovery," in *Proceedings of the 18th International Conference on Program Comprehension*, 2010, pp. 68–71.

[34] M. C. Panis, "Successful deployment of requirements traceability in a commercial engineering organization...really," in *Proceedings of the 18th IEEE International Requirements Engineering Conference*, 2010, pp. 303–307.

[35] M. P. Robillard, "Automatic generation of suggestions for program investigation," in *Proceedings of the 10th European Software Engineering Conference*, 2005, pp. 11–20.

[36] J. J. Rocchio, *The SMART Retrieval System – Experiments in Automatic Document Processing*. Prentice Hall, Inc., 1971, ch. Relevance feedback in information retrieval, pp. 313–323.

[37] R. Settimi, J. Cleland-Huang, O. Ben Khadra, J. Mody, W. Lukasik, and C. De Palma, "Supporting software evolution through dynamically retrieving traces to UML artifacts," in *Proceedings of 7th IEEE International Workshop on Principles of Software Evolution*, 2004, pp. 49–54.

[38] D. Shepherd, Z. Fry, E. Gibson, L. Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *Proceedings of the 6th International Conference on Aspect Oriented Software Development*, 2007, pp. 212–224.

[39] X. Zou, R. Settimi, and J. Cleland-Huang, "Term-based enhancement factors for improving automated requirement trace retrieval," in *Proceedings of International Symposium on Grand Challenges in Traceability*, 2007, pp. 40–45.