Leveraging Historical Co-change Information for Requirements Traceability

Nasir Ali[1], Fehmi Jaafar[2], and Ahmed E. Hassan[1]
[1]Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Canada
[2]LBIT Team, DIRO, Université de Montréal, Canada
{nasir,ahmed}@cs.queensu.ca, jaafarfe@iro.umontreal.ca

***Abstract*—Requirements traceability (RT) links requirements to the corresponding source code entities, which implement them. Information Retrieval (IR) based RT links recovery approaches are often used to automatically recover RT links. However, such approaches exhibit low accuracy, in terms of precision, recall, and ranking. This paper presents an approach (CoChaIR), complementary to existing IR-based RT links recovery approaches. CoChaIR leverages historical co-change information of files to improve the accuracy of IR-based RT links recovery approaches. We evaluated the effectiveness of CoChaIR on three datasets, *i.e.*, iTrust, Pooka, and SIP Communicator. We compared CoChaIR with two different IR-based RT links recovery approaches, *i.e.*, vector space model and Jensen–Shannon divergence model. Our study results show that CoChaIR significantly improves precision and recall by up to 12.38% and 5.67% respectively; while decreasing the rank of true positive links by up to 48% and reducing false positive links by up to 44%.**

***Index Terms*—Traceability, requirements, source code, co-change, information retrieval**

I. Introduction

Requirements traceability (RT) links requirements and source code entities to ensure that all the requirements have been implemented by the developers and that source code is consistent with the documentation. During software maintenance and evolution tasks, developers add, remove, and modify software systems to meet ever-changing users' requirements. Consequently, RT links between requirements and source code become obsolete. Manually creating RT links is an effort intensive task. Information Retrieval (IR) techniques are often used to automatically recover RT links with various accuracy [1]. IR-based RT approaches heavily depend on textual information to create RT links. Due to poor textual similarity and noise in textual information [2], IR-based RT approaches often create many false positive links.

In this paper, we conjecture that adding extra non-textual based information, *i.e.*, co-change information of files [3], [4], could improve the accuracy of IR-based RT approaches. Our conjecture stems from the previous work by Kagdi *et al.* [5], [6] that if two or more files were frequently co-changed together (in source code repository) for a longer period of time; they may have a conceptual link between them and may belong to the same requirement/concept. We extend their work by linking requirements (evolving outside the source code repository) to source code files instead of only linking source code file with the source code repository. To test our conjecture, we propose an approach that combines CO-CHAnge information of source code files with IR techniques (CoChaIR) to improve the accuracy of IR-based RT approaches.

To evaluate the effectiveness of CoChaIR, we perform an empirical study using CoChaIR, Jensen-Shannon model (JSM), and Vector Space Model (VSM) on three datasets, *i.e.*, iTrust, Pooka, and SIP. We use precision, recall, and ranking metrics to measure the improvement brought by our CoChaIR approach over JSM and VSM. We address the following two research questions in our empirical study:

RQ1: Does CoChaIR lead to better accuracy?

CoChaIR improves the precision and recall of JSM and VSM by up to 12.38% and 5.67% respectively. CoChaIR removes up to 44% of the false positive links. In all the cases (*i.e.*, when comparing our approach to baseline approaches like JSM and VSM across the three studied datasets), we observed an improvements over the baseline IR techniques. When examining the effect size [7] of such improvements, we noted a large effect size in 92% of the cases. This is a very promising improvement over our prior work [8]. Our prior work made use only of textual information. It did not result in improvement in all cases, when run on the same studied datasets. For the cases, that it did show improvement, only 66% of these cases had a large effect size.

RQ2: Does CoChaIR provide better ranking?

CoChaIR decreases the rank of true positive links by up to 48%. CoChaIR uses the number of co-changes for each class to rank true positive links higher in the ranked list.

A careful analysis of our datasets and results show that the use of non-textual information (as done by CoChaIR) lead to higher effect on improvement than our previous work [8] due to the fact that almost 30% of the change messages were empty message or had less than three words. Moreover in a few cases, many of the change log messages were not informative, *e.g.*, "fixed some bugs in this revision". In short, the low quality of the textual content, *e.g.*, change log messages, directly impacts the accuracy, *i.e.*, lower recall than baseline, of any textual based approach (*e.g.*, our prior work [8]).

The remainder of the paper is organized as follows: Section II provides a brief description of the state-of-the-art IR techniques and co-change approaches. Section III describes the proposed approach in details and sketches our implementation. Section IV provides details on its empirical study and the results. Sections VI and VII provide discussion on the results,

and threats to the validity of our findings. Finally, Section VIII concludes while highlighting avenues for future work.

II. Related Work

RT links recovery and feature location as well as historical co-change information topics are related to our research work.

A. Traceability and Feature Location Approaches

Several researchers (*e.g.*, [1], [9], [10], [11]) have used IR techniques to perform traceability recovery and feature location. Often, IR-based traceability [9] and feature location [11] approaches, use VSMs, probabilistic rankings, or a VSM transformed using latent semantic indexing (LSI) to create traceability links. Comparisons have been made among different IR techniques, *e.g.*, [1], [12], with inconclusive results. On several datasets, the VSM and JSM perform favourably in comparison to more complex techniques, such as LSI [1] or latent dirichlet allocation (LDA) [12]. Yet, algebraic models, *e.g.*, the VSM [9] and probabilistic model, *e.g.*, the JSM model [1], have become a reference baseline for both feature location [11], [13] and traceability recovery [9], [14].

IR techniques use textual information of high-level documents, *e.g.*, requirements, and low-level documents, *e.g.*, source code, to create traceability links between them[9]. Unfortunately, there is a problem of conceptual distance between requirements and source code. For example, a developer may not use the same terms as used in requirements. Consequently, this conceptual distance will lead to low textual similarity and poor accuracy of IR techniques.

Different improvements have been proposed for IR techniques. Some of these improvements focus on the different weighting schemes of terms [1], and some pre-processing techniques of documents, such as splitting source code identifiers [15]. Structural information of source code has shown to supplement IR techniques [16], [17], [18]. The structural information of source code generally refers to the function calls, inheritance, aggregation, or association. However, due to some anti-patterns [19] and–or source code quality [20] issues, structure information could be misleading. For example, a controller class design smell controls various functionality and–or calls of various classes [19]. In the presence of controller class design smell, the structure information may not be accurate. Whenever available dynamic data [11] proved to be complementary and useful for traceability recovery by reducing the search space. However, such as dynamic data is resource consuming and does not always possible to collect execution traces of a system [2]. Software repositories also have been mined to uncover traceability links [6], [8].

IR techniques face textual similarity problems due to noisy textual information [2]. Ali *et al.* [8] linked requirements to bug reports and SVN/CVS logs to improve the accuracy of IR techniques. However, their proposed improvement also depends on textual information, which faces the same problem as IR techniques, *i.e.*, noise in textual information. In some cases, due to textual noise, their approach performed worst than baseline IR techniques. In addition, the authors mentioned most of the SVN/CVS logs messages did not have any textual information. Therefore, their approach could not work in such cases. In this paper, we do not use textual information to improve the accuracy of IR-based RT approaches.

To the best of our knowledge, none of the previous work studied the impact of historical co-change information on RT link recovery using IR techniques. The work presented in this paper is complementary to existing IR-based RT approaches and propose improvement to IR based RT approaches. CoCHaIR uses current state-of-the-art IR techniques to create baseline links, then uses historical co-change information to recalculate the textual similarity, and tries to discard false positive links. Our proposed approach can be used as a filtering layer on all the previously mentioned IR-based RT approaches.

B. Co-change Analysis Approaches

Researchers [3], [21], [22] have proposed approaches to analyse co-changing files. They conjectured that two files are in a co-change relationship if they frequently changed together in past. This co-change information of files could help in software maintenance tasks [3], [4]. Co-change information of source code files help developers to examine potential files linked to maintenance task at hand. Ying *et al.* [22] and Zimmermann *et al.* [3] applied association rules to identify co-changing files. Their hypothesis is that past co-changed files can be used to recommend source code files potentially relevant to a change request. Fehmi *et al.* [4] proposed an approach Macocha to detect co-changes. Their approach uses KNN algorithm to group change and bit vectors to analyse co-changing files. Their results showed improved precision and recall of the co-change analysis over an approach based on association rules.

Kagdi *et al.* [5], [6] presented a heuristic-based approach to link files evolving only inside the source code repository. However, the authors did not link files, *e.g.*, requirements, evolving outside the source code repository.

All the previous promising work in co-change analysis helped in various software maintenance activities, *e.g.*, change impact analysis [23], [24]. To the best of our knowledge, none of the previous work has been used to support RT.

III. CoChaIR: Combining Historical Co-change Information and IR Techniques

We now present the details of CoChaIR that uses textual information of requirements, source code, and historical co-changing information as input. Figure 1 shows the high-level view of CoChaIR. It has three main modules, *i.e.*, an IR engine, co-change analyser, and a similarity calculator. We explain below the abstract model of CoChaIR and explain each module.

A. CoChaIR Abstract Model

In CoChaIR, we represent a traceability link as a triple {requirements, classes, and similarity} and we use the following notations. Let $R = \{r_1, \ldots, r_N\}$ be a set of requirements

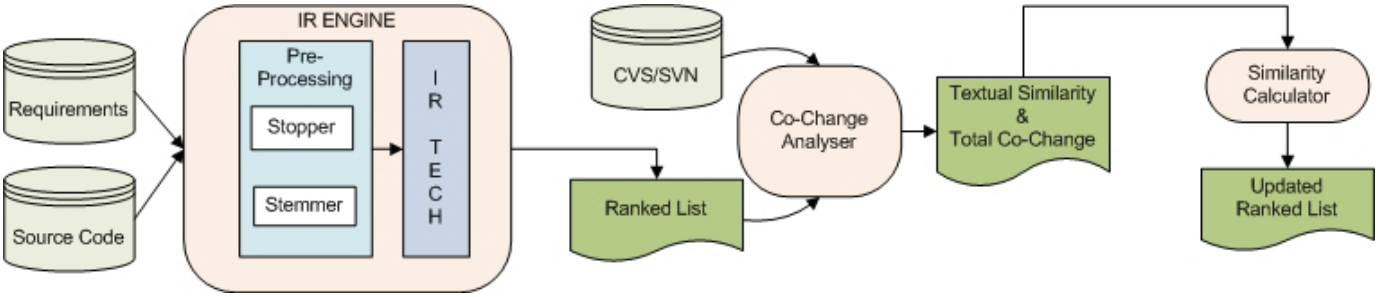


Fig. 1: High-level View of CoChaIR

and $C = \{c_1, \ldots, c_N\}$ be a set of classes supposed to implement these requirements.

Let $\mathcal{L} = \{L_1, \ldots, L_N\}$ be a set where each L_i is a set of classes $\{c_1, \ldots, c_j\}$ linked to a requirement r_i; whose cosine similarity is greater than 0. Let $Q = \{Q_1, \ldots, Q_N\}$ be a set in which each Q_i is a subset of L_i, in which all classes have co-changed together. The function $\alpha(L_i)$ returns the set Q_i of classes co-changing together.

Finally, let us define two functions β, and γ: the first function, $\beta(r_i, c_j)$, returns the similarity score $\sigma_{i,j}$ between a class $c_j \in Q_i$ and a requirement r_i computed by the IR engine and the function $\gamma(c_j, c_k)$ returns how many times a class c_j co-changed with other classes $c_k \in Q_i$. CoChaIR normalises the co-change values for each class c_j as:

$$\delta_j = \frac{\sum_{k=1}^{p} (\gamma(c_j, c_k))}{\max_{g \in Q_i} \{ \sum_{k=1}^{p} (\gamma(c_g, c_k)) \}} \quad (1)$$

where p is the total number of co-changes, $c_j \neq c_k$, max is the maximum number of co-changes for a class c_g in a set Q_i.

We also define $\psi_{i,j}$, a function that computes the final similarity between a class $c_j \in Q_i$ and a requirement r_i by combining the $\sigma_{i,j}$ and δ_j as:

$$\psi_{i,j} = \lambda \, \delta_j + (1 - \lambda) \, \beta(r_i, c_j) \quad (2)$$

where $\lambda \in [0, 1]$, represents the weight of the IR technique and normalised total number of co-changes for a class c_j. The higher the evidence, *i.e.*, number of co-changes $\sum(\gamma(c_j, c_k))$, the higher the new similarity $\psi_{i,j}$. In the contrary, little evidence decreases $\psi_{i,j}$ relatively to the similarities of other $c_j \in Q_i$.

B. IR Engine

CoChaIR can make use of any IR technique as an engine to create baseline RT links $\mathcal{L}$ between requirements R and classes C. CoChaIR is not dependent on a particular IR technique. IR techniques consider both requirements and classes as textual documents to create links among them. IR techniques take some processed documents, as explained in the following, as input to build a $m \times n$ term-by-document matrix, where m is the number of all unique terms that occur in the documents and n is the number of documents in the corpus. Then, each cell of the matrix contains a value $w_{i,j}$, which represents the weight of the i^{th} term in j^{th} document. A weight represents the importance of a term in the corpus of all terms. Various term weighting schemes are available to compute the weight of a term [1], [9].

IR techniques compute the similarity between two documents based on the similarity of their terms and–or the distributions thereof. The similarity 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 between two documents and 0 similarity means that two documents do not share any textual information. A high similarity value between two documents suggests a potential link between them. Various IR techniques [1], [9], [10] can be used to compute the textual similarity between requirements and classes. Thus, the IR engine of CoChaIR generates a set $\mathcal{L}$.

C. Co-change Analyser

Co-change analyser takes RT links $\mathcal{L}$ as input. For each set L_i, the co-change analyser examines how many classes, $c_j \in L_i$, co-changed together in past. A class c_j is stored in a set Q_i if it co-changes together with other classes in L_i. Co-change analyser uses function $\alpha(L_i)$ to generate set Q_i.

More precisely, the co-change analyser mines CVS/SVN logs to examine co-changing classes. A CVS/SVN change log contains several attributes: changed class names, dates of changes, and committers' names that committed the changes. We are only interested in the changed classes and time information. Co-change analysis techniques [3], [4], [21], [22] use the class names and time information to examine the number of times they were committed together in the CVS/SVN. Co-change analysis techniques show two classes were co-changing if they were multiple times, in a defined time window, committed in CVS/SVN. CoChaIR does not depend on any particular co-change analysis technique.

D. Similarity Calculator

The similarity calculator takes the textual similarity $\sigma_{i,j}$ generated by the IR engine and the set Q generated by function $\alpha(L_i)$. It uses the function δ_j to get the total number of co-changes for each c_j. For example, if a requirement R_1 is linked to a class c_1 and the co-change analyser indicates that $c_1 \in Q_1$ co-changed two times with c_2 and four times with c_3 then the total number of relationships found for c_1 is 6, *i.e.*, $\sum(\gamma(c_j, c_k)) = 6$. If $\max_{g \in Q_i} \{\sum_{k=1}^{p} (\gamma(c_g, c_k))$ is 10 then δ_j will be $6/10 = 0.6$. Based on Equation 2, the similarity

calculator recalculates the similarity $\psi_{i,j}$ for each RT link for a class in set Q.

Similarity calculator uses DynWing [8] to compute the λ weight in the Equation 2. DynWing is a dynamic weighting scheme to automatically calculate λ weights. It treats $\sigma_{i,j}$ and δ_j as two different experts giving their opinion for a single RT link. DynWing uses different combination of λ weights till it finds an optimal combination that maximizes the $\psi_{i,j}$. Similarity calculator generates an updated set Q mapped to R with new similarity values for each RT link.

IV. Empirical Study

We perform an empirical study on three datasets to evaluate the effectiveness of our approach to create RT links. We use precision, recall, and ranking to assess the improvement over "traditional" JSM and VSM techniques and, consequently, the reduction of the experts' effort brought to the maintainer when tracing requirements and validating traceability links to source code. We choose JSM and VSM to compare CoChaIR because these two IR techniques provide better results for traceability than other IR techniques [1].

A. Goal, Variables, and Research Questions

The *goal* of our empirical study is to evaluate the effectiveness of our novel approach in creating RT links against two IR techniques, *i.e.*, VSM and JSM, using requirements, source code, and CVS/SVN logs as an additional sources of information. The *quality focus* is the ability of CoChaIR to recover RT links between requirements and source code with better accuracy in terms of precision, recall, and ranking [11]. The *perspective* is that of practitioners and researchers, interested in recovering RT links with greater precision, recall values, and decreased ranking of true positive links than that of currently-available IR-based RT approaches. The *objects* of our case studies are three open-source systems, *i.e.*, iTrust, Pooka, and SIP Communicator.

We use precision, recall, and ranking as dependent variables. Precision and recall measures have values in the range $[0, 1]$ and ranking values depend on the size of the ranked list. We use the approaches, either single IR technique, *i.e.*, JSM and VSM, or CoChaIR$_{JSM}$, and CoChaIR$_{VSM}$, as independent variables. We address following the two research questions in our empirical study:

RQ1 *Does CoChaIR lead to better accuracy?*
RQ2 *Does CoChaIR provide better ranking?*

B. Statistical Evaluation

We assess whether or not the differences in precision, recall values and ranking are statistically significant between the CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM RT approaches. To select an appropriate statistical test, we use the Shapiro-Wilk test to analyse the distributions of our data points. We observe that these distributions do not follow a normal distribution. Thus, we use a non-parametric test, *i.e.*, Mann-Whitney test, to test our null hypotheses.

An improvement in accuracy might be statistically significant but it is also important to estimate the magnitude of the difference between the accuracy levels achieved with a single IR technique and CoChaIR. We use a non-parametric effect size measure for ordinal data, *i.e.*, Cliff's d [7], to compute the magnitude of the effect of CoChaIR on precision, recall, and ranking as follows:

$$d = \left| \frac{\#(x_1 > x_2) - \#(x_1 < x_2)}{n_1 n_2} \right|$$

where x_1 and x_2 are precision, recall, or ranking values achieved with CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM, and n_1 and n_2 are the sizes of the sample groups. The effect size is considered small for $0.15 \leq d < 0.33$, medium for $0.33 \leq d < 0.47$ and large for $d \geq 0.47$.

C. Studied Datasets

We use several criteria to select three datasets, *i.e.*, iTrust, Pooka, and SIP Communicator. First, we select open-source systems, so that other researchers can replicate our experiment. Second, we avoid small systems that do not represent systems handled by most developers. Yet, all three systems were small enough so that we were able to manually recover and validate their RT links. Third, we selected datasets with change log history. For example, some publicly available datasets[1], *e.g.*, eTour and SMOS , do not contain historical information. Finally, their source code and requirements are freely available online.

iTrust[2] is a medical application that provides patients with a means to keep up with their medical history and records as well as communicate with their doctors. iTrust (version 10) dataset contains 35 and 218 requirements and classes respectively. We used 3 years history of SVN logs for co-change information.

Pooka[3] is an e-mail client written in Java using the JavaMail API. Pooka (version 2.0) dataset contains 90 and 298 requirements and classes respectively. We used 10 years history of SVN logs for co-change information.

SIP Communicator[4] is an audio/video Internet phone and instant messenger. SIP (version 1.0) dataset contains 82 and 1,771 requirements and classes respectively. We used 7 years history of SVN logs for co-change information.

We use three oracles, *i.e.*, Oracle$_{\text{iTrust}}$, Oracle$_{\text{Pooka}}$, and Oracle$_{\text{SIP}}$, to compute the precision, recall values, and ranking of CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM when applied on the three studied datasets. For iTrust, we use a manually built oracle by its own developers. We did not create or participate in creating the RT links for iTrust. It is a totally independent dataset. For Pooka and SIP, the first author and another Ph.D. student created traceability links between the requirements of the two systems and their source code classes. They read the requirements and manually looked for classes in

[1] http://www.coest.org/
[2] http://agile.csc.ncsu.edu/iTrust/
[3] http://www.suberic.net/pooka/
[4] http://www.jitsi.org

the source code that implement the requirements. They used Eclipse[5] to search for the source code and stored all manually-built RT links in a FacTrace[6] database. Two professors used the FacTrace voting system to accept or reject all the manually-built RT links. We did not use, at no point of the process, any automated technique to build the oracles. All the requirements and source code are available online[6].

D. Baseline RT Links Recovery

We use the JSM and VSM techniques to create the baseline RT links. Abadi *et al.* [1] performed empirical study on several IR techniques to recover traceability links. Their empirical study results show that JSM and VSM outperform other IR techniques. Thus, we use both JSM and VSM to recover RT links and compare their results in isolation with those of CoChaIR$_{JSM}$ and CoChaIR$_{VSM}$. We performed standard documents pre-processing steps [9], [10], [8] to recover RT links. Below we explain the steps followed by each RT approach to create the RT links.

1) Linking Requirements and Classes using VSM: In VSM, requirements and source code classes are represented as a vector in the space of all the terms. Different term weighting schemes could be used to construct these vectors. In this paper, we use the standard TF/IDF weighting scheme [9]. Term frequency (TF) is often referred as local term weight. TF is described by a $t \times d$ matrix, where t is the number of terms and d is the number of documents in the corpus. TF is often called local weight. TF will assign more weight to the most frequent terms in a document, but this by itself does not mean that they are important terms. The inverse document frequency, IDF, of a term is calculated to measure the global weight of a term and is computed as $IDF = \log_2\left(\frac{|D|}{|d:t_i \in d|}\right)$. Then, TF/IDF is defined as:

$$(TF/IDF)_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \times \log_2\left(\frac{|D|}{|d : t_i \in d|}\right)$$

where $n_{i,j}$ are the occurrences of a term t_i in document, *i.e.*, a requirement or a class, d_j, $\sum_k n_{k,j}$ is the sum of the occurrences of all the terms in document d_j, $|D|$ is the total number of documents d in the corpus, and $|d : t_i \in d|$ is the number of documents in which the term t_i appears.

Once documents are represented as vectors of terms in a VSM, the RT links are created between the requirements and classes with their own similarity value depending on their textual similarity. The similarity is measured by the positive cosine of the angle between requirements and classes vectors (because the similarity between two documents cannot be negative).

2) Linking Requirements and Classes using JSM: JSM is an IR technique proposed by Abadi *et al.* [1] to recover traceability links. It is driven by a probabilistic approach and a hypothesis testing technique. JSM represents each document through a probability distribution, *i.e.*, a normalised $t \times d$ matrix. The probability distribution of a requirement and a class is:

$$p = \frac{n(w,d)}{T_d}$$

where $n(w,d)$ is the number of times a term appears in a document, *i.e.*, a requirement or a class, d and T_d is the total number of terms appearing in a document d. The empirical term distribution can be modified to take into account the global weight of a term, *e.g.*, IDF. After considering the global weight, each document distribution must be normalised. Once all requirements and classes are represented as a probability distribution, JSM computes the distance between their probability distribution and returns a ranked list of RT links. JSM ranks classes via the "distance" of their probability distributions to that of the requirements:

$$\begin{aligned} JSM(q,d) &= H\left(\frac{p_q + p_d}{2}\right) - \frac{H(p_q) + H(p_d)}{2} \\ H(p) &= \sum h(p(w)) \\ h(x) &= -x \log x \end{aligned}$$

where $H(p)$ is the entropy of the probability distribution p, and p_q and p_d are the probability distributions of the two documents (a "requirement" and a "class"), respectively. By definition, $h(0) \equiv 0$. We compute the similarity between a requirement and a class using $1 - JSM(q,d)$. The similarity values are in $]0,1]$.

E. Linking Requirements and Classes using CoChaIR

In CoChaIR, we take the processed corpora and co-change information of classes as input to reorder and filter baseline links. Any IR technique and co-change analysis technique could be integrated into CoChaIR. In this paper, we use the JSM and VSM IR techniques to build the baseline links and Macocha as co-change analyser. Macocha provides promising results for co-change analysis [4].

For each requirement, JSM and VSM generate a ranked list of classes. Each ranked list contains potential true positive RT links in descending order. We use Macocha to examine the co-change relationships between a class to other classes in the same ranked list. We discard a class from the ranked list if it does not have any co-change history with other classes in the same ranked list, *i.e.*, L_i. Thus, for each requirement, we have only the classes that have some co-change history with other classes in the ranked list, *i.e.*, Q_i, and the number of times a class co-changed with other classes. We combine the total co-changes information and textual similarity computed using JSM or VSM (see Equation 2) to compute a new similarity value for each RT link. To select λ values, we use DynWing [8] to compute an optimal weight for λ in Equation 2. Thus, CoChaIR produce two sets, *i.e.*, CoChaIR$_{JSM}$ and CoChaIR$_{VSM}$, of RT links.

V. Empirical Study Results

This section presents the results of our two research questions. For each research question, we present its motivation,

[5]http://www.eclipse.org/

[6]http://www.factrace.net/tool

approach, evaluation metric(s), and a discussion of our findings.

RQ1: ***Does CoChaIR lead to better accuracy?***

Motivation: Linking all the requirements to all the classes could help to create RT links. However, it will create many false positive links. Consequently, it will require more developer's effort to manually discard false positive link. IR-based RT approaches automatically create RT links but with low accuracy. Thus, it does not free a developer from manual verification of false positive links. We use CoChaIR to improve the accuracy of existing IR techniques by integrating historical co-change information with IR techniques. In this question, we want to measure the amount of improvement our proposed approach brings over state of the art IR-based RT approaches, *i.e.*, JSM and VSM.

Approach: To answer RQ1, we perform an experiment using CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM RT approaches on three datasets, *i.e.*, iTrust, Pooka, and SIP Communicator. We use precision and recall to measure the improvement brought by CoChaIR over JSM and VSM. For RQ1, we formalise following null hypotheses:

H_{01}**:** *There is no statistical difference in the precision of the recovered RT links when using CoChaIR$_{JSM}$ or a JSM-based approach.*

H_{02}**:** *There is no statistical difference in the recall of the recovered RT links when using CoChaIR$_{JSM}$ or a JSM-based approach.*

We have similar null hypotheses, *i.e.*, H_{03} and H_{04}, pertaining to CoChaIR$_{VSM}$ and VSM.

We perform several experiments with various threshold points on the recovered RT links to perform statistical tests on precision and recall values. We used multiple threshold points to analyse whether or not CoChaIR improves the precision and recall on all the points. We use a threshold t to prune the set of RT links, keeping only links whose textual similarity values are greater than or equal to $t \in [0, 1]$. We use different values of t from 0 to 1 per steps of 0.01 to obtain different sets of RT links with varying precision and recall values, for all approaches. We then perform paired-wise Mann-Whitney test to measure the improvements brought by CoChaIR. In the paired-statistical tests, two chosen approaches must have the same number of data points. Therefore, we keep the same threshold t values for both approaches. For example, if JSM discards all RT links whose textual similarity values are below the 0.75 threshold, then we also use the same 0.75 threshold for CoChaIR$_{JSM}$.

Evaluation Metrics: We use the RT links' set of CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM, as described in previous paragraph, to compute the accuracy of each approach on iTrust, Pooka, and SIP. Two state-of-the-art IR metrics, *i.e.*, precision and recall, are used to evaluate the accuracy of ranked lists generated by IR techniques and CoChaIR.

TABLE I: Average values of precision and recall for iTrust, Pooka, and SIP datasets and Cliffs d results

	Precision			Recall		
	VSM	CoChaIR$_{VSM}$	Effect Size	VSM	CoChaIR$_{VSM}$	Effect Size
iTrust	48.99	60.98	0.72	23.77	27.09	0.86
Pooka	34.32	38.15	0.55	12.76	14.66	0.12
SIP	14.12	19.69	0.90	14.25	15.74	0.92
	JSM	CoChaIR$_{JSM}$	Effect Size	JSM	CoChaIR$_{JSM}$	Effect Size
iTrust	32.72	45.10	1.0	39.26	44.93	0.93
Pooka	29.87	31.98	0.55	15.67	19.55	0.47
SIP	18.16	19.29	0.74	18.65	20.58	0.87

$$Precision = \frac{|\{relevant\ links\} \cap \{retrieved\ links\}|}{|\{retrieved\ links\}|}$$

Precision is defined as the number of relevant RT links retrieved divided by the total number of retrieved documents by an approach. If precision is 1 (or 100%) then all the recovered RT links are true positive.

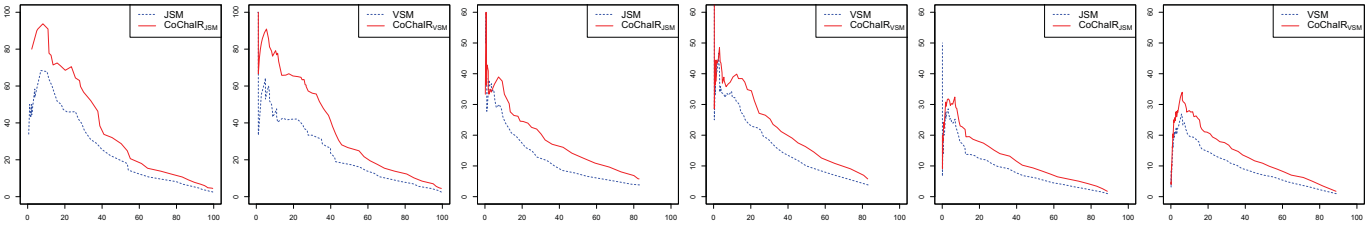
$$Recall = \frac{|\{relevant\ links\} \cap \{retrieved\ links\}|}{|\{relevant\ links\}|}$$

Recall is defined as the relevant RT links retrieved divided by the total number of relevant RT links. It is a ratio between the number of RT links that are successfully retrieved and the number of RT links that are relevant. If recall is 1 (or 100%), then all relevant RT links have been retrieved by an approach.

We use Oracle$_{\text{iTrust}}$, Oracle$_{\text{Pooka}}$, and Oracle$_{\text{SIP}}$ to compute the precision and recall values of the RT links recovered using CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM.

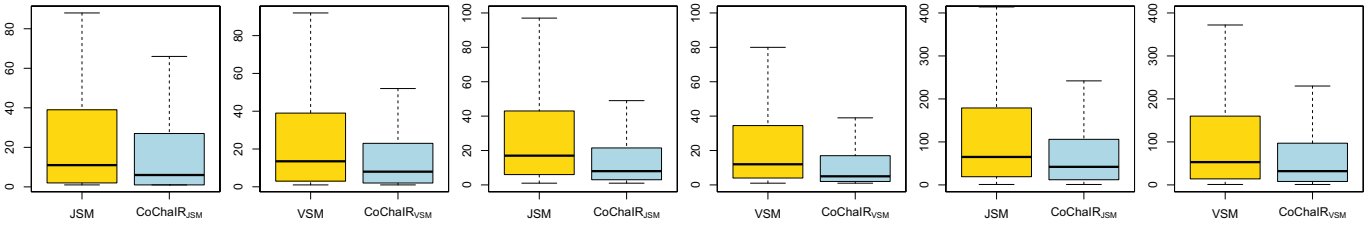
Results: CoChaIR improves the accuracy, on average, by up to 12.38% precision and 5.67% recall. Figure 2 shows the precision and recall graphs of CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM. CoChaIR provides better precision and recall values at all threshold points than the two IR techniques. Table I provides on average precision and recall values calculated by comparing the differences between the CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM RT approaches. CoChaIR provides better accuracy on iTrust dataset than Pooka and SIP. We found Macocha produced less false positive co-changes for iTrust than Pooka and SIP. In future, if we have better co-change analysis techniques then CoChaIR could provide much better results than observed in this empirical study. DynWing automatically calculates optimal weight for λ in Equation 2. However, it is quite possible that using different weighting schemes with CoChaIR would provide different results.

We performed the statistical tests described in Section IV-B to verify whether or not the average improvements in precision and recall are statistically significant. We have statistically-significant evidence to reject H_{01}, H_{02}, H_{03}, and H_{04} null hypotheses. In all the cases (*i.e.*, when comparing our approach to baseline approaches like JSM and VSM across the three studied datasets), p-values are below the standard significant value, *i.e.*, $\alpha < 0.05$. Results show that in 92% of the cases CoChaIR improves the accuracy of IR techniques with large



(a) iTrust (b) iTrust (c) Pooka (d) Pooka (e) SIP (f) SIP

Fig. 2: Precision and recall values of CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM, with the threshold t varying from 0.01 to 1 by step of 0.01. **The X axis shows recall and Y axis shows precision**.



(a) iTrust (b) iTrust (c) Pooka (d) Pooka (e) SIP (f) SIP

Fig. 3: Ranking of CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM. **Y axis shows the rank of classes and X axis shows the RT approach name**.

effect size. Only in the case of Pooka (VSM), we did not observe any significant effect on improvement. However, on average, CoChaIR produced better results for Pooka (VSM). We note that CoCHaIR has a 37.5% better large effect on precision and 8% better large effect on recall, than our previous work [8].

Combining historical co-change information with IR techniques improves the accuracy of IR techniques. CoChaIR provides better precision and recall by up to 12.38% and 5.67% respectively

RQ2: ***Does CoChaIR provide better ranking?***

Motivation: An approach might provide better accuracy, in terms of precision and recall, but all the true positive links could be down in the ranked list. Hence, precision and recall alone are not enough since developers have limited time so they need true positive links higher in the ranked list. Thus, it becomes important to compare the rank of two different RT approaches to measure the improvement brought by proposed approach.

Approach: To answer RQ2, we use the ranking of true positive RT links recovered using CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM of iTrust, Pooka, and SIP datasets. Lower the rank of a true positive link higher the link would be in ranked list. For RQ2, we formulate following null hypotheses:

H_{05}**:** *There is no statistical difference in terms of ranking between CoChaIR$_{JSM}$ and JSM.*

H_{06}**:** *There is no statistical difference in terms of ranking between CoChaIR$_{VSM}$ and VSM.*

We analyse the rank of all true positive links in the ranked list generated by CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM. For each dataset, we generated four ranked list for each RT approach. We compare CoChaIR$_{JSM}$ and CoChaIR$_{VSM}$ generated ranked lists with JSM and VSM generated ranked lists of each requirement.

TABLE II: Average Rankings of True Positive Links Using CoChaIR$_{JSM}$, CoChaIR$_{VSM}$, JSM, and VSM

	VSM vs. CoChaIR$_{VSM}$			JSM vs. CoChaIR$_{JSM}$		
	Ranking			Ranking		
	VSM	CoChaIR	Effect Size	JSM	CoChaIR	Effect Size
iTrust	30.37	18.15	0.77	30.23	18.99	0.67
Pooka	23.57	13.17	0.68	29.15	15.22	0.72
SIP	125.49	76.30	0.88	138.50	82.03	0.89

Evaluation Metrics: In this paper, we use the ranking metric [11] to evaluate the proposed approach. Precision and recall are important measures to analyse the accuracy of an approach. However, both IR metrics do not provide in depth analysis for the location of true positive links in the ranked list generated by an IR-based RT approach. It is quite possible, for example, recall of an approach is high but all the true positive links are at the end of the ranked lists. In this case, a developer must manually verify all the false positive links before he could find the actual true positive RT link. Thus, the higher is a true positive RT link in a ranked list (decreased rank), the less is the developers' effort and the more accurate is an approach.

IR techniques produce a ranked list L_i of classes C potentially linked to a requirement r_i. The ranked list is in a descending order $\mathrm{DescOrd}\,(L_i)$ based on the textual similarity $\beta(r_i, c_j)$ of the RT links. The size of a ranked list n depends on the number of classes in it. Let us define a function $\varphi(r_i, c_j)$ that returns the rank of the true positive link in $\mathrm{DescOrd}\,(L_i)$. For example, if a requirement r_1 is linked to 10 classes $c_{1...10}$ then the ranked list size $\mathrm{DescOrd}\,(L_{1...10})$ would be $n = 10$. $\varphi(r_i, c_j)$ will get the true positive link L_{r_1,c_7} from

an oracle and look into $\mathrm{DescOrd}\,(L_{1...10})$ ranked list. Thus, if the location of L_{r_1,c_7} is 5^{th} in the ranked list then $\varphi(r_1, c_7)$ will return $rank = 5$ for the link L_{r_1,c_7}. If a true positive link does not exist in a ranked list then $\varphi(r_i, c_j)$ will return 0. A 0 rank link means a RT approach could not find the true positive link or in other words low recall. A rank value is from 1 to n. Function $\varphi(r_i, c_j)$ uses an oracle to verify whether or not a RT link is true positive. We eliminate 0 rank links for both proposed and baseline RT approaches.

CoChaIR returns a similar ranked list in descending order of the similarities computed using Equation 2. We use $\text{Oracle}_{\text{iTrust}}$, $\text{Oracle}_{\text{Pooka}}$, and $\text{Oracle}_{\text{SIP}}$ to compute the ranking of true positive RT links recovered using CoChaIR_{JSM}, CoChaIR_{VSM}, JSM, and VSM.

Results: CoChaIR lowers the rank of true positive links by up to 48%. Table II reports the results of the ranking of true positive links. Figure 3 shows that CoChaIR_{JSM} provides slightly better results for JSM than CoChaIR_{VSM}. Lower rank of a true positive link means higher position in the ranked list. For example, in the case of Pooka (JSM), on average across requirements, CoChaIR_{JSM} decreases the rank of true positive RT links from 29.15 to 15.22.

We have statistically significant evidence to reject the H_{05} and H_{06} null hypotheses. The p-values, for all the comparison of CoChaIR_{JSM} vs. JSM and CoChaIR_{VSM} vs. VSM, are below the standard significant value, *i.e.*, $\alpha < 0.05$. Table II shows that ranks were decreased with large effect size, *i.e.*, $d > 0.47$, in all the cases.

CoChaIR helps to decrease the rank, by up to 48%, of true positive RT links and puts true positive RT links higher in the ranked lists when compared to "traditional" IR techniques alone.

VI. Discussion

We now discuss lesson learned from applying CoChaIR on the three datasets.

A. Analysis of Discarding a Non-Co-Changing Class Constraint of CoChaIR

We defined a constraint in CoChaIR to remove a class c_i from a ranked list if the class c_i does not co-change with other classes c_k in set L_i. This constraint may discard some true positive links too. For example, if some classes are recently introduced in the system and they do not have any co-change history. To analyse such cases, we performed an experiment with and without discarding non-co-changing class constraint of CoChaIR.

For each ranked list set L_i, first, we build two sets Q_i and NC_i. Q_i contains all the classes that were in co-change with other classes in set L_i. NC_i contains all the classes that were never co-changed with other classes in set L_i. Second, for each set Q_i and NC_i, we compute a new similarity value. For set Q_i, we follow the same steps as explained in Section III-A to compute a new similarity value. Equation 2 assigns more similarity value to a class if the class co-changes more than the other classes in set Q_i. For set NC_i, we simply divide the textual similarity of every non-co-changing class by two. In this way, we reward a link with higher similarity and penalize a non-co-changing class by decreasing its similarity value. Third, we merge both sets Q_i, NC_i, and their update similarity values to create a new set Q_i^*. Lastly, we compute precision, recall, and ranking of each set Q_i^* and compare it with Q_i.

We found no dramatic difference between the results of Q_i^* and Q_i. Table III shows the results of CoChaIR with and without discarding non-co-changing class constraint. CoChaIR_Q shows the results of with constraint and CoChaIR_Q^* shows the results of without. Table III shows there is no difference between CoChaIR_Q and CoChaIR_Q^* for recall and minor decrease in precision values for CoChaIR_Q^*. In terms of ranking, results slightly went down compared to CoChaIR_Q. However, both CoChaIR_Q and CoChaIR_Q^* results are still significantly better than baseline links. Thus, we conclude that discarding non-co-changing class constraint provides slightly better results than without constraint. However, a project manager can make the final decision whether or not he wants to use a constraint.

We analysed that discarding non-co-changing classes reduced false positive links of iTrust, Pooka, and SIP by up to 43%, 34%, and 44% respectively. Whereas, penalizing non-co-changing classes did not remove any false positive link but yet it provides better results than baseline IR techniques. We observed, only in the case of Pooka and SIP, at $t = 0$, a couple of true positive links were discarded due to the lack of co-change information for the classes. These two to three links were discarded at the cost of 34% to 44% reduced false positive links and 46% lower rank of true positive links.

We observed all the true positive links of iTrust, Pooka, and SIP co-changed together in past. Consequently, removing non-co-changing classes did not negatively impact recall of proposed approach. Thus, we conclude that if a software system does not have long or complete co-change history then penalizing non-co-changing classes could be used to improve the accuracy of IR techniques without missing any true positive links. However, if a system has long change history for all the files then discarding link constraint could be used. A developer can make a choice whether or not to put constraint on CoChaIR before starting the RT links recovery process.

B. Qualitative Analysis

Table I shows better improvement in iTrust dataset than Pooka and SIP. We manually examined all three datasets to find out the reason. There were two main reasons that caused different improvements; first, the requirements for Pooka and SIP were pre-requirement with limited textual information. On average, a pre-requirement contains 15 words. These pre-requirements caused low conceptual similarity between requirements and source code [2]. If there is a low similarity between requirements and source code, then no matter how good an IR-based RT approach is, it may not

TABLE III: Average values of precision and recall for CoChaIR$_Q$ and CoChaIR$^*_Q$

	VSM Results					
	Precision		Recall		Ranking	
	CoChaIR$_Q$	CoChaIR$^*_Q$	CoChaIR$_Q$	CoChaIR$^*_Q$	CoChaIR$_Q$	CoChaIR$^*_Q$
iTrust	60.98	60.08	27.09	27.09	18.15	21.74
Pooka	38.15	37.91	14.66	14.66	13.17	15.68
SIP	19.69	19.28	15.74	15.74	76.30	98.52

	JSM Results					
	CoChaIR$_Q$	CoChaIR$^*_Q$	CoChaIR$_Q$	CoChaIR$^*_Q$	CoChaIR$_Q$	CoChaIR$^*_Q$
iTrust	45.10	44.71	44.93	44.93	18.99	20.93
Pooka	31.98	31.77	19.55	19.55	15.22	17.67
SIP	19.29	19.10	20.58	20.58	82.03	100.19

produce results with high precision and recall values [8]. CoChaIR improves the similarity of true positive links by adding co-change information of classes. For example, JSM computed 12% textual similarity between R_{46}, *i.e.*, "change the way your folders are displayed and manipulated", and class `FolderInfo.java` and placed it at 12^{th} rank in the ranked list of R_{46}. Whereas, CoChaIR$_{JSM}$ combined textual similarity and historical co-change information using Equation 2. CoChaIR$_{JSM}$ assigned a new similarity, *i.e.*, 26%, between R_{46} and `FolderInfo.java` and placed it at the top in the ranked list of R_{46}. CoChaIR tries to overcome the limitation of textual similarity issue and improves IR techniques results. Using CoChaIR as filtering approach with other IR-based RT approaches could provide much better results.

Second, we observed that Macocha produced some false positive co-change results. Macocha and other co-change analysis techniques are not precise yet [4]. However, CoChaIR is not depended on a particular co-change analysis technique; more advance and accurate co-change analysis techniques could be integrated with CoChaIR. We analysed there were some true positive co-changing classes from the co-change analysis point of view [3], [4]. However, when we looked at true positive RT links of these classes, we found that these co-changing classes were linked to different requirements. All these cases led to poor co-change evidence for Pooka and SIP. Consequently, they lead to lower improvement in precision and recall.

In the case of iTrust, all the requirements were well documented post requirements. There was better textual similarity between requirements and source code terms. In addition, there were less false positive co-changing classes. All this led to better improvement for iTrust than Pooka and SIP.

C. Parameter Sensitivity Analysis of ChaIR

We also manually tuned the parameters in Equation 2 to analyse the impact of manual parameter tuning. We found that assigning more weight to co-change information provides higher recall and less precision. DynWing provides better combination between precision and recall values without much human intervention [8]. In our empirical study, for DynWing, we put co-change information higher than textual similarity. We did not change this setting for all the experiments to avoid any bias. We changed global constraint of DynWing to see its impact on the results. We put textual similarity higher than co-change information in a global constraint. This change in global constraint did not dramatically change the results. In both settings results were almost similar and statistically significant.

VII. Threats to Validity

Several threats potentially limit the validity of our experiments. We now discuss potential threats and how we control or mitigate them.

Construct validity: To mitigate construct validity threat, we used widely adopted metrics, precision, recall, and ranking, to assess the IR techniques and CoChaIR as well as their improvement. The oracle (traceability matrix) used to evaluate the tracing accuracy could also impact our results. To mitigate this threat, two Ph.Ds. (one is the first author) manually created RT links oracles then two professors (neither are co-authors) verified their content to avoid imprecision. Moreover, we used iTrust traceability oracle developed by the iTrust developers who did not know the goal of our empirical study.

Internal Validity: The internal validity of our empirical study could only be threatened by our choice of the λ value: other values could lead to different results. To mitigate this threat, we use the same DynWing approach to compute λ for all the datasets and RT approaches. However, it is possible that other λ values might provide different results. In Section VI-A, we analysed the impact of CoChaIR constraint. We used 2 as denominator to penalize non-co-changing classes. It is quite possible using different denominator provides different results. In future, we will analyse the impact of denominator. Using Macocha as co-change analyser could impact the result of our empirical study. A developer may commit multiple classes all together over a longer period of time. This may impact the results of Macocha. Macocha is not yet precise co-change analysis approach. To mitigate this threat, we studied different co-change analysis approaches in the literature. We found Macocha provides better results than previous co-change analysis techniques [4]. It is quite possible using other co-change analysis techniques provide different results. However, CoChaIR is not dependent on any specific co-change analysis techniques. Thus, in future, more advanced and precise co-change analysis techniques could be used with CoChaIR.

External Validity: Our empirical study is limited to three datasets iTrust, Pooka, and SIP. Yet, our approach is applicable to any other software systems. However, we cannot

claim that the same results would be achieved with other systems. Different systems with different CVS/SVN change logs, requirements, reverse engineering tools, and source code may lead to different results. However, the three selected datasets have different SVN change logs, requirements, and source code quality. Our datasets selection reduces the threat to the external validity. However, more studies, preferably on industrial datasets, are required to generalise the results of our empirical study.

Conclusion validity: We mitigate this threat by paying attention to the distribution of our empirical study results. We verified the data distribution of our results using Shapiro-Wilk test. We used non-parametric statistical test, *i.e.*, Mann-Whitney, and effect size, *i.e.*, Cliff's delta, because our data is not normally distributed.

VIII. Conclusion and Future Work

In this paper, our conjecture was that the use of co-change information combined with IR techniques helps to improve the accuracy of IR-based RT approaches. To verify our conjecture, we proposed a new RT approach, *i.e.*, CoChaIR. CoChaIR works as a filtering approach of RT links generated by IR techniques and computes new similarity values based on co-change information for each class.

The results reported in our empirical study demonstrate that, in general, CoChaIR improves the accuracy of IR techniques. Results show that CoChaIR could produce different results on different datasets. However, in all the three studied datasets, CoChaIR statistically improved the accuracy of IR-based RT approaches. We thus conclude that our conjecture is supported: the accuracy of the IR techniques is improved by integrating co-change information. It provides better accuracy and lower the effort required to manually discard false positive links.

There are several ways in which we are planning to extend this work. First, we will analyse other co-change types and change propagation types, *e.g.*, diphase co-change. We will evaluate different co-change analysis techniques and their impact on IR-based RT approaches. We will deploy CoChaIR in a development environment and perform experiments with developers to analyse how effectively CoChaIR can help developers in recovering traceability links.

References

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

[2] N. Ali, Y.-G. Gueheneuc, and G. Antoniol, "Requirements traceability for object oriented systems by partitioning source code," in *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, ser. WCRE '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 45–54. [Online]. Available: http://dx.doi.org/10.1109/WCRE.2011.16

[3] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 563–572.

[4] F. Jaafar, Y.-G. Gueheneuc, S. Hamel, and G. Antoniol, "An exploratory study of macro co-changes," in *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE, 2011, pp. 325–334.

[5] H. Kagdi, J. Maletic, and B. Sharif, "Mining software repositories for traceability links," in *Program Comprehension, 2007. ICPC '07. 15th IEEE International Conference on*, June 2007, pp. 145 –154.

[6] H. Kagdi and J. Maletic, "Software repositories: A source for traceability links," in *International Workshop on Traceability in Emerging Forms of Software Engineering (GCT/TEFSE@07)*, March 2007, pp. 32 –39.

[7] G. Macbeth, E. Razumiejczyk, and R. D. Ledesma, "Cliff's delta calculator: A non-parametric effect size program for two groups of observations," *Universitas Psychologica*, vol. 10, no. 2, pp. 545–555, 2011.

[8] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trustrace: Mining software repositories to improve the accuracy of requirement traceability links," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, p. 1, 2012.

[9] 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.

[10] 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.

[11] 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.

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

[13] W. Zhao, L. Zhang, Y. Liu, J. Sun, and F. Yang, "Sniafl: Towards a static noninteractive approach to feature location," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, pp. 195–226, April 2006.

[14] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," *ACM Trans. Softw. Eng. Methodol.*, vol. 16, no. 4, 2007.

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

[16] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella, "Design-code traceability recovery: selecting the basic linkage properties," *Science of Computer Programming*, vol. 40, no. 2-3, pp. 213–234, 2001.

[17] ——, "Design-code traceability for object-oriented systems," *Annals of Software Engineering*, vol. 9, no. 1, pp. 35–58, 2000.

[18] C. McMillan, D. Poshyvanyk, and M. Revelle, "Combining textual and structural analysis of software artifacts for traceability link recovery," in *Traceability in Emerging Forms of Software Engineering, 2009. TEFSE'09. ICSE Workshop on*. IEEE, 2009, pp. 41–48.

[19] N. Moha, Y.-G. Guéhéneuc, L. Duchien, and A.-F. Le Meur, "Decor: A method for the specification and detection of code and design smells," *Software Engineering, IEEE Transactions on*, vol. 36, no. 1, pp. 20–36, 2010.

[20] F. Khomh, *Patterns and quality of object-oriented software systems*, 2010.

[21] G. Antoniol and Y.-G. Guéhéneuc, "Feature identification: a novel approach and a case study," in *Proceedings of IEEE International Conference on Software Maintenance*. Budapest Hungary: IEEE Press, Sept 26-29 2005, pp. 357–366.

[22] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *Transactions on Software Engineering*, vol. 30, no. 9, pp. 574–586, 2004.

[23] M. Ceccarelli, L. Cerulo, G. Canfora, and M. Di Penta, "An eclectic approach for change impact analysis," in *Proceedings of the 32nd International Conference on Software Engineering*. New York, NY, USA: ACM Press, 2010, pp. 163–166.

[24] G. Canfora, M. Ceccarelli, L. Cerulo, and M. Di Penta, "Using multivariate time series and association rules to detect logical change coupling: An empirical study," in *Proceedings of the 2010 IEEE International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society Press, pp. 1–10.