Requirements Traceability for Object Oriented Systems by Partitioning Source Code

Nasir Ali[1,2], Yann-Gaël Guéhéneuc[1], and Giuliano Antoniol[2]
[1] *Ptidej Team, DGIGL, École Polytechnique de Montréal, Canada*
[2] *SOCCER Lab, DGIGL, École Polytechnique de Montréal, Canada*
E-mail: {nasir.ali,yann-gael.gueheneuc}@polymtl.ca,antoniol@ieee.org

***Abstract*—Requirements traceability ensures that source code is consistent with documentation and that all requirements have been implemented. During software evolution, features are added, removed, or modified, the code drifts away from its original requirements. Thus traceability recovery approaches becomes necessary to re-establish the traceability relations between requirements and source code.**

This paper presents an approach (Coparvo) complementary to existing traceability recovery approaches for object-oriented programs. Coparvo reduces false positive links recovered by traditional traceability recovery processes thus reducing the manual validation effort.

Coparvo assumes that information extracted from different entities (*e.g.*, class names, comments, class variables, or methods signatures) are different information sources; they may have different level of reliability in requirements traceability and each information source may act as a different expert recommending traceability links.

We applied Coparvo on three data sets, Pooka, SIP Communicator, and iTrust, to filter out false positive links recovered via the information retrieval approach *i.e.*, vector space model. The results show that Coparvo significantly improves the of the recovered links accuracy and also reduces up to 83% effort required to manually remove false positive links.

***Keywords*—Traceability, requirements, source code, experts, Object-Oriented, source code partitions.**

I. Introduction

Requirements traceability is defined as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction" [1]. During software evolution developers add, remove, and modify software functionality to meet ever changing user needs. Developers may or may not update documentation and traceability links whenever the source code is modified [2], which creates a logical distance between source code and documentation; possibly invalidating existing traceability relations. It becomes necessary to recover and validate traceability links between documentation, *e.g.*, requirements, and source code to ease software evolution tasks, program understanding, or updating existing system's functionality.

Motivation: Requirements traceability has received much attention over the past decade. Many researchers have used Information Retrieval (IR) approaches [3], [4], [5] to establish traceability links between high-level documents *e.g.*, requirements, manual pages, and design documents, and low-level documents, *e.g.*, source code and UML diagrams [3], [4], [6], [7]. IR-based approaches assume that all software artifacts are in textual format or can be thought of as textual documents. Then, they compute textual similarity between two software artifacts, *e.g.*, a class and a requirement. A high textual similarity means that two software artifacts probably share several concepts [3] and that, therefore they are likely linked to one another. The effectiveness of IR approaches is measured using IR metrics: recall, precision, or average of both (F-measure) [3], [6], [8]. For a given query, recall is the percentage of actual retrieved links over the total number of pertinent links while precision is the percentage of correctly retrieved links to the total number of traces retrieved. High recall can be achieved by simply linking each requirement to a source code file using an IR approach, but it decreases precision. The lower precision, the higher manual intervention is required to review candidate links and remove false positives [3].

In this paper, we present an approach, Coparvo, to reduce the number of false positive links. Coparvo assumes that information extracted from different entities (*e.g.*, class names, comments, class variables or methods names) are different source of information, code partition can be thought of as an information source. Each information source may act as an expert recommending traceability links.

Our conjecture is that including all sources of information, *i.e.*, code partitions, may negatively impact precision and recall. Indeed when modifying a class, a developer much likely keeps the class API in line with the implemented feature while comments may not be updated. Thus, including outdated comments may create huge number of false positive links.

Example: Let us imagine that a developer is required to trace an email client requirement to its source code. The source code is in Java and requirements are written in English. We assume that the developer is using the IR approach, vector space model (VSM), to recover traceability links between source code and requirements. The developer is tracing a requirement Req_1 - "verify email address format before storing it in address book". Let us assume that the `EmailAddressFormatChecker` class is responsible to verify email address format and `AddAddressbookRecord` is responsible to store email addresses in the address book, whereas `SendEmail` sends out email. Let us further assume that in `SendEmail`

an object `emailAddressFormatChecker` of type `EmailAddressFormatChecker` is created to verify email addresses before sending out the emails. In such situation, the developer risks to link Req_1 also to `SendEmail` because VSM will find matched terms, email, address, and format, between Req_1 and `SendEmail`. Thus, it is important to consider the position of the matched term in source code, *e.g.*, variable name versus comments.

Approach: object oriented (OO) Source code is partitioned in four partitions, *e.g.*, class, method, variable names, and comments. Coparvo merges all requirements in one file, further, each Source code partition is used to create a fictitious document containing all the text extracted from the source code related to the given partition. Coparvo considers four experts and thus four documents are created containing class names, methods names, variable names, and comments respectively. In the following, we will use as synonyms the terms partition or expert as both refers to the same concept.

The fictitious documents are used to compute similarity between requirements and the source code partitions. Coparvo ranks partitions based on the computed similarity; the top most high similar partitions are considered as trustable experts. If there are two or more partitions with the same textual similarity to requirements then we consider all of them as an expert. Coparvo then uses the identified experts to classify traceability links recovered with any IR approach. Each expert votes on the link to accept or reject it. To accept a link at least to two experts (or the majority of experts if there are three or more experts) must agree on a link, *i.e.*, have a non-zero computed similarity.

Validation: We apply Coparvo to reduce false positive links of a standard Vector Space Model (VSM) with *TF/IDF* weighting scheme. We use three software systems: Pooka, SIP communicator, and iTrust. Our findings show that, in general, Coparvo improves accuracy of VSM and it also reduces up to 83% efforts required to manually remove false positive links.

This paper is organized as follows: Section II provides a brief description on the state-of-the-art software artifacts' traceability approaches. Section III describes proposed approach in detail and sketches our implementation of proposed approach. Section IV presents the three case studies while Section V & VI reports and discusses their results. Finally, Section VII concludes with future work.

II. Related work

Much work [9], [3], [10] has been done by researchers to recover traceability links between high-level documents, *e.g.*, requirements, and low-level documents, *e.g.*, source code.

Antoniol et al. [9], [11] proposed an approach for automatically recovering traceability links between object-oriented design models and code the similarity of paired elements from design and code. The authors used class attributes as traceability anchors to recover traceability links. Antoniol et al. [12] proposed a method to build traceability links between two software releases of an object-oriented system and point out differences between the releases. The method recovers an 'as is' design from C++ software releases, compares recovered designs at the class interface level, and helps the user to deal with inconsistencies by pointing out regions of code where differences are concentrated. The comparison step exploits an edit distance and a maximum match algorithm. The focus of this work is not on trace dependencies between requirements and code.

Sherba *et al.* [13] proposed an approach, TraceM, based on technique from open-hypermedia and information integration. TraceM manages traceability links between requirements and architecture. An open hypermedia system enables the creation and viewing of relationship in heterogeneous systems. TraceM allows the creation, maintenance, and viewing of traceability relationships in tools that software professionals use on a daily basis. Maider [14] *et al.* refocused attention on practical ways to apply traceability-information models in practice to encourage wider adoption of traceability. The authors highlighted the typical decisions involved in creating a basic traceability-information model, suggested a simple UML-based representation for its definition, and illustrated its central role in the context of a modeling tool. Maletic *et al.* [7] proposed a XML-based traceability query language, TQL. TQL supports queries across multiple artefacts and multiple traceability link types. TQL has primitives to allow complex queries construction and execution support.

Eddy et al. [10] proposed a new technique called prune-dependency analysis that can be combined with existing techniques to dramatically improve the accuracy of concern location. The authors developed CERBERUS, a hybrid technique for concern location that combines information retrieval, execution tracing, and prune dependency analysis. Andrea et al. [15] proposed an approach helping developers to maintain source code identifiers and comments consistent with high-level artifacts. The approach computes textual similarity between source code and related high-level artifacts, *e.g.*, requirements. The textual similarity helps developers to improve source code lexicon. Zou et al. [16] performed empirical studies to investigate Query Term Coverage, Phrasing, and Project Glossary term-based enhancement methods that are designed to improve the performance of a probabilistic automated tracing tool. The authors proposed a procedure to automatically extract critical keywords and phrases from a set of traceable artifacts to enhance the automated trace retrieval.

The precision and the recall [3] of the links recovered during traceability analyses are influenced by a variety of factors, including the conceptual distance between high-level documentation and low-level artifacts, the queries and the applied IR approach. Comparisons have been made between different IR approach, *e.g.*, [17] and [18], with inconclusive

Table I
SOURCE CODE SECTIONS USED IN EXPERIMENTATION

Acronym	Identifier Type
CN	Class Name - one name per file
MN	All Public and Private Method Names of a Class
VN	Class and Method Variable Names of a Class
CMT	All Block and Single line Comments of a Class

results. On certain data set, the vector space model performs favorably in comparison to more complex techniques, such as Jensen inequality or probabilistic latent semantic semantic analyses [18]. Yet, the vector space model [3], [4], [17] with TF/IDF weighting schema [19] is a reference baseline for feature location [20] and traceability recovery [3], [17].

To best our knowledge, all the above mentioned approaches use textual similarity among various software artifacts to recover traceability links. The work presented in this paper is complementary to existing IR-based approaches, because it exploits significant source code partitions in terms of textual similarity and use them as experts to vote on recovered links. Our conjecture in this paper is that including low textual similarity source code parts can lead to low precision and recall.

III. COPARVO

This section describes our proposed approach, COde PARtitioning and VOting, Coparvo, to improve the accuracy of IR-based approach and reduce effort of a project manager, in particular VSM, by partitioning source code. Coparvo is automated and supported by FacTrace[1] nasir-icpc11a, which provides several modules that help from traceability recovery to traceability links verification.

A. Partitioning Source Code

To process source code, a Java parser is used to extract all source-code identifiers. The Java parser build an abstract syntax tree (AST) of the source code that can be queried to extract required identifiers, *e.g.*, class, method names, etc. Each Java source code file is thus partitioned in four parts (See Table I) and textual information is stored in four separate files.

B. Requirements and Source Code Pre-processing

We remove non alphabetical characters and then use the classic Camel Case algorithm to split identifiers into terms. Then, we perform the following steps to normalise requirements and source code sections: (1) convert all upper-case letters into lower-case and remove punctuation; (2) remove all stop words (such as articles, numbers, and so on); and, (3) perform word stemming using the Porter Stemmer bringing back inflected forms to their morphemes.

[1]http://www.factrace.net

C. Requirements and Source Code Indexing

Without the loss of generality, we use VSM irbook99 to index all the processed documents. In VSM each query is a requirement and documents are source code elements. Query and documents are viewed as a vector of terms. Different term weighting schemes can be used to construct these vectors. The most popular scheme is TF/IDF. Term frequency (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. The most frequent term will have more weight in TF but it does not mean that it is important term. The inverse document frequency (IDF) of a term is calculated to measure the global weight of a term: $(TF/IDF)_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \times \log_2\left(\frac{|D|}{d:|t_i \in d|}\right)$, where $n_{i,j}$ is the occurrences of term t_i in document d_j, $\sum_k n_{k,j}$ is the sum of occurrences of all terms in document d_j, $|D|$ is the total number of documents in the collection, and $|d : t_i \in d|$ is the number of documents in which the term t_i appears.

D. Defining and Ascertaining Experts

Let $R = \{r_1, \ldots, r_n\}$ be a set of requirements or high-level documents, $\mathscr{C} = \{C_1, \ldots, C_m\}$ be a set of implementing classes. Following Bunge ontology bunge77, let $X = \langle x, P(x)\rangle$ be a substantial individual *i.e.*, an object, where the object X is identified by its unique identifier x, and $P(x)$ a set of properties, in this paper, the collection of all source code partitions or collection combination thereof *i.e.*, all possible information sources. To define information sources, let ψ_i be a family of functions $i = 1, \ldots N$ each function selects a sub-set of X properties, for example, the class names and/or method names. In other words, each ψ_i function creates a new set of documents having some of the $P(X)$ properties.

We are interested in finding the high similarity values between the set of requirements R, as merged into a single document $R_{all} = \bigcup_j r_j$, and the document created by merging ψ_i projections $\bigcup_j \psi_i(C_j)$. The high cosine similarity shows the high trust over an expert. We order the high to low similarity experts to attain top two experts for voting. We use the cosine similarity between $\bigcup_j \psi_i(C_j)$ and requirement R_{all}. The highest similarities can be computed as:

$$\sigma_{max} = \max_i \{\sigma(R_{all}, \bigcup_j \psi_i(C_j))\}$$

where σ_{max} is the maximum attainable similarity and we are not interested in the absolute value rather in the rank associated to each projection ψ_i to identify most trustable experts and to select top two source-code experts β_i associated to the projections ψ_i for further processing. If two source code partitions have same similarity to requirements then we keep both of them. We consider two extreme cases: (1) *if the difference among source code partitions similarities is equal to or less than* 5% *then we consider both/all experts;*

(2) *if difference between the first and second top source-code projection is equal to or greater than* 95% *then we only consider the first expert (top most information source)*. The rationale of these two cases is that if the difference is equals to or less than 5%, then two or more partitions use similar semantics. The second rare case could occur if developers totally used different identifiers' names for two source code partitions; *e.g.*, coding a "send email function" and commenting it with "patient information" comments. Then, there would be high distance between method name and comments because they do not share any semantics.

We consider at least the two top source-code partitions as experts for voting because if two partitions of a class are linking to a requirement then we can more likely trust the link. We cannot assume that all four partitions would link to a requirement because developers normally add many functionalities in a class abbes2011csmr. In addition, including more experts would impact precision and recall negatively.

There may be as many ψ_i functions as there are subsets of $\mathscr{C}$, however, for practical reasons we are interested in ψ_i that are (1) *arguably meaningful*; and (2) *easy to compute*. For example, a projection dividing classes into text documents corresponding to odd line numbers and even line numbers may be easy to compute but not meaningful. Thus, in this paper, we consider the four projection: CN, VN, MN, CMT and combination thereof, *e.g.*, MN and CMT using both comments and API, which we denote $MN+CMT$ for the sake of simplicity.

E. Link Recovery and Voting Process

Coparvo assumes that traceability links have already been recovered by an IR approach. In this paper, we use the standard VSM model recovering links between elements of $\mathscr{C}$ and R; let this set be $\mathscr{L} = \{l_1, \ldots, l_M\}$ where M can be as high as $n \times m$.

Coparvo uses the top rated expert(s) β_i to score each link l_q and decide via majority voting if the links is likely to be a real link or should be rejected.

Suppose that VSM creates a link between a requirement R_1 "adding prelim support for spam filters" and a class "SpamFilter". Further assume that Coparvo uses CN, MN, VN, and CMT as experts. For Coparvo, links l_q are the base links, experts vote and validate to confirm these links. To accept a link, at least two β_i must agree on a link l_q. Let us further assume, that CN and MN are the top two experts and that there are no ties, *i.e.*, this means Coparvo only uses CN and MN as experts. Then CN and MN will vote on the given link, this is to say the link between a requirement R_1 and a class `SpamFilter`, will be accepted if and only if both CN and MN experts also return a non-zero similarity between `SpamFilter` and R_1. If the links is accepted then it will be assigned the highest similarity among the three computed *i.e.*, standard VSM, CN, and MN.

Finally, the actual way in which experts assign a similarity between a requirement and a code projection ψ_i is not important but arguably it should be coherent with the baseline recovery process and should produce sounds and meaningful results. In this paper we simply re-used VSM also for the different expert β_i. In future work we will investigate the effect of mixed configuration *e.g.*, VSM as baseline and Jensen inequality or topics model for the experts.

IV. Empirical Study

We perform an empirical study with three systems to assess the accuracy of our proposed approach for requirement traceability in term of F-measure. The empirical study provides data to assess the accuracy improvement over a traditional VSM-based approach and, consequently, the reduction of the project managers' effort brought to the maintainer when tracing requirements and validating traceability links to source code.

A. Goal

The *goal* of our case studies is to evaluate the effectiveness of Coparvo in improving accuracy of VSM and reducing effort required to manually discard false positive links. The *quality focus* is the ability of proposed approach to recover traceability links between high-level documents and source code in terms of F-measure [21].

F-measure is the harmonic mean of precision and recall which is computed as $F(j) = \frac{2}{\frac{1}{R(j)}+\frac{1}{P(j)}}$ where $R(j)$ is the recall for the j^{th} document in the ranking. $P(j)$ is the precision for the j^{th} document in the ranking, and $F(j)$ is the harmonic mean of $R(j)$ and $P(j)$ (thus, relative to the j^{th} document in the ranking). The function F assumes values in the interval $[0,1]$. It is 0 when no relevant documents have been retrieved and is 1 when all ranked documents are relevant. Further, the harmonic mean F assumes a high value only when both recall and precision are high. Therefore, determination of the maximum value for the F can be interpreted as an attempt to find the best possible compromise between recall and precision [21].

The *perspective* is that of practitioners and researchers, interested in recovering traceability links with greater F-measure value than currently-available traceability recovery approaches based on IR techniques. In addition, to remove as many as possible the false positive links.

B. Research Questions

The research questions that our empirical study addresses are:

- ***RQ 1*: *How does Coparvo help to to find valuable partitions of source code that help in recovering traceability links?***
- ***RQ 2*: *How does Coparvo help to reduce the effort required to manually verify recovered traceability links?***

Table II
STATISTICS DESCRIBING DATASETS

	Pooka	SIP	iTrust
Version	2.0	1.0	10
Number of Classes	298	1,771	526
Number of Methods	20,868	31,502	3,404
Source Code Sizes	244,870 LOCs	486,966 LOCs	19,604 LOCs
	5.39 MB	27.3 MB	27.3 MB

RQ 3*: *How does the F-measure value of the traceability links recovered by Coparvo compare with a traditional VSM-based approach?

To answer these research questions, we assess the F-measure of proposed approach when identifying correct traceability links between requirements and source code on the one hand and between requirements and source code partitions on the other hand. Thus, we apply Coparvo and a VSM-based approach on the three systems seeking to reject the null hypotheses:

H_0**:** *There is no difference in the F-measure of the recovered traceability links when including whole source code or source code partitions selected by Coparvo.*

It is possible to formulate and accept an alternative hypothesis if the null hypothesis is rejected with relatively high confidence, which admits a positive effect of Coparvo on the retrieval accuracy:

H_a**:** *Recovering taceability links using Coparvo* significantly improves *the accuracy of the IR-based approach, in particular VSM.*

C. Variables

We use F-measure as independent variable and the Coparvo and VSM as dependent variables to empirically attempt rejecting the null hypotheses.

D. Objects

We select the three open-source systems, *Pooka*, *SIP*, and *iTrust*, because they satisfy several criteria. 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 recover and validate their requirements and traceability links manually in a previous work. Finally, their source code was freely available in their respective SVN repositories. Table II provides some descriptive statistics of the three systems.

Pooka[2] is an email client written in Java using the JavaMail API. It supports reading email through the IMAP and POP3 protocols. Outgoing emails are sent using SMTP. It supports folder search, filters, context-sensitive colors, and so on. SIP[3] is an audio/video Internet phone and instant messenger that supports some of the most popular instant messaging and telephony protocols, such as SIP, Jabber, AIM/ICQ, MSN, Yahoo! Messenger, Bonjour, IRC, RSS. iTrust[4] 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, including selecting which doctors to be their primary caregiver, seeing and sharing satisfaction results, and other tasks. iTrust allows the staff to keep track of their patients through messaging capabilities, scheduling of office visits, diagnoses, prescribing medication, ordering and viewing lab results, and so on.

E. Coparvo Use

We now details how we gather and prepare the input data necessary to our empirical study.

Requirements: In a previous work [22], we used PRE-REQUIR [8] to recover requirements for Pooka and SIP. We recovered 90 and 82 functional requirements for both systems respectively. iTrust is complete data set with source code, requirements, and traceability links matrix.

We used these previously-built requirements to create manual traceability links between requirements and source code. Two of the authors created traceability links and the third author verified all the links to accept or reject them. We used FacTrace [5] to create/verify the manually-built traceability links, which form two oracles, $\text{Oracle}_{\text{Pooka}}$ and $\text{Oracle}_{\text{SIP}}$, of respectively 546 and 949 traceability links for Pooka and SIP and which we use to compute the F-measure of proposed approach and of the VSM-based approach. iTrust $\text{Oracle}_{\text{iTrust}}$ is provided by the developer. $\text{Oracle}_{\text{iTrust}}$ contains 183 traceability links at class level.

Obtaining Source Code: We downloaded the source code of Pooka v2.0, SIP v1.0-draft, and iTrust v10.0 from their respective SVN repositories. Table II provides descriptive statistics of the three data set of source code. We made sure that we could compile and run three systems by setting up the appropriate environments and downloading the relevant libraries before building traceability links.

Partitioning Source Code: We query AST of Java source code to extract various parts of source code, in particular CN, MN, VN, and CMT and save them in respective files. The output of this step are the whole source code identifiers' files, partitions of source code, *e.g.*, CN, MN etc., and combination of partitions that we use for creating traceability links as explained in Sections III. We perform pre-processing steps on source code partitions and requirements for further processing.

Ascertaining Experts: First, we merge all the requirements for each system, in particular Pooka, SIP Comm., and iTrust, in one a file. Second, we merge all classes' CN, MN, VN, and CMT in four different files and name them as *cn.txt, mn.txt, vn.txt, and cmt.txt*. Lastly, we use VSM to compute similarity between merged requirement document

[2]http://www.suberic.net/pooka/

[3]http://www.jitsi.org/

[4]http://agile.csc.ncsu.edu/iTrust/

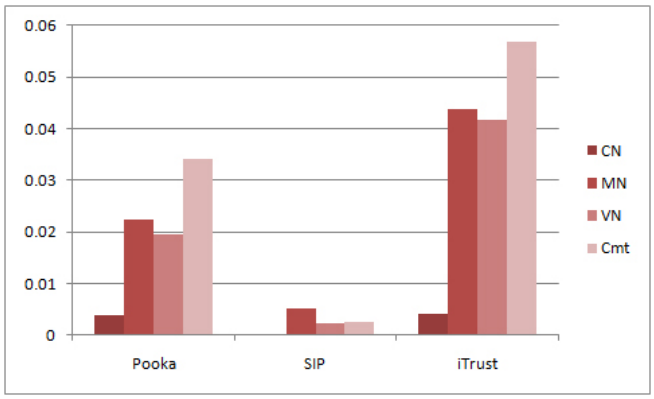

Figure 1. Top Experts for Traceability Link Recovery

and merged source code partitions file. Figure 1 shows the top experts (β_i) that has high similarities with requirements. For example, Pooka shows that MN and CMT must agree on each link. In the case of SIP Comm., we have extreme case (see Section III, where difference between VN and CMT is less than 5%. Therefore, we will keep VN and CMT as second, MN as first expert.

Link Recovery and Voting Process: First, we use the VSM to create traceability links between requirements and whole source code ($\mathscr{L}$), which creates 11,056, 79,422, and 7,166 links for Pooka, SIP, and iTrust, respectively.

Second, we link experts to requirements using the VSM. For example, in Pooka, MN and CMT are top experts. We use every class's MN to link them to requirements and same for CMT. Each set of traceability link between requirements and source code partition is stored in their respective category. These links will be used during expert voting schema. For example, we extract all the *MN* from `SpamSearchTerm.java` and `SpamFilter.java` and use VSM to link it to appropriate requirement, such as "*adding prelim support for spam filters*". We store this link in *MN* category. We repeat the same steps for *CMT* and store recovered links in *CMT* category.

Lastly, we use $\mathscr{L}$ as our base links and asserted experts vote on these base links. For example, in Pooka, if VSM recovers a link between R_1 and $Class_1$, then MN and CMT of $Class_1$ must link to R_1, if any of the compulsory expert does not link to R_1, we discard that link. We finally get the maximum similarity among $\mathscr{L}$ and the experts' links and update the VSM similarity value of that link.

F. Analysis Method

We performed the following analysis on the recovered links by our proposed approach, to answer our research questions and attempt rejecting our null hypotheses.

We use $\text{Oracle}_{\text{Pooka}}$, $\text{Oracle}_{\text{SIP}}$, and $\text{Oracle}_{\text{iTrust}}$ to compute the F-measure values of the VSM and Coparvo. The VSM approach assigns a similarity value to each and every traceability link between requirements and whole source code, whereas Coparvo uses its own process (see Section III) to create traceability links among requirements and ψ_{opt} source code sections.

Table III
COPARVO AND VSM TOTAL RECOVERED LINKS AT 0 THRESHOLD AND P-VALUES OF F-MEASURE

	VSM Recovered Links	Coparvo Recovered Links	Effort Saved	p-Value
Pooka	11,056	4,514	59%	<0.01
SIP Comm.	79,422	13,271	83%	<0.01
iTrust	7,166	4,384	39%	<0.01

We use a threshold t to prune the set of traceability links, keeping only links whose similarities values are greater than or equal to $t \in [0,1]$. We use different values of t from 0.01 to 1 per steps of 0.01 to obtain different sets of traceability links ($\mathscr{L}$) with varying F-measure values, for both approaches. We use these different sets to assess which approach provides better F-measure values. Then, we use the Mann-Whitney test to assess whether the differences in F-measure values, in function of t, are statistically significant between the VSM and Coparvo. Mann-Whitney is a non-parametric test; therefore, it does not make any assumption about the distribution of the data.

V. EXPERIMENT RESULTS

Figure 2 shows the F-measure values of VSM and Coparvo. It shows that β_i voting on recovered links provides better F-measures at different threshold t values.

Table III shows that Coparvo reduces project managers' effort from 39% to 83%. Table IV shows that using different combination of source code partitions and whole source code provide poor results than proposed approach. SIP Comm. results shows that considering VN and CMT as second compulsory expert improves results.

We have statistically significant evidence to reject the H_0 hypothesis for all the three case studies. Table III shows that the p-values are below the standard significant value, $\alpha = 0.05$. We approve alternative hypothesis H_a. Table IV shows that using other source code partitions than β_i also provide some time better precision, recall, and F-measure over VSM but not better than Coparvo. For example, in iTrust only using MN provides up to 19% F-measure but it decreases 8% precision than standard VSM, whereas Coparvo improves all precision, recall, and F-measure values.

Thus, we answer the RQ-1 as follow: partitioning source code and merging all of them and requirements in their respective files helps to find high similarity source code identifiers that can be considered as experts to vote on baseline links.

We answer the RQ-2 as follow: Coparvo uses expert voting scheme on recovered links to filter out false positive links. Coparvo helps to reduce effort, from 39% to 83%, required to manually remove false positive links.

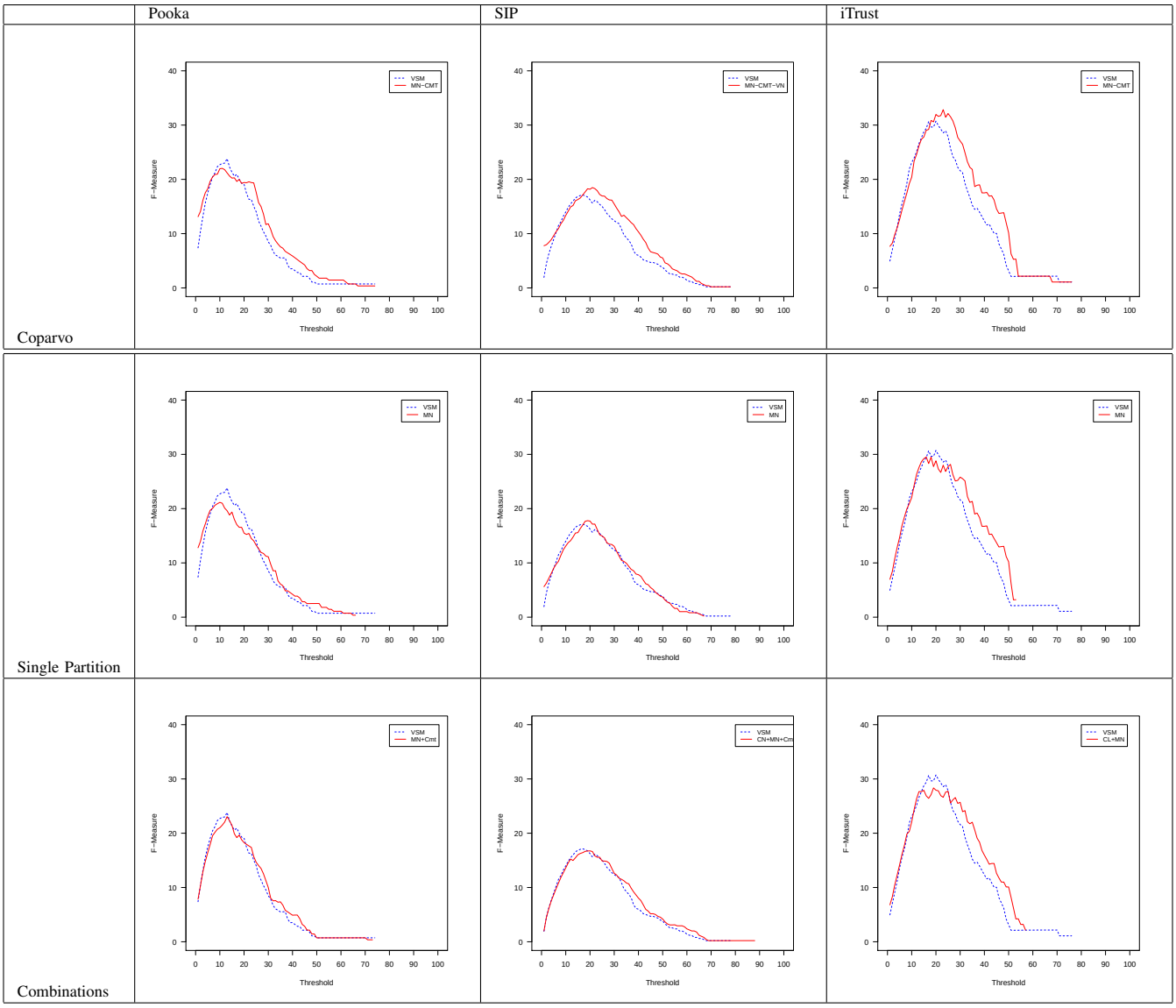

Figure 2. F-measure values at different level of threshold. These graphs only shows the highest F-measure results in different categories for the sake of simplicity.

We answer the RQ-3 as follow: source code partitions β_i statistically increases F-measure over the traditional way of recovering traceability links using whole source code.

VI. Discussion & Qualitative Analysis

We now discuss lesson learned from applying our approaches on the case studies.

A. Effort Reduction

Coparvo is completely automatic, it does not require additional effort from project manager. Table III shows the different number of traceability links recovered with VSM and Coparvo at $t = 0$. It shows that Coparvo helps to reduce by up to 83% the effort to manually discard false positive links. It also provides better precision, recall, and F-measure.

B. Ascertaining Experts

We recovered pre-requirements using PREREQUIR [8] for Pooka and SIP Communicator. Thus, the requirements of Pooka and SIP Communicator were not detailed and have less textual description, on average 15 words per requirement. iTrust, which comes with details requirements, has better textual similarity between requirements and source code partitions.

We only consider top two experts, more in the case of a tie, for voting. At least these two selected experts must agree on a link. The rationale behind this is that including lower quality expert for voting may impact results negatively. We performed more experiments to support our claim. Table IV shows that as we include low quality expert in voting, it start

Table IV
COPARVO, VSM, AND OTHER SOURCE CODE PARTITIONS' COMBINATION RESULTS (BOLD VALUES REPRESENT THE TOP EXPERTS (β_i) VOTING RESULTS). (+) SIGN REPRESENTS THE COMBINATION OF SOURCE CODE PARTITIONS, WHEREAS (-) SIGN REPRESENTS DIFFERENT EXPERTS' VOTING ON RECOVERED LINKS BY VSM (COPARVO)

	Pooka			SIP Comm.			iTrust		
	Precision	Recall	F-Measure	Precision	Recall	F-Measure	Precision	Recall	F-Measure
VSM	42.28	11.14	8.19	14.10	13.25	7.17	49.03	20.38	12.87
MN-CMT	**43.22**	**11.69**	**9.34**	15.99	15.62	8.39	**57.14**	**23.05**	**14.77**
MN-VN-CMT	43.67	10.04	8.73	**17.12**	**14.64**	**8.70**	54.80	23.75	14.53
CN-MN-VN-CMT	40.99	7.16	8.12	16.61	11.33	8.15	42.10	22.04	15.10
CN	36.18	6.48	5.98	13.71	9.87	5.32	31.16	18.77	12.32
MN	42.22	10.75	8.28	17.54	15.87	8.57	41.57	28.63	19.87
VN	18.68	9.74	6.80	20.11	13.12	7.48	34.32	12.24	7.83
CMT	41.12	10.36	7.59	17.47	12.33	6.64	59.73	20.10	12.54
CN + MN	48.47	9.82	7.45	16.15	14.66	7.62	40.71	26.57	18.35
CN + VN	32.70	8.90	6.15	19.89	13.14	7.33	30.73	14.94	9.66
CN + CMT	39.16	10.92	8.04	19.05	12.18	6.49	54.84	20.68	12.79
CN + MN + VN	46.87	9.51	6.81	16.47	13.94	7.54	47.43	18.47	12.56
CN + MN + CMT	41.46	11.37	8.50	18.88	12.59	6.82	49.86	21.64	13.76
MN + VN	37.80	11.11	7.99	16.08	15.05	8.25	52.05	18.09	12.36
MN + CMT	40.67	11.46	8.52	19.06	12.57	6.84	52.56	21.44	13.66
VN + CMT	45.01	9.93	7.03	16.24	12.93	6.99	51.99	19.37	11.64

decreasing the results.

C. Different Scenarios

To effectively evaluate our findings, we compare our proposed approach with three scenarios (i) *the fours experts (whole source code) are required to improve F-measure* (ii) *only one expert is sufficient to improve F-measure (single source code partition)*, and (iii) *simply removing low similarity source code sections can improve F-measure*. Below we discuss these scenarios in details:

The fours experts are required (Baseline): We included all identifiers, in particular CL, MN, VN, and CMT, to recover traceability links using VSM. Table IV shows the results in the first section of table. We also used these results as baseline to compare other two scenarios to measure the accuracy improvement.

Single Source Code Partition: We split each Java file in four partitions: CL, MN, VN, and CMT. We used each partition and combination of the partition to recover traceability links. Table IV shows that in SIP Communicator and iTrust results, only including MN increases precision, recall, and F-measure over baseline results. However, comparing to Coparvo, none of the result using single source code partition has all three values, in particular precision, recall, and F-measure, better than Coparvo.

Combinations of Source Code Partitions: We performed this step to evaluate how much ascertaining experts and their voting help to improve F-measure in comparison to removing low similarity source code partitions to improve F-measure. For example, we only included MN and CMT source code partitions of Pooka to recover traceability links. The fourth part of Table IV shows that it improves F-measure and recall, but decreases precision, when compared to baseline. Whereas, source code partitions combination provides lower precision, recall, and F-measure when compared to Coparvo. This comparison shows that only removing low similarity source code partitions does not help to improve precision, recall, and F-measure. It is also important that each link recovered by VSM must be voted by at least two top experts recovered by Coparvo.

D. Role of Identifiers

In classical IR-based approaches all the terms extracted from an artefact are used to define the semantics of the artefact [23]. In Coparvo we partition source code in four parts and measure the role of each partition on requirements traceability. The third section of Table IV shows that developers usually integrate requirements concepts in MN. The second important part of source code is comments that play an important role to recover traceability links. However, in the case of SIP Communicator, CMT does not have much superiority over VN.

Thus, Coparvo also helps experts to find out which parts of the solution domain has high distance with problem domain. It also helps to improve the quality of identifiers as well as software quality that can help in program comprehension tasks. For example, in the case of SIP communicator, the similarity is very low between source code parts and requirements that results in poor quality links and huge number of recovered traceability links (see Table III). It shows that developers are not using requirements terminology while implementing them. Requirements document size could also be the reason behind this low similarity. It alerts project managers that requirements include little textual information that may be unclear and vague etc. However, the similarity level among source code partitions and requirements shows which partition has more distance or low similarity to requirements.

Figure 1 shows that developers almost never used CN to define any requirements' term in SIP Communicator. Therefore, similarity between requirements and CN is near 0. They implemented requirements concepts at method level. iTrust has high similarity between source code partitions and requirements and it is well documented. Therefore, it provides better precision, recall, and F-measure using VSM and higher improvement after applying Coparvo.

E. 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: Construct validity concerns the relation between theory and observations. In our empirical study, we used widely adopted metrics, precision, recall, and F-measure, to assess the IR technique as well as their improvement. The oracle (traceability matrix) used to evaluate the tracing accuracy could also impact our results. To mitigate such a threat, two authors created manual traceability oracles for Pooka and SIP Communicator and then the third author verified to avoid imprecision in the measurements. Moreover, we used iTrust traceability oracle developed by the developers who did not know the goal of our empirical study.

External Validity: The external validity of a study relates to the extent to which we can generalize its results. Our case studies are limited to three systems, Pooka, SIP, and iTrust. It is not comparable to industrial projects, but the data sets used by other authors [9], [17], [18] to compare different IR methods have a comparable size. However, we cannot claim that the same results would be achieved with other systems. Different systems with different identifiers' quality ,reverse engineering code approach, requirements, using different software artifacts and other internal or external factors [22] may lead to different results. However, the three selected systems have different source code quality and requirements. Our choice reduces this threat to validity.

Conclusion validity: Conclusion validity threats deals with the relation between the treatment and the outcome. We paid attention not to violate assumptions made by statistical tests. Whenever conditions necessary to use parametric statistics did not hold, *e.g.*, assumption on the data distribution, we used nonparametric tests, in particular Wilcoxon test for paired analysis. Wilcoxon does not make any assumption on the data distribution.

VII. Conclusion and Future Work

We proposed the use of source code partitioning to improve the performances of IR-based traceability recovery approach, in particular of Vector Space Model (VSM). Source code partition and merging all partitions and requirements were used by Coparvo to ascertain experts. We kept only top two experts, more in the case of tie, for voting on the recovered links by VSM. Our conjecture is that including lower similarity source code partitions can impact precision and recall.

The results achieved in the reported case study demonstrated that, in general, the proposed approach improves the retrieval accuracy of vector space model. Coparvo could produce different results on different data sets. However, we used three systems that mitigate this threat. In all three systems, Coparvo improved the accuracy of IR-based approach and reduces efforts required to manually remove false positive links.

We analysed that as the source code size increases, IR-based approach recover more links. It makes a project managers' job difficult, because she must manually remove false positive links. The proposed approach automatically removes false positive links and reduces the effort between 39% and 83%. Results show that manual effort reduces as the number of recovered links increases.

We traced pre-requirements that were recovered using PREREQIR [8] for Pooka and SIP Communicator. The quality of pre-requirements is low as it provides the basic definition of users' needs. In the case of iTrust, requirements were properly documented and they were post-requirements. Therefore, we can see a high improvement in precision, recall, and F-measure using Coparvo. Using quality identifiers' names (see Section VI) and requirements will provide better results using Coparvo.

There are several ways in which we are planning to continue this work. First, we are planning to apply Coparvo on heterogeneous software artifacts to analyse accuracy improvement and effort reduction. Second, add more datasets to generalise our findings. Lastly, we will use other IR-based approach to quantify the improvement using our proposed approach. Our empirical case studies demonstrate that each source code partition has potential but not all the partitions are equal in recovering traceability links. We will use different weighting schemes for each source code partition to measure the accuracy of IR-based approaches.

VIII. Acknowledgment

This work has been partially supported by the NSERC Research Chairs on Software Cost-effective Change and Evolution and on Software Patterns and Patterns of Software.

References

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

[2] N. Ali, "Knowledge based reverse engineering process framework," in *In proceedings of International Conference of Software Engineering Research and Practice (SERP'08)*. CSREA Press, 2008.

[3] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, vol. 28, no. 10, pp. 970–983, 2002.

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

[5] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Trust-based requirements traceability," in *ICPC '11: Proceedings of the International Conference on Program Comprehension (ICPC'11)*. Washington, DC, USA: IEEE Computer Society, 2011, p. 10.

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

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

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

[9] 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.

[10] M. Eaddy, A. Aho, G. Antoniol *et al.*, "Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *The 16th IEEE International Conference on Program Comprehension*. IEEE, 2008, pp. 53–62.

[11] G. Antoniol, B. Caprile, A. Potrich, and P. Tonella, "Design-code traceability for object-oriented systems," *Annals of Software Engineering*, vol. 9, no. 1, pp. 35–58, 2000.

[12] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Maintaining traceability links during object-oriented software evolution," *Software: Practice and Experience*, vol. 31, no. 4, pp. 331–355, 2001.

[13] S. A. Sherba and K. M. Anderson, "A framework for managing traceability relationships between requirements and architectures," in *Second International Software Requirements to Architectures Workshop (STRAW 03), Part of International Conference on Software Engineering*, 2003, pp. 150–156.

[14] P. Mader, O. Gotel, and I. Philippow, "Getting back to basics: Promoting the use of a traceability information model in practice," in *TEFSE '09: Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 21–25.

[15] A. D. Lucia, M. D. Penta, and R. Oliveto, "Improving source code lexicon via traceability and information retrieval," *IEEE Transactions on Software Engineering*, vol. 37, pp. 205–227, 2011.

[16] X. Zou, R. Settimi, and J. Cleland-Huang, "Improving automated requirements trace retrieval: a study of term-based enhancement methods," *Empirical Software Engineering*, vol. 15, no. 2, pp. 119–146, 2010.

[17] A. D. 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.

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

[19] W. B. Frakes and R. Baeza-Yates, *Information Retrieval: Data Structures and Algorithms*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[20] 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.

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

[22] N. Ali, W. Wu, G. Antoniol, M. D. Penta, Y.-G. Guéhéneuc, and J. H. Hayes, "Moms: Multi-objective miniaturization of software," in *27th IEEE International Conference on Software Maintenance*, IEEE. IEEE CS Press, 2011, p. 10.

[23] G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the role of the nouns in ir-based traceability recovery," in *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*. IEEE, 2009, pp. 148–157.