An Empirical Study on Requirements Traceability Using Eye-Tracking

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

***Abstract*—Requirements traceability (RT) links help developers to understand programs and ensure that their source code is consistent with its documentation. Creating RT links is a laborious and resource-consuming task. Information Retrieval (IR) techniques are useful to automatically recover traceability links. However, IR-based approaches typically have low accuracy (precision and recall) and, thus, creating RT links remains a human intensive process. We conjecture that understanding how developers verify RT links could help improve the accuracy of IR-based approaches to recover RT links. Consequently, we perform an empirical study consisting of two controlled experiments. First, we use an eye-tracking system to capture developers' eye movements while they verify RT links. We analyse the obtained data to identify and rank developers' preferred source code entities (SCEs), *e.g.*, class names, method names. Second, we use the ranked SCEs to propose two new weighting schemes called SE/IDF (source code entity/inverse document frequency) and DOI/IDF (domain or implementation/inverse document frequency) to recover RT links combined with an IR technique. SE/IDF is based on the developers preferred SCEs to verify RT links. DOI/IDF is an extension of SE/IDF distinguishing domain and implementation concepts. We use LSI combined with SE/IDF, DOI/IDF, and TF/IDF to show, using two systems, iTrust and Pooka, that $LSI_{DOI/IDF}$ statistically improves the accuracy of the recovered RT links over $LSI_{TF/IDF}$.**

***Keywords*—Requirements traceability, source code, eye tracking, LSI, LDA.**

I. Introduction

Requirements traceability (RT) is "the ability to describe and follow the life of a requirement, in both forwards and backwards direction" [1]. RT links help developers to understand programs and ensure that their source code is consistent with its documentation. However, creating RT links is a laborious and resource-consuming task. Moreover, during software evolution, developers add, remove, and modify functionalities to meet ever-changing users' needs without updating RT links due to the lack of resources (time and effort). Consequently, automated techniques to recover RT links are important to save resources. Many researchers have used Information Retrieval (IR) techniques [2], [3], [4] to develop techniques to create traceability links between requirements (and other "high-level" artifacts) and source code. They have shown that IR techniques in general and Vector Space Model (VSM) [3] and Latent Semantic Indexing (LSI) [2] techniques in particular are useful for creating RT links.

However, these researchers also have shown and discussed [5] that IR-based techniques have different levels of accuracy depending on their parameters and on the systems on which they are applied. Thus, they agreed that there is still room for improving the accuracy of IR-based techniques, in particular by understanding how developers create/verify RT links [5], [6]. Thus, in this paper, *our conjecture is that understanding how developers verify RT links can allow developing an improved IR-based technique to recover RT links with better accuracy than previous techniques*.

We bring evidence supporting our conjecture by stating and answering the following research questions:

***RQ1*:** What are the important source code entities[1] (SCEs) on which developers pay more attention when verifying RT links?

By answering the RQ1, we obtain a ranked list of SCEs, ranked based on the average amount of developers' visual attention on each SCE. That list shows that developers pay more visual attention on method names then comments then variable names and, finally, class names. Using this list, we build a new weighting scheme, SE/IDF, which integrates developers' preferred SCEs during RT links verification task. We compare this new weighting scheme against the state-of-the-art scheme, TF/IDF, and ask the following RQ2:

***RQ2*:** Does using SE/IDF allow developing a RT links recovery technique with a better accuracy than a technique using TF/IDF?

We observe in the ranked list of developers' SCEs that developers tend to prefer domain concepts over implementation concepts when verifying RT links. Indeed, while verifying RT links, developers pay more visual attentions to SCEs (class names, method names, and so on) representing domain concepts than those representing implementation concepts. Domain concepts are concepts pertaining to the system use. For example, in the Pooka e-mail client, `addAddress`, `saveAddressBook`, and `removeAddress` in `AddressBook.java` are domain-related identifiers. Implementation concepts relate to data structures, GUI elements, databases, algorithms, and so on. Usually, developers use implementation concepts to recall the names of data types or algorithms. For example, in the Pooka e-mail client, `addFocusListener`,

[1]In this paper, we call "source code entities" the class names, method names, variable names, and comments.

`updateThread`, and `buttonImageMap` in `AddressEntryTextArea.java` are implementation concepts. Using this difference between domain and implementation concepts, we build another weighting scheme, DOI/IDF, which distinguishes between domain and implementation related SCEs. We compare this new weighting scheme against the state-of-the-art scheme, TF/IDF, and ask the following RQ3:

***RQ3*:** Does using DOI/IDF allow developing a RT links recovery technique with a better accuracy than a technique using TF/IDF?

To answer these questions, we conduct an empirical study on RT consisting of two controlled experiments. The goal of the first experiment, to answer RQ1, is to observe developers' eye movements using an eye-tracker while they verify RT links to identify important SCEs. We perform this experiment with 26 subjects. We collect and analyse the data related to the developers' visual attention on SCEs. If developers pay more visual attention on a SCE, we consider it an important SCE.

The goal of the second experiment, to answer RQ2 and RQ3, is to measure the accuracy improvement of a LSI-based RT links recovery technique using SE/IDF and DOI/IDF over one using TF/IDF. We use LSI because it has been shown to produce interesting results [2]. We also use latent Dirichlet allocation (LDA), as used in previous work [7], [8], to distinguish SCEs related to domain or implementation concepts.

We create three sets of RT links using LSI: $LSI_{TF/IDF}$, $LSI_{SE/IDF}$, and $LSI_{DOI/IDF}$, to compare them. Results show that, on iTrust and Pooka, $LSI_{DOI/IDF}$ statistically improves precision, recall, and F-measure on average up to 11.01%, 5.35%, and 5.03%, respectively.

This paper is organised as follows: Section II provides a brief overview of the state-of-the-art RT techniques and of the use of eye-tracking system in program comprehension. Section III and IV provide the details of our experiment with the eye-tracker to understand developers' preferred SCEs the results. Sections III and V describe our novel weighting schemes, SE/IDF and DOI/IDF, sketches our implementation, and the results. Section VI reports the discussion on our findings. Finally, Section VIII concludes with future work.

II. Related work

We now present some work related to RT, zoning, and the use of eye-trackers in program comprehension.

RT: In past couple of decades, RT has received much attention. Many researchers have proposed various techniques [3], [9] with various accuracy results [5] to recover traceability links between high-level documents, *e.g.*, requirements, and low-level documents, *e.g.*, source code. IR techniques have been used by researchers since the early works on traceability recovery [3] and feature location [9]. Often, IR-based RT techniques [2], [3], [9] use vector space models, probabilistic rankings, or a vector space model transformed using latent semantic indexing. Whenever available, dynamic data [9] proved to be complementary and useful for traceability recovery by reducing the search space.

Antoniol *et al.* [10] proposed a technique for automatically recovering traceability links between object-oriented design models and code. The authors used class attributes as traceability anchors to recover traceability links. Ali *et al.* [11] proposed COPARVO to recover traceability links between the source code of object-oriented systems and requirements. They partitioned source code in four parts (class, method, and variable name, and comments). Each source code part then "votes" on the RT links recovered using VSM. Results showed that using source code parts improved the accuracy of the VSM RT technique.

Wang *et al.* [6] performed an exploratory study of feature location with developers. They recorded video of the developers' actions during feature location tasks. They analysed the process used by developers to train a second group of developers. Their results showed that the second group performed better than the first one. However, this study only consisted of manual feature location tasks. The authors did not provide any automated RT links recovery technique based on their observations with developers.

We draw inspiration from this previous work to observe concretely how developers read source code when verifying RT links and to use the results of this observation to develop new weighting schemes to improve the accuracy of automated RT links recovery techniques.

Zoning: Some researchers [12], [13], [14] observed that if a term appears in different zones of a document, then its importance changes. This idea that a term has different importance to a reader depending on where it appears has been investigated in the domain of information retrieval [12]. Search engines, such as Google, assign higher ranks to the Web pages that contain the searched terms in specific parts of the pages, *e.g.*, their titles.

Erol *et al.* [13] used a questionnaire to ask participants which parts of some documents are more important for performing tasks. They concluded that title, figure, and abstract are the most important parts for both searching and understanding documents while figure caption is only important for understanding.

Thus, we conclude that physically dividing documents into zones, *e.g.*, title and abstract, has been already investigated in the field of information retrieval. However, we report the first analysis of developers' visual attention on SCEs using eye-tracking. In particular, we believe that people's preferences for documents may differ from developers' preferences for source code.

Eye-Tracking: Eye-trackers have recently been used to

study program comprehension. De Smet *et al.* [15] performed 3 different eye-tracking experiments to investigate the impact of Visitor, Composite and Observer design patterns, and Model View Controller style on comprehension tasks. Yusuf *et al.* [16] also conducted a study using an eye-tracker to analyse how well a developer comprehends UML class diagrams. Their results showed that developers tend to use stereotypes, colours, and layout to have a more efficient exploration and navigation of the class diagrams. Uwano *et al.* [17] also conducted an experiment to characterise the performance of developers while reviewing the source code. They concluded that the more developers read the source code, the more efficiently they find defects. Sharif *et al.* [18] carried an eye-tracking experiment to analyse the effect of identifier style, *i.e.*, camel case and underscore, on developers' performance while reading source code. Sharif *et al.* [19] suggested in their position paper that eye-tracking system could be used in the field of traceability.

To the best of our knowledge, none of previous work performed experiment to analyse what are important SCEs for developers, how does SCE impact RT if different importance are given to each SCE, and the role of domain and implementation entities. The work presented in this paper is complementary to the existing IR-based RT techniques, because it exploits the identifiers' importance based on their position (*e.g.*, class or method names) or their role (*e.g.*, domain or implementation).

III. Empirical Study

Goal: Our empirical study consists of two controlled experiments. The goal of our first experiment is to identify developers' preferred SCEs using an eye-tracker during RT links verification process. In particular, we want to observe precisely which SCE receives more visual attention while verifying RT links. In our second experiment, our goal is to improve an IR-based RT links recovery technique by proposing novel weighting schemes based on developers' preferred SCEs, as observed in the eye-tracking experiment.

Study: In our first experiment, we use an eye-tracker to capture developers' visual attention. In our second experiment, we use the observations from the first experiment to design two novel weighting schemes, *i.e.*, SE/IDF and DOI/IDF. We perform experiments using both schemes combined with LSI to measure the improvement of these schemes when compared to TF/IDF weighting scheme.

Relevance: Understanding the importance of various SCEs is important from the point of view of both researchers and practitioners. For researchers, our results bring further evidence to support our conjecture that all SCEs must be weighted according to their importance. For practitioners, our results provide concrete evidence that they should pay attention to identifier names to help developers in performing RT links recovery. In addition, using more domain terms in source code would improve the accuracy of any IR-based technique to create RT links.

Hypotheses: We formulate the following null hypotheses:

$\boldsymbol{H}_{01}$**:** All SCEs have equal importance for developers.

$\boldsymbol{H}_{02}$**:** Domain and implementation related SCEs have equal importance for developers.

$\boldsymbol{H}_{03}$**:** There is no difference between the accuracy of a LSI-based technique using TF/IDF and using the novel weighting scheme, SE/IDF in terms of F-measure.

$\boldsymbol{H}_{04}$**:** There is no difference between the accuracy of a LSI-based technique using TF/IDF and using the novel weighting scheme, DOI/IDF in terms of F-measure.

$\boldsymbol{H}_{01}$ and $\boldsymbol{H}_{02}$ are related to RQ1 while $\boldsymbol{H}_{03}$ is related to RQ2 and $\boldsymbol{H}_{04}$ is related to RQ3. In Section IV, we address $\boldsymbol{H}_{01}$ and $\boldsymbol{H}_{02}$ while in Section V, we address $\boldsymbol{H}_{03}$ and $\boldsymbol{H}_{04}$.

IV. Experiment Design: Eye-Tracking

Eye-trackers are designed to work with the human visual system, it provides the focus of attention during the cognitive process of a human [20]. An eye-tracker provides us with two main types of eyes-related data: fixations and saccades. A fixation is the stabilisation of the eye on an object of interest for a period of time, whereas saccades are quick movements from one fixation to another. We use fixations measure the subjects' visual attention because comprehension occurs during fixations [21], [22]. This choice directs our experiment settings.

A. Eye-tracking System

We use FaceLAB from Seeing Machine [23] which is a video based remote eye-tracker. We use two 27" LCD monitor for our experiment: the first one is used by the experimenter to set up and run the experiments while monitoring the quality of the eye-tracking data. We use the second one (screen resolution is 1920 x 1080) for displaying the Java source code and the questions to the subjects.

B. Experiment Settings

We show pieces of source code to developers and a requirement to verify RT links. We make sure that all the SCEs are easy to read and understand to avoid any bias. We define an Area of Interest (AOI) as an rectangle that encloses one (and only one) type of SCE. In this paper, we establish two sets of AOIs for our source code stimulus. The first set contains four SCEs (class name, method name, variables, and comments) and we use this set to analyse which SCE is more important than the others for subjects. For the second set, each SCE could belong to either domain or implementation concept. For each AOI, we calculate time by adding the duration of all fixations available in that AOI. Eye-tracker provides us the duration of each fixation in milliseconds. The total time of fixation shows the

amount of time spent by each subject on specific AOI while s/he focused on that part to understand it. The eye-tracker captures the fixations at the granularity of line. Therefore, for each identifier, we consider the line that contains the identifier. We divide the total calculated time spent on a SCE by that specific source code lines of code (LOC) to have the average time for each SCE. For example, if the comments are written on two lines and a subject spends 40 milliseconds to read comments, we divide 40 by 2 to get the time spent on each comment line.

C. Subjects Selection

There are 26 subjects from École Polytechnique de Montréal and University de Montréal. Most of the subjects performed traceability creation/verification tasks in the past; they are representative of junior developers, just hired in a company. The subjects are volunteers and they have guaranteed anonymity and all data has been gathered anonymously. We received the agreement from the Ethical Review Board of École Polytechnique de Montréal to perform and publish this study. There are 7 female and 19 male subjects. Out of 26 subjects, there are 4 masters and 22 Ph.D. students. All of the subjects have, on average, 3.39 years of Java programming experience. The subjects could leave experiment at any time, for any reason, without any kind of penalty. No subject left the study and it took on average 20 minutes to perform the experiment including setting up eye-tracking system. The subjects were aware that they are going to perform RT tasks, but do not know the particular experiment research questions. There was only one case where we could not have fixations because of subject's eyeglasses. Thus, we excluded that subject and also one pilot subject from our study and analyse total 24 subjects.

D. Source Code Selection

We use a criteria to select the source code used in our eye-tracking experiment. We use Java programming language to perform our experiment because it is one of the OOP languages that contains several different SCEs, *i.e.*, class, method, variable names, and comments. In addition, we select short and easy to read source code that could fit on one screen to have better control over eye-tracking system. However, the source code size is similar to previous eye-tracking studies [24], [25]. We remove the automatically generated comments and we mix the domain and implementation related terms in the various SCEs, *e.g.*, class or method name. Font size was 20, and the 6 pieces of source code have 19, 18, 19, 18, 24, 28 LOCs.

E. Links, Tasks, and Questionnaires

We ask subjects to manually verify RT links. A subject can read a requirement and source code on screen to verify a RT link between them. If a subject thinks that yes there must be a link, s/he can simply write on the paper. We capture subjects' eye movement during the RT links verification task. There are six requirements and source codes that subjects must read to verify a link between them. We manually created RT links to evaluate the subjects' answers. The first two authors manually created traceability links between source code and requirements. The third author verifies the manually created links to avoid any bias. One of the subjects performed a pilot-study to validate that the requirements used in the experiment are clear and simple and the source code on the screen is easy to read and understand. The tasks given to the subjects in our experiment consist of answering specific questions by viewing Java source code. Each question deals with the implementation of a requirement. For example, a Java code implements the requirement "CalculateArea class takes an input the radius of a circle to calculate its area". A subject can answer as 'true" if s/he thinks that the code is implementing the specified requirement. Due to page limitation, we provides all the data online [2]

F. Procedure

We conduct the experiment in a quiet, small room where the eye-tracking system is installed. We divide the experiment into four steps. In the first step, we provide a single page instruction and guidelines to the subjects to perform the experiment. In the second step, we ask the subjects to provide their Java programming experience in years, if they have any. Before running the experiment, we briefly give a tutorial to explain the procedure of the experiment and the eye-tracking system (e.g., how it works and what information is gathered by the eye-tracker).

The subjects are seated approximately $70cm$ away from the screen in a comfortable chair with arms and head rests and the eye-tracker is calibrated. This process takes less than five minutes to complete. After this, the environment in front of them is just a Java source code image with a question at the right side of the image.

In the third step, we ask the subjects to read the source code, requirements, and verify traceability links between them. We instruct subjects that they can press space bar button when they find the answer. Space bar button will take them to a blank screen with next image number. Subjects could move easily to write down their answer on the paper. For each question, we ask subjects to spend adequate time to explore the code while we capture subjects' eye movements during traceability verification process.

In the last step, we ask subjects' feedback about source code readability and understandability. We also ask their source code entity preferences to verify RT links. For example, if they prefer more comments over variable names to verify a link. We ask this question to analyse if subjects followed the same pattern in eye-tracking experiment or not.

[2]http://www.ptidej.net/download/experiments/icsm12a/

G. Analysis And Result

We perform the following analysis to answer **RQ1**, (see Section I) and try to reject our null hypotheses. We have one independent variable: the SCEs and we use two dependent variables: (1) the total time of fixation spent by subjects on each SCE and (2) the percentage of correct answers. We use Taupe [15] to analyse the collected data. Taupe software system is developed in Ptidej research group [3] to help researchers visualise, analyse, and edit the data recorded by eye-tracking systems.

For each subject, we calculated the total fixation time that a subject spent at a specific SCE, *e.g.*, class or method name, in milliseconds. We apply Kruskal-Wallis rank sum test on four sets of SCEs to analyse if subjects have equal preference, in particular visual attention, for class, method, variable names and comments or they are all statistically different for them. The Kruskal-Wallis rank sum test is a non-parametric method for testing the equality of the population medians among different groups. The two sets are subjects' data that we collected when they performed RT links verification task. Table I reports that p-value of Kruskal-Wallis test, for all the SCEs results, are statistically significant (the p-value is below than the standard significant value, *i.e.*, 0.05). In addition, a wide majority of the fixations that show subjects' visual attention are found on method names and comments. We analyse that as soon as the subjects read the requirement, they go directly to read method names and–or comments. Also, we analyse all the heatmaps consisting of cumulative fixations of all the subjects for RT verification task. In Figure 1, we also present the heatmap for one of our source codes which shows a large number of fixations on the method name, comments, and requirement.

Moreover, we observed that subjects' visual attention is more on SCEs that belong to domain concept than the implementation. For example, more fixations were at "radius" identifier and very less on "bufRead" identifier. We observed that subjects were more interested in domain related SCEs to comprehend the source code. We have on average 48% domain and 52% implementation SCEs in our experiment. Subjects on average spent 4865.3 milliseconds on domain identifiers, whereas only 1729.8 millisecond on implementation SCEs. Our post-experiment question about subjects' personal ranking for SCEs to comprehend the source code is also in agreement with observations we made with the eye-tracking.

Thus, we reject H_{01} and H_{02} and answer our **RQ1** that developers pay different visual attention on different SCEs to verify RT links. In particular, developers' visual attention changes as the SCEs change. In addition, developers almost doubles the visual attention if a SCE appears in domain or implementation concept of a program.

[3]http://www.ptidej.net/research/taupe/

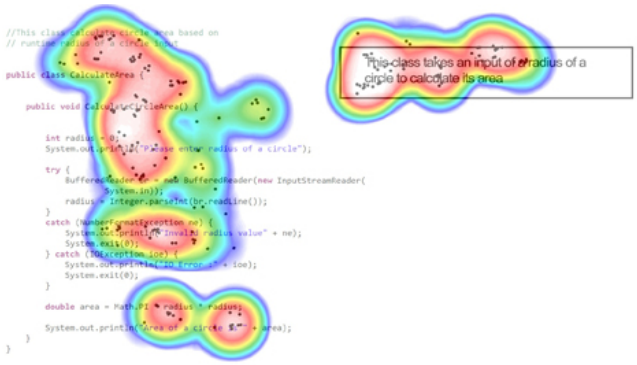

Figure 1. A heatmap showing the cumulative fixations of subjects. The colors red, orange, green and blue indicate the decrease in number of fixations from highest to lowest.

	Class	Method	Variable	Comments	p-value
Avg. time (ms)	2317.25	5701.1	3181.81	4542.41	< 0.01
Ranking	1	4	2	3	–

Table I
AVERAGE TIME ROW SHOWS AVERAGE TIME SPENT ON EACH SCE. RANKING ROW SHOWS THE RANKING OF EACH SCE BASED ON ITS IMPORTANCE/FIXATIONS (4 MEANS THE MOST IMPORTANT).

V. EXPERIMENT DESIGN: SE/IDF AND DOI/IDF

This section describes our proposed weighting schemes SE/IDF and DOI/IDF to improve the accuracy of an IR technique by integrating developers' SCEs preferences. We tune the proposed weighting schemes based on the results that we achieved with our eye-tracking experiment, Section III. We perform experiment on two datasets, *i.e.*, Pooka and iTrust, to analyse how much SE/IDF and DOI/IDF weighting schemes improve the F-measure. We explain all the steps of the experiment in this section. Proposed weighting schemes are supported by FacTrace[4] [4].

IR techniques give importance to a term based on its probability or frequency in the corpus. We analyse that subjects have different preferences, for different SCEs, to perform RT verification task. Our proposed SE/IDF and DOI/IDF weighting schemes allow a developer to provide extra parameters to tune the weighting scheme to increase or decrease a term's importance in a corpus.

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 (here document is referred to the requirement or source code) in the corpus. The TF is often called local weight. The most frequent term has more weight in TF but it does not mean that it is an important term. In our proposed scheme, we consider each source code entity as a separate entity and assign weights according to its importance:

$$SC_j = C_j \cup M_j \cup V_j \cup Cmt_j$$

where, SC_j is the source code of a $class_j$ composed of entities that are class (C_j), method name (M_j), variable

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

name (V_j), and comments (Cmt_j). A source code entity could belong to domain D_j or A_j implementation concept of a program.

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

where $n_{i,j}$ is the number of occurrences of a term t_i in a document d_j, and $\sum_k n_{k,j}$ is the total number of occurrences of all terms in the document. The inverse document frequency (idf) is the quantity of the term distribution in the corpus. The common calculation method is as follows:

$$IDF_i = \log_2 \left(\frac{|D|}{d : |t_i \in d|} \right)$$

where $|D|$ 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. The larger $d : |t_i \in d|$ in documents set, the smaller contribution of the term in the corpus.

$$scetf_{i,j} = \begin{cases} tf_{i,j} \times \alpha \ if \ t_i \in C_j \setminus R_j \\ tf_{i,j} \times (\alpha \times \lambda_1) \ if \ t_i \in C_j \cap R_j \\ tf_{i,j} \times \beta \ if \ t_i \in M_j \setminus R_j \\ tf_{i,j} \times (\beta \times \lambda_2) \ if \ t_i \in M_j \cap R_j \\ tf_{i,j} \times \gamma \ if \ t_i \in V_j \setminus R_j \\ tf_{i,j} \times (\gamma \times \lambda_3) \ if \ t_i \in V_j \cap R_j \\ tf_{i,j} \times \delta \ if \ t_i \in Cmt_j \setminus R_j \\ tf_{i,j} \times (\delta \times \lambda_4) \ if \ t_i \in Cmt_j \cap R_j \\ tf_{i,j} \times \Psi \ if \ R_j \setminus SC_j \end{cases}$$

Where scetf is **s**ource **c**ode **e**ntity **t**erm **f**requency, $\alpha, \beta, \gamma, \delta$ are weights that can be tuned by a developer according to the importance of each SCE, R_j is a j^{th} requirement, and λ_k is term weight if it appears in both SCE and requirement. For example, if a same term appears in both class name and the requirement then it will multiply the weights by λ_1. Where Ψ is the weight for a requirement term that does not appear in source code. In the case, where t_i appears in more than one of the subset $C_j \setminus R_j, M_j \setminus R_j, V_j \setminus R_j, Cmt_j \setminus R_j, C_j \cap R_j, M_j \cap R_j, V_j \cap R_j$, and $Cmt_j \cap R_j$, then we compute $scetf_{i,j}$ for each of the subset and we chose the maximum value. Thus, we define SE/IDF (source code entity/inverse document frequency) as follow:

$$SE/IDF_{i,j} = scetf_{i,j} \times IDF_i$$

Now, we extend our SE/IDF to consider the domain and implementation concepts of a program as follow:

$$DOITF_{i,j} = \begin{cases} tf_{i,j} \times \Upsilon \ if \ t_i \in D_j \\ tf_{i,j} \times \Phi \ if \ t_i \in A_j \end{cases}$$

Where $DOITF$ is **d**omain **o**r **i**mplementation **t**erm **f**requency, Υ and Φ are term weight. For example, if a term belongs to domain concept of a system then we multiply term by Υ. Thus, now we define DOI/IDF (domain or implementation/inverse document frequency) as follows:

$$DOI/IDF_{i,j} = (SE/IDF_{i,j} + DOITF_{i,j}) \times IDF_i$$

A. Objects

We use a criteria to select the source code that are used in our experiment. First, we select open-source systems, Pooka[5] and iTrust [6] , because their source codes are freely available thus other researchers can replicate our experiment. Second, we avoid small systems that do not represent the real world systems usually handled by developers. We download the source code of Pooka v2.0 and iTrust v1.0 from their respective subversion repositories.

Pooka is an email client written in Java using the JavaMail API. Pooka version 2.0 has $244,870$ LOC, 298 classes, $20,868$ functions, and 90 requirements. iTrust is a medical application. iTrust is developed in Java and it has $19,604$ LOC, 526 classes, $3,404$ functions, and 35 requirements.

B. Procedure

We perform the following steps to create RT links between source code and requirements using $LSI_{TF/IDF}$, $LSI_{SE/IDF}$, and $LSI_{DOI/IDF}$.

Gathering Requirements, Source Code and Building Oracles: We recover 90 functional requirements for Pooka in our previous work [4]. We use these requirements to manually create traceability links between requirements and source code. In the case of Pooka, two of the authors create traceability links and the third author verifies all the links to accept or reject them. In the case of iTrust, we use online [6] available requirements and manually built traceability links. $\text{Oracle}_{\text{iTrust}}$ and $\text{Oracle}_{\text{Pooka}}$ contain 546 and 183 traceability links respectively.

Extracting Concepts: We use LDA (Latent Dirichlet Allocation) to extract domain and implementation concepts from the source code. LDA considers that documents are represented as a mixture of words acquired from different latent topics, where each topic is characterised by a distribution of words. More details on using LDA to extract domain concept could be found in [8]. LDA takes three parameters α, β, and k to create the topics. Where α is the dirichlet hyperparameter for topic proportions, β is the dirichlet hyperparameter for topic multinomials, and k is the number of topics. In this experiment, as in previous work [7], [26], the values for the parameters α and β are set to 25 (50/k) and 0.1 respectively. The value assigned for k is 2. We use MALLET[7] to extract concept from source code using LDA.

Generating Corpus: To process source code, we use Java parser [11] to extract all source code identifiers. The Java parser builds an abstract syntax tree (AST) of source code that can be queried to extract required identifiers, *e.g.*, class, method names, etc. Each Java source code file is thus divided in four SCEs and textual information is stored in their

[5]http://www.suberic.net/pooka/
[6]http://agile.csc.ncsu.edu/iTrust
[7]http://mallet.cs.umass.edu

	Technique	Precision	Recall	F-Measure	p-value
Pooka	$LSI_{TF/IDF}$	14.71	21.07	9.52	< 0.01
	$LSI_{SE/IDF}$	18.30	24.86	12.47	
	$LSI_{DOI/IDF}$	25.73	26.42	14.56	
iTrust	$LSI_{TF/IDF}$	36.43	33.14	17.76	< 0.01
	$LSI_{SE/IDF}$	38.66	33.97	18.21	
	$LSI_{DOI/IDF}$	39.55	35.07	20.64	

Table II
AVERAGE PRECISION, RECALL, AND F-MEASURE VALUES AND WILCOXON P-VALUES.

respective source code files. We use these files to apply proposed weighting schemes to create RT links.

Pre-processing the Corpus: We perform standard state-of-the-art [2] pre-processing steps. We remove non-alphabetical characters and then use the classic Camel Case and underscore algorithm to split identifiers into terms. Then, we perform the following steps to normalise requirements and SCEs: (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 to bring back inflected forms to their morphemes.

Experiment Settings: Proposed weighting schemes require parameters to tune the equation. We assign weights to SE/IDF and DOI/IDF equations' parameters based on subjects' SCEs preferences during eye-tracking experiment. We use ranking based on fixations(see Table I), observed during eye-tracking experiment, to define the weights of α, β, γ, and δ (see Section V). We normalise the ranking and assign weights to SCEs. To tune parameters of SE/IDF and DOI/IDF, we use the average fixation time that subjects spent on each SCE, domain, and implementation related SCEs. We assign $0.4, 0.3, 0.2$, and 0.1 weight to β, δ, γ, and α respectively. If a term appears in both requirement and source code, then we simply double the weight of that term. We use lowest SCE weight and divide it by 2 to get the weight if a term only appears in requirement but not in source code. Thus, we assign weight 2 and 0.05 to λ_k and Ψ respectively. We observe that 74% and 26% of time subjects looked at domain and implementation identifiers respectively. Thus, we set 0.74 and 0.26 for Υ and Φ respectively.

RT Links Creation: We use LSI [2] to create $LSI_{TF/IDF}$, $LSI_{SE/IDF}$, and $LSI_{DOI/IDF}$ RT links. The processed corpus is transformed into a term-by-document matrix, where each requirement and source code document is represented as a vector of terms. The values of the matrix cells represent the weights of the terms in the documents, which are computed using the traditional TF/IDF and proposed SE/IDF and DOI/IDF weighting schemes. Once all the requirements and source code documents have been represented in the LSI subspace, we compute the similarities between requirements and source code to create RT links. We take the cosine between their corresponding vector representations for calculating the similarity.

C. Analysis Method

We perform the following analysis on the recovered RT links to answer our research questions, RQ2 and RQ3, and attempt to reject our null hypotheses. We use $LSI_{SE/IDF}$, $LSI_{DOI/IDF}$, and $LSI_{TF/IDF}$ as independent variables and F-measure as a dependent variable to empirically attempt to reject the null hypotheses. We compute F-measure [11] of the $LSI_{SE/IDF}$, $LSI_{DOI/DIF}$, and $LSI_{TF/IDF}$ RT links in comparison to $Oracle_{Pooka}$ and $Oracle_{iTrust}$.

We use a threshold t to prune the set of traceability links, keeping only links whose similarity values are greater than 0. We use different values of t from 0.01 to 1 per step of 0.01 to obtain different sets of traceability links with varying precision and recall values. We use the same t number of threshold values, for comparing two techniques, to have the same number of data points for paired statistical test. We use these different sets to assess which technique provides better F-measure values. Then, we use the Wilcoxon rank sum test to assess whether the differences in F-measure values, in function of t, are statistically significant among the $LSI_{SE/IDF}$, $LSI_{DOI/DIF}$, and $LSI_{TF/IDF}$ techniques.

D. Results

Figure 2 shows the F-measure values of $LSI_{TF/IDF}$, $LSI_{SE/IDF}$, and $LSI_{DOI/IDF}$. It shows that $LSI_{DOI/IDF}$ provides better F-measures at all the different values of threshold t. Figure 2 shows that assigning different weights to SCEs, depending on their role and position in source code, provides better accuracy.

Table II shows that $LSI_{DOI/IDF}$ statistically improves on average up to 11.01%, 5.35%, 5.03% for precision, recall, and F-measure respectively. Results show that adding domain and implementation related SCEs information statistically improve over SE/IDF weighting. We perform Wilcoxon rank sum test on F-measure scores to analyse if the improvement is statistically significant or not.

We have statistically significant evidence to reject the H_{03} and H_{04} hypothesis for both datasets' results. Table II shows that the p-values are below than the standard significant value, *i.e.*, $\alpha = 0.05$. Thus, we answer the **RQ2** and **RQ3** as follow: integrating the developers knowledge, *i.e.*, SCEs preferences, in the IR technique statistically improves the accuracy. Adding the domain and implementation related SCEs information yield better accuracy than SE/IDF. Integrating developers' knowledge in the automated techniques could yield better accuracy, thus it is important to further observe and analyse developers to find out how they perform RT tasks.

VI. DISCUSSION

In response to the research questions defined in Section I, we analyse that developers have different preferences for

different SCEs. Mostly, developers read method names to search for the specific implementation of a requirement. We observe that as soon as developers read a requirement they start looking for specific methods of interest and read the comments. Developers least bother with the class names because one class may contain several functionalities [11]. This is also the case for variable names because variable names could be object names of other classes. To avoid the bias of eye-tracking observations, we asked our subjects through a post-experiment questionnaire about the source code entities that help them to verify RT links. The results of cross verification questions are in agreement with the observations that we made with the eye-tracking system. This step gives us confidence in our findings and mitigates the threats to validity.

In regular document zoning, researchers mentioned that the title of a document is important [12], [13], [14]. If we map a document structure into source code structure then class name is equal to a document title. In our experiment, we analyse that developers do not give preference to class name. We assigned more weight to class name and observed that it decreases the F-measure value. Thus, we conclude that a document is not same as source code. In addition, Figure 2 shows that if we only consider SCEs and assign weights, it does not improve as much as if we integrate the domain and implementation related SCEs information.

We integrated our eye-tracking observations in an IR-based technique, to analyse if it helps, to improve its accuracy when compared to $LSI_{TF/IDF}$. Our results show that observing how developers perform RT tasks and integrating those observations in an automated RT technique can improve its accuracy. This is the first step towards using eye-tracking systems to learn how developers perform RT tasks. Results are promising and exploring more into previously mentioned direction (observing developers) can improve the accuracy of automated techniques.

We analyse less effectiveness of SE/IDF and DOI/IDF on iTrust than on Pooka. We investigate the reason, we find that in iTrust, many classes do not contain all the SCEs, *i.e.*, class, method, variable name, and comments. In addition, we observe that iTrust contains more implementation related SCEs than domain. In the case of Pooka, we observe that it contains more domain related SCEs than implementation. We observe that if a dataset contains all the SCEs and more domain related identifiers then proposed SE/IDF and DOI/IDF weighting schemes perform better accuracy in terms of F-measure. However, even in the case of iTrust, it has less domain related SCEs and fewer classes containing all the SCEs; the proposed weighting schemes provide better accuracy. We observe that as the dataset size increases, SE/IDF and DOI/IDF provide much better results. For example, Pooka is ten times bigger than iTrust dataset and Pooka provides better results than iTrust. Our conjecture is that on bigger datasets SE/IDF and DOI/IDF would provide better results. However, we need more empirical studies to support this claim.

We set the weights of our weighting schemes based on the observations we made during our eye-tracking experiment. It is quite possible that using other weights may yield different results. However, with the current parameters, we observe that proposed weighting schemes outperform traditional TF/IDF. We will perform more case studies in future to analyse the impact of current static weight on other datasets.

VII. Threats to Validity

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

Construct validity: Construct validity concerns the relation between theory and observations. In our empirical study, they could be due to measurement errors. We use time spent on each AOI and percentages of correct answer to measure the subjects' performances. These measures are objective, even if small variations due to external factors, such as fatigue, could impact their values. We minimise this factor by using small source code and all the subjects finished the whole experiment within 20 minutes. In our empirical study, we use the widely-adopted metric F-measure to assess the IR technique as well as its improvement. The oracle used to evaluate the tracing accuracy could also impact our results. To mitigate such a threat, two authors manually created traceability links for Pooka and then the third author verified the links. In addition, Pooka oracle was not specifically built for this empirical study. We used it in our previous studies [4], [11]. We use iTrust traceability oracle developed independently by the developers who did not know the goal of our empirical study.

Internal Validity: The internal validity of a study is the extent to which a treatment affects change in the dependent variables. Learning threats do not affect our study for a specific experiment because we provide six different source codes and the subjects did not know the experiment goal. Selection of subject threats could impact our study due to the natural difference among the subjects' abilities. We analysed 24 out of 26 subjects with various experience to mitigate this threat. For our proposed technique, individual subjects' Java experience can cause some fluctuation in fixation that may lead to biased results. However, we minimised this threat by asking all subjects for their general source code comprehension preference at the end of the experiment. The results of our post-experiment questionnaire are in agreement with our eye-tracking experiment findings.

External Validity: The external validity of a study relates to the extent to which we can generalise its results. The issue of whether students as subjects are representative of

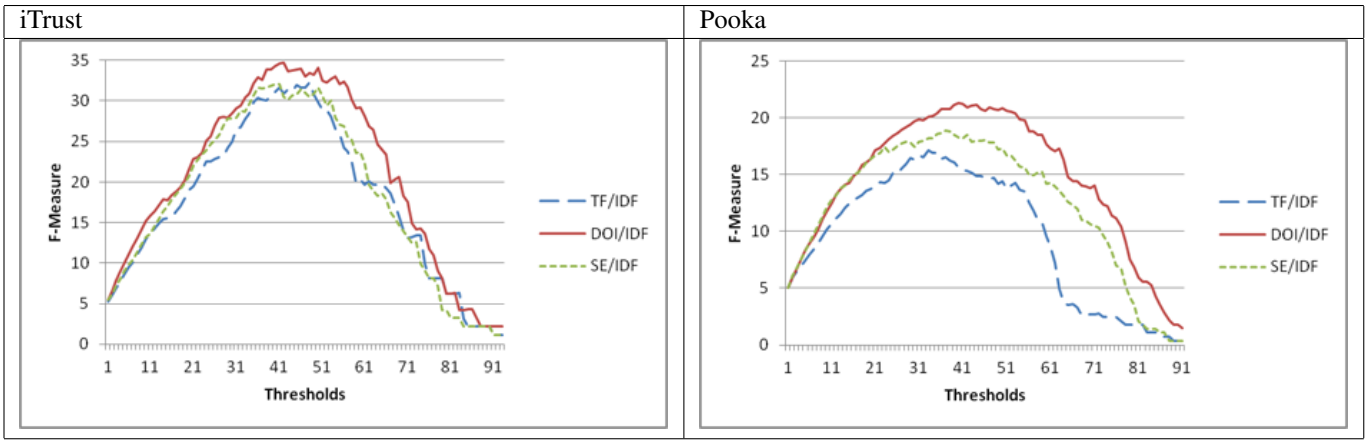

Figure 2. $LSI_{TF/IDF}$, $LSI_{SE/IDF}$, and $LSI_{DOI/IDF}$ F-measure values of iTrust and Pooka

software professionals is one of the threats to generalise our result. However, our subjects are graduate students with the average 3.39 years of Java experience and they have good knowledge of Java programming. In addition, some of the subjects have industrial experience. We use small source code to perform the eye-tracking experiment and in some development environment, *e.g.*, Eclipse, developers' source code preferences could be different. Using development environment with eye-tracking system could weaken the control over the experiment and increase fixation offsets. Therefore, we use post experiment question to ask subjects about their general source code comprehension preference to avoid bias. Our empirical study is limited to two systems, Pooka and iTrust. They are not comparable to industrial projects, but the datasets used by other authors [2], [3], [9] to compare different IR techniques have a comparable size. However, we cannot claim that we would achieve same results with other systems. Different systems with different identifiers' quality, reverse engineering techniques, requirements, using different software artifacts and other internal or external factors [5] may lead to different results. However, the two selected systems have different source code quality and requirements. Our choice reduces this threat to validity.

Conclusion validity: Conclusion validity threats deal with the relation between the treatment and the outcome. We pay attention not to violate assumptions of the performed statistical tests. In addition, we use non-parametric tests that do not make any assumption on the data distribution.

VIII. Conclusion and Future Work

In this paper, we conjecture that understanding how developers verify RT links could help improving the accuracy of IR-based techniques to recover RT links. To support this conjecture, we ask three research questions pertaining to (1) important source code entities (SCEs) used by developers to verify RT links; (2) domain vs. implementation-related entities; and (3) the use of SCEs to propose the new weighting schemes, SE/IDF and DOI/IDF, and to build a LSI-based technique to recover RT links.

We answer these RQs as follows: first, we analysed the eye movements of 24 subjects to identify their preferred SCEs when they read source code to verify RT links using an eye-tracker. Results show that subjects have different preferences for class names, method names, variable names, and comments. They search/comprehend source code by reading method names and comments mostly. We reported that, as soon as developers read a requirement, they pay immediately attention to method names and then comments. We analysed that subjects gives more importance to a SCE if it appears in domain concept of a program.

Second, we propose two new weighting schemes namely SE/IDF and DOI/IDF, to assign different weights to each SCE. For DOI/IDF, we need domain and implementation SCEs for each class. In this paper, we use LDA to distinguish domain concepts from implementation. More advance and complex techniques [7] could be used to separate domain and implementation concepts.

Third, using developers' preferred SCEs and their belonging to either domain or implementation, we then propose SE/IDF and DOI/IDF, new weighting schemes based on the results of the two previous RQs. SE/IDF and DOI/IDF replace the usual TF/IDF weighting scheme so that we could integrate the observations about the subjects' preferred SCEs ranked list into TF/IDF. We use a LSI that uses SE/IDF and DOI/IDF, on two datasets, iTrust and Pooka to show that, in general, the proposed schemes have a better accuracy than TF/IDF in terms of F-measure. In both systems, proposed weighting schemes statistically improve the accuracy of the IR-based technique. We analyse that assigning more weight to domain concepts yields better accuracy.

There are several ways in which we are planning to continue this work. First, we will apply SE/IDF and

DOI/IDF on more datasets. Second, we will analyse in which SCE a requirement term plays a more important role. Third, we will apply proposed weighting schemes on heterogeneous software artifacts to analyse the improvement of the accuracy. Fourth, we will apply SE/IDF and DOI/IDF on feature location techniques. Lastly, we will use some automated techniques to tune SE/IDF and DOI/IDF parameters to improve its accuracy.

ACKNOWLEDGMENT

The authors would like to thank the participants of the study as this work would not be possible without their collaboration. This work has been partially supported by the NSERC Research Chairs on Software Cost-effective Change, 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," *1st International Conference on Requirements Engineering*, pp. 94–101, April 1994.

[2] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2003, pp. 125–135.

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

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

[5] ——, "Factors impacting the inputs of traceability recovery approaches," in *Software and Systems Traceability*, A. Zisman, J. Cleland-Huang, and O. Gotel, Eds. New York: Springer-Verlag, 2011, ch. 7.

[6] J. Wang, X. Peng, Z. Xing, and W. Zhao, "An exploratory study of feature location process: Distinct phases, recurring patterns, and elementary actions," in *Proceedings of 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 213 –222.

[7] S. Abebe and P. Tonella, "Towards the extraction of domain concepts from the identifiers," in *18th Working Conference on Reverse Engineering (WCRE)*, 2011, pp. 77 –86.

[8] G. Maskeri, S. Sarkar, and K. Heafield, "Mining business topics in source code using latent dirichlet allocation," in *Proceedings of the 1st India software engineering conference*. New York, NY, USA: ACM, 2008, pp. 113–120.

[9] 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.

[10] 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.

[11] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol, "Requirements traceability for object oriented systems by partitioning source code," in *18th Working Conference on Reverse Engineering (WCRE)*, oct. 2011, pp. 45 –54.

[12] G. Kowalski, *Information retrieval architecture and algorithms*. Springer-Verlag New York Inc, 2010.

[13] B. Erol, K. Berkner, and S. Joshi, "Multimedia thumbnails for documents," in *Proceedings of the 14th annual ACM international conference on Multimedia*, ser. MULTIMEDIA '06. New York, NY, USA: ACM, 2006, pp. 231–240.

[14] Y. Sun, P. He, and Z. Chen, "An improved term weighting scheme for vector space model," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, vol. 3. IEEE, 2004, pp. 1692–1695.

[15] B. De Smet, L. Lempereur, Z. Sharafi, Y.-G. Guéhéneuc, G. Antoniol, and N. Habra, "Taupe: Visualising and analysing eye-tracking data," *System for Science of Computer Programming*, 2011.

[16] S. Yusuf, H. Kagdi, and J. Maletic, "Assessing the comprehension of uml class diagrams via eye tracking," in *Proceedings of 15th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, 2007, pp. 113–122.

[17] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," in *Proceedings of the 2006 symposium on Eye tracking research & applications (ETRA)*. New York, NY, USA: ACM, 2006, pp. 133–140.

[18] B. Sharif and J. Maletic, "An eye tracking study on camelcase and under_score identifier styles," in *Proceedings of 18th International Conference on Program Comprehension (ICPC)*. IEEE, 2010, pp. 196–205.

[19] B. Sharif and H. Kagdi, "On the use of eye tracking in software traceability," in *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, NY, USA, 2011, pp. 67–70.

[20] A. Duchowski, "A breadth-first survey of eye-tracking applications," *Behavior Research Methods*, vol. 34, no. 4, pp. 455–470, 2002.

[21] K. Rayner, "Eye movements in reading and information processing: 20 years of research." *Psychological bulletin*, vol. 124, no. 3, p. 372, 1998.

[22] A. Duchowski, *Eye tracking methodology: Theory and practice*. Springer-Verlag New York Inc, 2007.

[23] Seeing Machine, "Seeing Machine's website - FaceLAB," http://www.seeingmachines.com/product/facelab/, 2012, accessed July 13 in 2012.

[24] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*. New York, NY, USA: ACM, 2012.

[25] R. Bednarik and M. Tukiainen, "An eye-tracking methodology for characterizing program comprehension processes," in *Proceedings of the 2006 symposium on Eye tracking research & applications*. New York, NY, USA: ACM, 2006, pp. 125–132.

[26] T. Griffiths and M. Steyvers, "Finding scientific topics," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 101, no. Suppl 1, pp. 5228–5235, 2004.