Noname manuscript No.
(will be inserted by the editor)

An Empirical Study on the Importance of Source Code Entities for Requirements Traceability

Nasir Ali, Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol

the date of receipt and acceptance should be inserted later

Abstract Requirements Traceability (RT) links help developers during program comprehension and maintenance tasks. However, creating RT links is a laborious and resource-consuming task. Information Retrieval (IR) techniques are useful to automatically create traceability links. However, IR-based techniques typically have low accuracy (precision, recall, or both) 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 RT techniques to create RT links. Consequently, we perform an empirical study consisting of four case studies. 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 types of Source Code Entities (SCEs), *e.g.*, domain vs. implementation-level source code terms and class names vs. method names. Second, we perform another eye-tracking case study to confirm that it is the semantic content of the developers' preferred types of SCEs and not their locations that attract developers' attention and help them in their task to verify RT links. Third, we propose an improved term weighting scheme, *i.e.*, Developers Preferred Term Frequency/Inverse Document Frequency ($DPTF/IDF$), that uses the knowledge of the developers' preferred types of SCEs to give more importance to these SCEs into the term weighting scheme. We integrate this weighting scheme with an IR technique, *i.e.*, Latent Semantic Indexing (LSI), to create a new technique to RT link recovery. Using three systems (iTrust, Lucene, and Pooka), we show that the proposed technique statistically improves the accuracy of the recovered RT links over a technique based on LSI and the usual Term Frequency/Inverse Document Frequency (TF/IDF) weighting scheme. Finally, we compare the newly proposed $DPTF/IDF$ with our original Domain Or Implementation/Inverse Document Frequency (DOI/IDF) weighting scheme.

Nasir Ali
School of Computing, Queen's University, Kingston, ON, Canada
E-mail: nasir@cs.queensu.ca

Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol
DGIGL, École Polytechnique de Montréal, Canada,
E-mail: zohreh.sharafi,yann-gael.gueheneuc@polymtl.ca, antoniol@ieee.org

1 Introduction

Requirements traceability (RT) is "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (*i.e.*, from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)" [27]. RT links help developers during program comprehension and maintenance tasks to ensure that their source code is consistent with the program documentation. However, creating RT links is a laborious and resource-consuming task. Moreover, during software evolution, developers add, remove, and–or modify functionalities to meet ever-changing users' needs without updating RT links due to lack of resources (time and effort). Consequently, automated techniques to create RT links are important to save resources. Many researchers have used Information Retrieval (IR) techniques [5, 9, 34] to develop techniques to create traceability links between requirements (and other "high-level" artifacts) and source code.

Indeed, several IR techniques have been applied to RT recovery and verification tasks, see [3, 32]. Yet, in general, the Vector Space Model (VSM) [9] is useful for creating RT links. However, VSM has some limitations, *e.g.*, synonym and polysemy [34]. Latent Semantic Indexing (LSI) [34] is an extension to VSM. LSI is proven to overcome the limitation of VSM and can be considered as the baseline approach to compare with. However, it is also known, see [3], that IR-based techniques are not accurate yet. Thus, researchers believe that there is still room for improving the accuracy of IR-based techniques, in particular by understanding how developers verify RT links [3, 46]. Thus, 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.*

To verify our conjecture, in our previous study [7], we observed developers during RT verification task. We answered the following research question while analysing which types of source code entities (SCEs[1]) are important for developers:

– **RQ0:** What are the important Source Code Entities (SCEs) to which developers pay more attention when verifying RT links?

We observed that developers pay more attention to domain-level source code terms, method names, and comments than to other SCEs; for sake of completeness we briefly summarize main contributions of [7] in Section 3.

We named the former research question RQ0 because this paper is grounded on RQ0: it builds upon [7] and investigates the impact of the presence of a type of SCEs on developers' efficiency while performing RT verification tasks. We are interested to observe whether facilitating the detection of the most and the least preferred types of SCEs impact the developers' efficiency. Therefore, in this paper, we first pose the following research question:

– **RQ1:** Are some types of SCEs preferred by developers because of their semantic content or because of their very presence in source code? Through answering

[1] In this paper, we call "source code entities" any domain-level term, implementation-level term, class name, method name, variable name, or comment found in a piece of code. Domain concepts are concepts pertaining to the use of the system by users. Implementation concepts relate to data structures, GUI elements, databases, and algorithms. For example, in the Pooka e-mail client, `addAddress` in `AddressBook.java` class and `addFocusListener` in `AddressEntryTextArea.java` are domain-level and implementation-level concepts, respectively.

RQ1, we observe that highlighting certain types of SCEs does not attract more developers' attention thus we answer RQ1 as follows: it is indeed the semantic content of the SCEs that is important during the RT link verification tasks. Thus, we confirm the results of RQ0 [7].

Based on the answers to RQ0 and RQ1, in this paper, we devise a new term weighting scheme to mimic developers' behavior giving more focus to certain types of SCEs. We name this new scheme Developers Preference-based Term Frequency and Inverse Document Frequency, $DPTF/IDF$ for short.

The traditional weighting scheme, Term Frequency and Inverse Document Frequency, TF/IDF, assigns weight to a term based on its frequency in a corpus whereas $DPTF/IDF$ weighs a term based on its frequency in a corpus and importance during the RT link recovery process. We showed in [6] that using a dynamic parameter tuning approach, named DynWing, led to an increase in the accuracy of a RT link recovery technique. Thus, we also apply DynWing to dynamically tune the parameters of $DPTF/IDF$.

Finally, to ascertain $DPTF/IDF$ accuracy, we compare it with our previous term weighting scheme, DOI/IDF [7], which also considers developers' preferences to assign weights. However, DOI/IDF divides a term twice with IDF, which may cause double penalty for important terms. In addition, DOI/IDF has more parameters to tune than $DPTF/IDF$.

Consequently, to verify the usefulness of $DPTF/IDF$, in this paper, we also address the following two research questions:

– **RQ2:** Does using $DPTF/IDF$ allow developing a RT links recovery technique with better accuracy than a technique using TF/IDF? Results show that on three systems, iTrust, Lucene, and Pooka, the $DPTF/IDF$ weighting scheme statistically improves precision, recall, and F-measure on average up to 11%, 6%, and 5.63%, respectively.
– **RQ3:** Does using $DPTF/IDF$ allow developing a RT links recovery technique with better accuracy than a technique using DOI/IDF? We find that $DPTF/IDF$ provides slightly better results than DOI/IDF. In addition, DynWing can dynamically tune $DPTF/IDF$. Thus, new improved $DPTF/IDF$ weighting scheme does not require a developer to tune parameters for each system separately.

To answer the four research questions, we conduct four case studies. The first case study was conducted in our previous paper [7] and the goal was to identify important types of SCEs using an eye-tracking system while developers verify RT links (RQ0). We performed this case study with 26 subjects and six source code fragments. We collected and analysed the data related to the developers' visual attention on SCEs. If developers paid more visual attention to a particular type of SCEs than to others, we considered it a more important type of SCEs. The first case study results showed that developers prefer certain types of SCEs.

However, it could be that their preferences are due to the very presence of these types of SCEs, not to their semantic content. Such a *contingent* preference could be due to the developers' implicit trust that method names are useful, because they have likely been taught in their programming classes that method names are important. This situation is similar to the situation in which subjects trust a piece of information by virtue if it being shown rather than by its intrinsic value, as it has been shown by Pan *et al.* [36] for Google results. Thus, we perform a second

1	**Goal**	To observe which types of SCEs receive more developers' visual attention during RT links verification tasks.
	Dependent variables	(1) Accuracy and (2) average fixation time
	Independent variables	(1) Type of SCE: domain and implementation-level source code terms, class name, method name, variable name, and comment.
	Subjects	26 students: 4 M.Sc. and 22 Ph.D
	Tasks	Six RT links verification tasks
	Results	Results show that developers prefer certain types of SCE. They spent more visual attention on method names and comments and also preferred domain-level SCEs to implementation-level ones.
2	**Goal**	To investigate the impact of the presence of a type of SCEs on subjects' efficiency during RT links verification tasks.
	Dependent variables	(1) Accuracy, (2) average fixation time, (3) total fixation time, and (4) effort
	independent variables	(1) Type of SCE: class name and method name.
	Subjects	14 students: 2 B.Sc., 3 M.Sc. and 9 Ph.D
	Tasks	Six RT links verification tasks
	Results	It is the content of the method names, and not just their presence, that subjects seek to perform the RT verification tasks.

Table 1 The summary of the eye-tracking case studies.

1	**Goal**	To improve an IR-based, RT link recovery technique using the novel $DPTF/IDF$ weighting scheme based on developers' preferred SCEs. Moreover, to compare the results of $DPTF/IDF$ against the previous TF/IDF and DOI/IDF schemes.
	Dependent variables	F-measure
	Independent variables	Sets of RT links using three techniques: $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$.
	Objects	Three open-source systems: iTrust (v10.0), Lucene (v2.9), and Pooka (v2.0).
	Findings	Results show that assigning different importance to different types of SCEs, depending on their role and occurrence in source code provides better accuracy. Moreover, $LSI_{DPTF/IDF}$ provides better F-measures at all the different values of threshold.

Table 2 The summary of the RT case study.

case study, again using an eye-tracking system, to gauge whether developers really consider the meaning of preferred types of SCEs (RQ2). The two eye-tracking case studies are summarized in Table 1.

The goal of the third and fourth case studies, to answer RQ3 and RQ4, is to measure the accuracy improvement of a LSI-based RT link recovery technique using $DPTF/IDF$ over one using TF/IDF or DOI/IDF. We use LSI because it has been shown to produce interesting results and overcome VSM limitations [34]. DOI/IDF requires to recognize domain concepts; as we seek to devise a completely automatic approach, to separate domain concepts from implementation concepts, we use Latent Dirichlet Allocation (LDA) with the same settings used in previous works [2] for the same task, *i.e.*, domain concepts identification. We summarize the RT case study in Table 2. The replication package for all the case studies is available online[2].

[2] http://www.ptidej.net/download/experiments/emse13b/

This paper is organised as follows: Section 2 provides a brief overview of the state-of-the-art RT techniques and of the use of eye-tracking system in program comprehension. Section 3 summarises the eye-tracking case study on developers' preferred types of SCEs and its results. Section 4 describes our second eye-tracking case study to confirm that it is the semantic content of the SCEs that attracted developers' attention and its results. Sections 5 and 6 describe our proposed weighting scheme and its empirical evaluation. Section 7 reports discussions on the scheme and its evaluation. Finally, Section 9 concludes with future work. We thus bring the following contributions with respect to our previous work [7]:

- We confirm that developers prefer certain types of SCEs because of the information that they bring, not because of their very presence.
- The new term weighting scheme, *i.e.*, $DPTF/IDF$, has less tuning parameters than DOI/IDF. In addition, DynWing is integrated into $DPTF/IDF$ scheme to automatically tune the parameters.
- We compare the use of the $DPTF/IDF$ tuned using DynWing and DOI/IDF to analyse the different potential improvements versus the effort required by DOI/IDF to manually tune parameters.
- We perform detailed statistical analysis to compare $DPTF/IDF$ with DOI/IDF.
- We perform case studies on three medium-size open-source systems, *i.e.*, one additional system, Lucene, in addition to iTrust and Pooka.

2 Related work

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

2.1 Eye-tracking

Eye-tracking systems have been used to study program and model comprehension. De Smet *et al.* [21] performed three different eye-tracking case studies to investigate the impact of the Visitor, Composite, Observer design patterns, and the Model View Controller style, on comprehension tasks. Their results showed that different patterns have different impacts on program comprehension. They also proposed a tool, Taupe, to analyse eye-tracking data.

Guéhéneuc *et al.* [29] and Yusuf *et al.* [47] also conducted eye-tracking studies 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 class diagrams while the relationships between classes may be less important.

Bednarik *et al.* [12] used eye-tracking systems to characterise the program comprehension strategies deployed by developers during dynamic program visualization. Their results showed that eye-tracking data can reveal important information about developers' behavior, which can be hard to access using other methods.

Uwano *et al.* [45] conducted an experiment to characterise the developers' performance while reviewing source code. They concluded that the more developers read the source code, the more efficiently they find defects.

Recently, Busjahn *et al.* [15] conducted a study to analyze the process of code reading using 10 subjects. They considered five elements in the source code: comments, comparisons, complex statements, simple assignments, and keywords. They investigated how subjects read a set of small pieces of Java source code to understand them. They concluded that reading natural language text is different from reading source code. They also found out that novices looked more at comments and comparisons while experts looked more at complex statements. Similar to the work of Busjahn *et al.*, in this paper, we divide source code into elements, which we call source code entities. We consider a different set of SCEs based on the structure of the Java class. Also, we look into the semantics of SCEs by considering implementation-level and domain-level ones.

Sharif *et al.* [41] carried on an eye-tracking case study to analyse the effect of identifier style, *i.e.*, camel case and underscore, on developers' performance while reading source code. Their results showed that there is no statistical differences in accuracy between the two styles. Yet, subjects recognise identifiers in the underscore style more quickly than camel case.

To the best of our knowledge, only Sharif *et al.* [43] explored the potential use of eye-tracking systems to improve traceability recovery techniques. However, they did not describe the linking at source-code level and did not quantify their results statistically or otherwise.

Eye-tracking systems have proven useful to understand developers' behaviour. We draw inspiration from this previous work to use an eye-tracking system for observing developers during RT verification task. We observe which types of SCEs are preferred by developers when accurately and efficiently verifying RT links.

2.2 Zoning

The developers' preference for certain types of SCEs could be only due to their particular locations in the source code, rather than to their semantic content. Indeed, some researchers observed that if a term appears in different zones of a document, then its importance changes [25,31,44].

This idea that a term has different importance to a reader depending on where it appears has also been investigated in the domain of information retrieval [31]. 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.* [25] used a questionnaire to ask participants which parts of documents are more important for performing certain tasks. They concluded that titles, figures, and abstracts are the most important parts for both searching and understanding documents while figure captions are only important for understanding.

However, to the best of our knowledge, the importance of terms at different source-code locations and the distinction between domain-level and implementation-level terms has only be explored in our previous work [7]. Moreover, readers' preferences for documents may differ from developers' preferences for source code.

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

2.3 RT

Preliminary to any software evolution task, a developer must comprehend the project landscape [17], in particular, the system architecture, design, implementation, and the relations between its various artifacts. RT links help to understand the relationship between a requirement and its implementation in the source code. However, due to continuous software evolution, RT links often become obsolete or outdated. Automatically recovering RT traceability links is a laborious task.

Many researchers [1,9,34,38] have proposed IR-based RT techniques to automatically recover RT links. Often, IR-based RT techniques [9,34,38] use VSM, probabilistic rankings, or VSM transformed using LSI. Whenever available, dynamic data proved to be complementary and useful for traceability recovery by reducing the search space [38].

Recently, Ali *et al.* [4] proposed COPARVO to recover traceability links between the source code of object-oriented systems and requirements. They partitioned source code into four parts (class name, method name , 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 a VSM-based RT technique.

Andrea *et al.* [19] proposed a technique helping developers to maintain source code identifiers and comments consistently with high-level artifacts. Their technique computes textual similarity between source code and related high-level artifacts, *e.g.*, requirements. The textual similarity helps developers to improve the source code lexicon. They have provided some comparisons between different IR techniques, *e.g.*, [20], with inconclusive results. They showed the importance of term vocabulary for recovering traceability links. However, they did not explore the difference between different types of SCEs.

Wang *et al.* [46] performed an exploratory study of feature location with developers. They recorded 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. They did not use their results to improve an automated RT links recovery technique based on their observations with developers.

In the following, we draw inspiration from the works of Andrea *et al.* [19] and Wang *et al.* [46] but with the aim of improving an automated RT link recovery approach. We use the developers' preferred types of SCEs to propose a novel weighting scheme, which could be used in previous RT link recovery approaches, such as [8,4,34,38], as shown in the following.

3 Developers' Preferred SCEs – An Eye Tracking Study

For the sake of completeness, we recall here a first case study performed to identify the types of SCEs preferred by developers during RT link verification tasks [7]. This case study and its results are summarised in Table 1.

We perform an empirical study using an eye-tracking to identify which types of SCEs are preferred by developers to verify RT links efficiently and accurately. We ask RQ0: What are the important Source Code Entities (SCEs) to which developers pay more attention when verifying RT links? We use an eye-tracking system to identify developers' preferred types of SCEs. Our goal is to observe precisely which types of SCEs receive more developers' visual attention during RT links verification tasks. We formulate the following null hypothesis to answer RQ0:

H_{01}: All types of SCEs have equal importance for developers.

We now recall the study design and steps that we followed in our previous eye-tracking study.

3.1 Eye-tracking System

Eye-tracking systems are designed to work with the human visual system. They provides the focus of subjects' attention during their cognitive process [23]. An eye-tracking system 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 to measure the subjects' visual attention because comprehension mostly occurs during fixations [24, 39]. This choice directs the settings of our first two case studies.

We use FaceLAB from Seeing Machine [40], which is a video based remote eye-tracking system, to collect the eye movements of developers performing RT links verification tasks. Using two built-in cameras, one infrared pad, and one computer, FaceLAB tracks subjects' eye-movements on the computer monitor. We use two 27" LCD monitors for our eye-tracking case studies: the first one is used by the experimenter to set up and run the experiments while monitoring the quality of the eye-tracking data; the second one (screen resolution is 1920×1080) is used to display the stimulus to subjects.

FaceLAB sends eye-movements data to a data visualization tool, Gaze Tracker from *Eye Response*[3]. GazeTracker stores eye-movement data including fixations and saccades associated with each image and displays all fixations on top of the stimulus. The time resolution of the eye-tracking system is the millisecond. We use Taupe [21] to analyse the collected data. Taupe is an open-source eye-tracking data analysis software[4] to help researchers visualise, analyse, and edit the data recorded by eye-tracking systems.

3.2 Case Study Design

We use six questions and source code snippets that subjects must read to verify a link between them. Each pair {question, piece of source code} form a stimulus. Table 3 shows the list of questions. Each question deals with the implementation of one requirement. For example, a piece of source code could implement the requirement "This class takes an input the radius of a circle to calculate its area". A subject could answer "yes" if (s)he thinks that the source code shown on the stimulus is implementing the specified requirement. There are four correct and two

[3] http://www.eyeresponse.com/

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

	Questions	Answer
1	This class takes as input the radius of a circle to calculate its area.	True/False
2	This class uses a JDBC driver to connect to a database and get data from myTable.	True/False
3	This class draws a circle in a JFrame.	True/False
4	This class uses the Java 2D API to draw lines and a rectangle in a JFrame.	True/False
5	This class compares two strings.	True/False
6	This class uses a JDBC driver to connect to a database and insert data in myTable.	True/False

Table 3 List of questions answered by subjects to verify RT links.

incorrect links. We have on average 48% domain-level and 52% implementation-level SCEs in our case study.

We manually created true RT links to evaluate the correctness of the the subjects' answers. The first two authors manually created traceability links between source code and requirements. The third author verified the manually created links to avoid any bias. A subject answer can either be right or wrong.

We capture subjects' eye movement during the RT link verification task. We define sets of Areas of Interest (AOIs) per pieces of source code as rectangles, each enclosing one (and only one) type of SCEs. Each SCE could either be a domain-level or an implementation-level term. For each AOI, we calculate time by adding the duration of all fixations in that AOI. The average time of fixations shows the amount of time spent by each subject on specific AOI while (s)he focused on that SCE to understand it.

The eye-tracking system captures the fixations at the granularity of line. Therefore, we consider the line that contains the SCE. We divide the total calculated time spent on a SCE by the number of lines of code that the SCE take to obtain an 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.

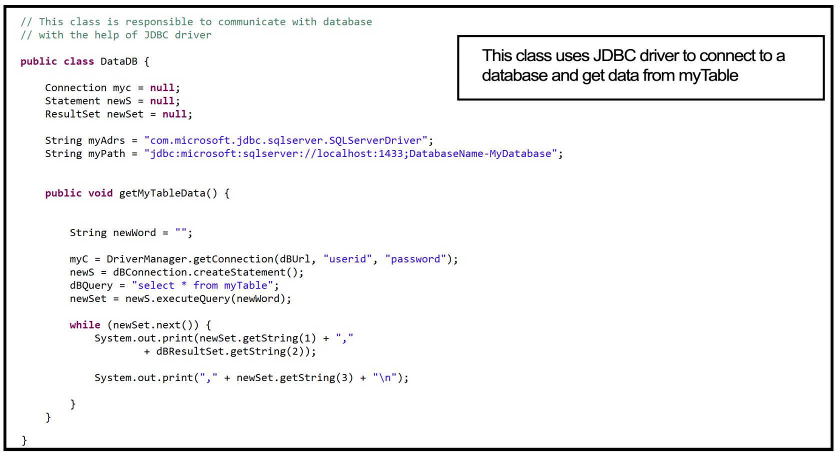
In a pre-experiment questionnaire, we collect data related to the mitigating variables, *e.g.*, number of years of experience in Java. In a post-experiment questionnaire, we ask our subjects to rank the source code entities based on how much they think that these entities help to understand the source code and verify a traceability link.

3.3 Subjects Selection

Out of 26 subjects, there were 4 M.Sc., and 22 Ph.D students who enroll in graduate programs in the Departments of Computer and Software Engineering at École Polytechnique de Montréal and Université 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 [30]. All of the subjects have, on average, 3.39 years of Java programming experience.

There was only one subject for whom we could not have eye-tracking data because of the subject's eyeglasses. Thus, we excluded that subject. We also excluded the subject from a pilot study to validate that the requirements used in the case study are clear and simple and the piece of source code on the screen is easy to read and understand. We analyse the data of a total of 24 subjects.

The subjects were volunteers and they have guaranteed anonymity and all data have been gathered anonymously. The subjects could leave the case study at any



```java
// This class is responsible to communicate with database
// with the help of JDBC driver

public class DataDB {

    Connection myc = null;
    Statement newS = null;
    ResultSet newSet = null;

    String myAdrs = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    String myPath = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName-MyDatabase";


    public void getMyTableData() {


        String newWord = "";

        myC = DriverManager.getConnection(dBUrl, "userid", "password");
        newS = dBConnection.createStatement();
        dBQuery = "select * from myTable";
        newSet = newS.executeQuery(newWord);

        while (newSet.next()) {
            System.out.print(newSet.getString(1) + ","
                    + dBResultSet.getString(2));

            System.out.print("," + newSet.getString(3) + "\n");

        }
    }
}
```

This class uses JDBC driver to connect to a database and get data from myTable

Fig. 1 The source code stimulus.

time, for any reason, without any kind of penalty. The subjects were aware that they are going to perform RT tasks, but did not know the particular case study research questions.

3.4 Objects Selection

We use several criteria to select the pieces of source code used in this eye-tracking case study. We use the Java programming language to perform our case study because it is well known by our subjects. In addition, Java is one of the (many) object-oriented programming languages that contains several different types of SCEs, *i.e.*, class names, method names, variable names, and comments.

We select short pieces of source code that fit in one screen to have better control over the eye-tracking system. These pieces contain one class and one method. The source code size is similar to previous eye-tracking studies [13,42]. The six pieces of source code have 19, 18, 19, 18, 24, and 28 lines of code. Font size is 20. There are four correct (true positive) and two incorrect (false positive) links.

We modify the code to remove any automatically generated source code comments and empty lines are used to distinguish between two types of SCEs. In addition, we mix domain-level and implementation-level terms in the different types of SCEs. For example, a domain term may appear in a method name or a class name. On average, we have 48% domain-level and 52% implementation-level SCEs in source code snippets.

Figure 1 shows one source code stimulus.

3.5 Dependent, Independent, and Mitigating Variables

We have one independent variable: the SCEs (domain and implementation-level source code terms, class name, method name, variable name, and comment). The dependent variables are chosen as follows:

- **Accuracy:** we quantify and measure this variable as the percentage of correct answers given by a subject while reading six pieces of source code to perform RT verification tasks.
- **Average fixation time:** we measure this variable as the amount of fixation time that each subject spends on each type of SCEs. We measure this variable using the eye-tracking system. To compute the average fixation time, we collect data about fixations on the AOIs in each piece of source code shown to the subjects and calculate time by adding the duration of all fixations available in that AOI for each SCE.

Mitigating variables may impact the effect of the independent variables on the dependent variables. In this first case study, we use a questionnaire to collect data about about the following mitigating variables: (1) level of study (B.Sc., M.Sc., or Ph.D.); (2) number of years of programming experience in general; and, (3) number of years of programming experience in Java.

3.6 Procedure

The procedure we followed in our study has mainly four steps. In the first step, we provide single-page guidelines to the subjects to perform the case study. In the second step, we ask subjects to answer the pre-experiment questionnaire to collect data related to the mitigating variables. Before running the case study, we briefly give a tutorial to explain the procedure of the case study and the eye-tracking system (*e.g.*, how it works and what information is gathered by the system).

The subjects are seated approximately $70cm$ away from the screen in a comfortable chair with arms and head rests and the eye-tracking system is calibrated. This process takes less than five minutes to complete. After this, the environment in front of the subjects is just a stimulus.

In the third step, we ask subjects to read the source code, requirements, and assess whether their is a RT link between them. We instruct subjects that they can press space bar key when they find the answer. Pressing the space bar key takes the subjects to a blank screen so that they can write down their answer on the answer sheet. For each question, we ask subjects to spend adequate time to explore the code while we capture subjects' eye movements during traceability verification process. The order of the questions was same for all the subjects.

In the last step, we ask subjects to provide feedback about source code readability and understandability. We also ask subjects which SCEs they prefer more than the others during RT verification task. We ask this question to analyse whether or not subjects followed the same pattern in eye-tracking case study.

3.7 Case Study Results

It took on average 20 minutes to perform the case study including setting up eye-tracking system. No subject mentioned any difficulty regarding the source code readability and understandability.

We perform the following analysis to answer RQ0 and try to reject the null hypothesis. For each subject, we calculate the average fixation time that (s)he spent on the different types of SCEs, *e.g.*, class or method name, in milliseconds.

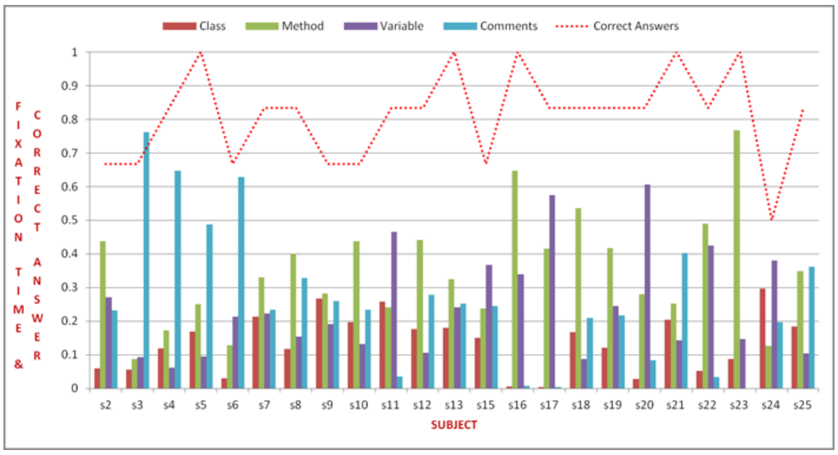


Fig. 2 Distribution of average correct answers and average fixation time.

	Domain	Impl.	**Class**	**Method**	**Variable**	**Comments**	p-value
Average fixation times (ms)	4865.30	1729.80	2317.25	5701.10	3181.81	4542.41	< 0.01
Rankings	5	1	2	6	3	4	–

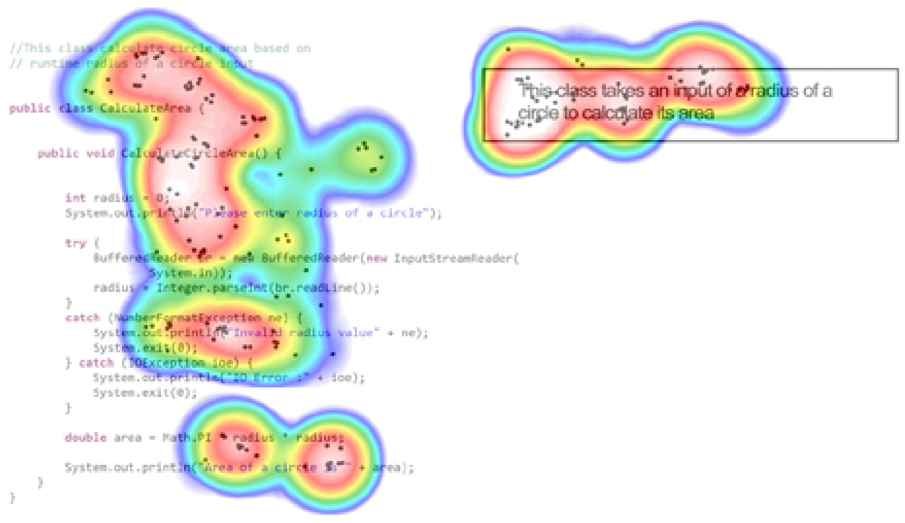
Table 4 Average fixation times spent on each type of SCEs. Rankings of each type of SCEs are based on the average fixation times (1 means the least important).

We consider the time for both wrong and correct answers. Figure 2 shows the distribution of average correct answers and average fixation times for each type of SCEs. The x axis shows the number of subjects while the y axis shows the average time spent on each type of SCEs and average correct answers. To show both average correct answers and time spent on each SCE, we normalise the total amount of data. S1 was the subject of our pilot study and we could not get accurate fixations and saccades for S14 due to eyeglasses. Thus, we excluded S1 and S14 from our study.

We apply Kruskal-Wallis rank sum test on the average fixation times that our subjects spent on four types of SCEs to analyse statistically the subjects' preferences for class names, method names, variable names, and comments. The Kruskal-Wallis rank sum test is a non-parametric method for testing the equality of the population medians among different groups. Table 4 reports that the p-value of the Kruskal-Wallis test for all the SCEs results is statistically significant: the p-value is below the standard significant value $\alpha = 0.05$. The table also reports the ranking of the types of SCEs and whether subjects preferred domain or implementation-level SCEs. It shows that subjects focused more on method names and then comments and also preferred domain-level to implementation-level SCEs.

By replaying the subjects' gazes, using the collected fixations and saccades and their time stamps, we observe that, as soon as the subjects read the requirements, they went directly to read method names and–or comments.

We also analyse the heatmaps consisting of the cumulative fixations of all the subjects on each stimulus. A heatmap is a color spectrum that represents the intensity of fixations. Figure 3 shows the heatmap for one of our subjects working on one of our source codes. It shows a large number of fixations on the method



```
//This class calculate circle area based on
// runtime radius of a circle input

public class CalculateArea {

    public void CalculateCircleArea() {

        int radius = 0;
        System.out.println("Please enter radius of a circle");

        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    System.in));
            radius = Integer.parseInt(br.readLine());
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid radius value" + ne);
            System.exit(0);
        } catch (IOException ioe) {
            System.out.println("IO Error :" + ioe);
            System.exit(0);
        }

        double area = Math.PI * radius * radius;

        System.out.println("Area of a circle is " + area);
    }
}
```

This class takes an input of a radius of a circle to calculate its area

Fig. 3 A heatmap showing the cumulative fixations of subjects. The colors red, orange, green, and blue indicate the decrease in the number and duration of fixations from highest to lowest.

name, comments, and requirement. (The heatmap shows that there is some offset in the position of the fixations and saccades collected using the eye-tracking system. Using Taupe, we managed/adjusted the offsets during the data analysis.) Using the heatmap, we confirm that subjects' visual attention is stronger on SCEs that belong to domain-level concepts than to implementation-level ones. For example, for the Question 1 in Table 3, more fixations are on the "radius" identifier and very few on the "bufRead" identifier. Subjects spent on average $4,865.3$ milliseconds on domain-level SCEs and only $1,729.8$ millisecond on implementation-level SCEs.

The selection of source code snippets with only one SCE per type (one class name, one method name, one paragraph of comments, and one group of variables) could be a direct threat to our results. It is quite possible that developers may have different SCEs preference on larger source code snippets with multiple SCEs. Because we are interested to find the most important SCE based on the time that our subjects spent, we used classes containing only one SCE of each type. Adding more sample of each SCE makes the analysis of the results more complicated because other factors must be considered, such as the relationship between the SCEs. For example, for a particular task, the developers may prefer variable names in large methods than method names. To avoid, such a problem, we chose to have a simple version of each class.

The size of the source code snippets used in this study are comparable to the ones that used in other eye-tracking studies [13,42]. At the end of the eye-tracking study, we ask subjects to answer post questionnaire compare and match with our eye-tracking results. The post-experiment questionnaire about subjects' personal ranking for SCEs to comprehend the source code confirms the results of eye-tracking data.

Thus, we reject H_{01} and answer RQ0 as follows: developers pay different visual attention on different types of SCEs to verify RT links, they prefer method names and comments. In addition, developers almost double their visual attention on domain-level SCEs in comparison with implementation-level SCEs.

4 Semantic Contents of SCEs – An Eye-Tracking Study

The results of the previous eye-tracking case study show that developers prefer some types of SCEs over others while performing RT link verification tasks. However, it is possible that their preferences are due only to the presence of these types of SCEs, not to their semantic content. Hence, developers could prefer method names and comments because they have been taught in their programming classes that method names are important. This situation is similar to the situation in which subjects trust a piece of information by virtue if it is being shown rather than by its intrinsic value, as it has been shown by Pan *et al.* for Google results [36]. Thus, we perform this second case study to answer RQ1: Are some types of SCEs preferred by developers because of their semantic content or because of their very presence in source code?

The goal of this case study, which is summarised with its results in Table 1, is to investigate the impact of the presence of a type of SCEs on subjects' efficiency while performing RT verification tasks. We want to observe whether facilitating the detection of the most preferred types of SCEs (method names) and the least preferred type of SCEs (class names) impact the subject' efficiency. We formulate the following hypotheses to answer RQ1:

H_{02}: There is no significant differences between the accuracy of developers verifying RT links with the source code in which a method name is easier to detect compared to the rest of code.

H_{03}: There is no significant differences between the time spent by developers verifying RT links with the source code in which a method name is easier to detect compared to the rest of code.

H_{04}: There is no significant differences between the effort of developers verifying RT links with the source code in which a method name is easier to detect compared to the rest of code.

We have similar null hypotheses pertaining to class names, the least preferred type of SCEs by developers.

4.1 Case Study Design

The design of this case study is similar to the previous eye-tracking case study that is explained in Section 3.

We use six pieces of source code in which either the method name (Method name in Bold, MiB) or the class name (Class name in Bold, CiB) is shown in bold and in bigger font size (font size = 36) in comparison to the rest of code (font size = 16). The six pieces of source code and questions are different from the ones used in the previous case study. In all six source code snippets, three out of six tasks have highlighted class names and the other three have highlighted method names.

We ask our subjects to manually verify RT links by answering one question per piece of source code. Each question deals with the implementation of a requirement. For example, a piece of source code implements the requirement "PrimeNumCalculator class calculates and prints the prime numbers between 0 and 10,000". A subject can answer "yes" if (s)he thinks that the code is implementing that specific requirement. There are four correct (true positive) and two incorrect (false positive) links. As with the previous case study, we manually created true RT links to evaluate the correctness of the subjects' answers. The first two authors manually created traceability links between source code and requirements. The fourth author verified the manually-created links to avoid any bias. A subject's answer can either be right or wrong.

4.2 Subjects Selection

Our 14 subjects are 2 B.Sc., 3 M.Sc., and 9 Ph.D. students in the Department of Computer and Software Engineering at École Polytechnique de Montréal. Again, most of these subjects performed traceability verification tasks in the past and are representative of junior developers [30]. All of these subjects have, on average, 2 years of Java programming experience.

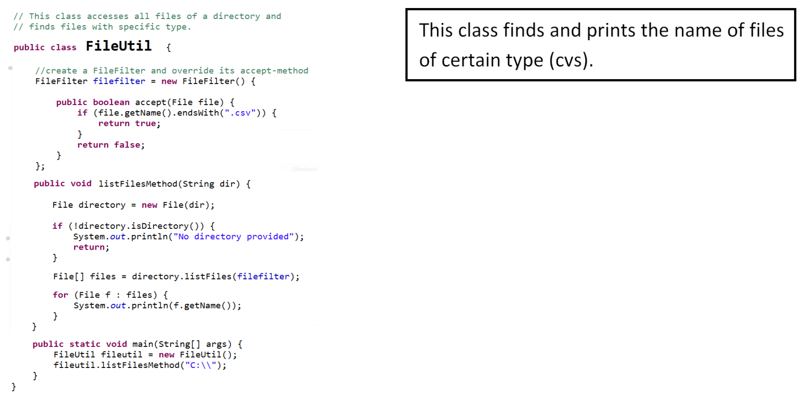
As with the previous case study, these subjects were all volunteers. Some subjects are common to both eye-tracking studies. However, for the second eye-tracking study, subjects do not know the goal of the study and the questions and code snippets are different.

4.3 Objects Selection

We use the same criteria and the same programming languages as in the previous case study to choose the pieces of code that make up the stimuli. The lengths of the pieces of code (in lines of code) are 28, 29, 19, 27, 33, and 35, respectively. In this case study, for each piece of source code, there is one instance of each type of SCEs. Thus, each piece of source code contains one class name, one method name, one comment, and one set of variables. Figure 4 shows an example of a source code stimulus CiB.

4.4 Dependent, Independent, and Mitigating Variables

The type of SCEs (class name or method name) is the independent variable. We select two different sets of dependent variables to compare and complement the results of the first study. Similar to the first eye-tracking study, the first set of dependent variables contains the subjects' average fixation times and accuracies. In the second set of dependent variables, because we are interested to investigate the impact of the presence of specific type of SCEs on subjects' efficiency, we put the following variables:



```java
// This class accesses all files of a directory and
// finds files with specific type.

public class FileUtil {

    //create a FileFilter and override its accept-method
    FileFilter filefilter = new FileFilter() {

        public boolean accept(File file) {
            if (file.getName().endsWith(".csv")) {
                return true;
            }
            return false;
        }
    };

    public void listFilesMethod(String dir) {

        File directory = new File(dir);

        if (!directory.isDirectory()) {
            System.out.println("No directory provided");
            return;
        }

        File[] files = directory.listFiles(filefilter);

        for (File f : files) {
            System.out.println(f.getName());
        }
    }

    public static void main(String[] args) {
        FileUtil fileutil = new FileUtil();
        fileutil.listFilesMethod("C:\\");
    }
}
```

This class finds and prints the name of files of certain type (cvs).

Fig. 4 Example of a source code stimulus in which the class name is written in bold using a larger font size.

- **Total fixation time:** we measure this variable as the amount of time that each subject spends on the source code stimuli. We use again the data provided by the eye-tracking system to measure this value for each stimuli separately.
- **Effort**: we use the Average Fixation Duration (AFD) metric for calculating the subjects' visual effort using data that are provided by the eye-tracking system. The formula for calculating AFD is presented in Equation 1, where $ET(F_i)$ and $ST(F_i)$ represent the end time and start time for a fixation F_i and n represent the total number of fixations in the stimulus. Longer fixations mean that our subjects spend more time looking at that stimuli. Therefore, stimuli that require shorter fixations are more efficient than the ones with longer fixations [16].

$$AFD(Stimulus) = \frac{\sum_{i=1}^{n}(ET(F_i) - ST(F_i))\text{Stimulus}}{n} \tag{1}$$

The mitigating variables are the same as in the previous case study and we reuse the same questionnaire to collect data about about the following mitigating variables: (1) level of study (B.Sc., M.Sc., or Ph.D.); (2) number of years of programming experience in general; and, (3) number of years of programming experience in Java.

4.5 Procedure

The procedure of this case study is identical to that of the previous case study. The subjects and stimuli are different but the collected data are identical: fixations and saccades as well as data related to the mitigating variables and the subjects' preferred types of SCEs.

		Class	**Method**	**Variable**	**Comments**	p-value
CiB	**Average fixation times (ms)**	1580.5	4351.167	905.3947	1003.593	< 0.01
	Rankings	3	4	1	2	–
MiB	**Average fixation times (ms)**	882.3333	4088.5	1428.517	1654.833	< 0.01
	Rankings	1	4	2	3	–

Table 5 Average times and rankings for CiB and MiB source code, spent on each type of SCEs. Rankings of each type of SCEs are based on average fixation times (1 means the least important).

	MiB		**CiB**	
Accuracies (Std dev.)	0.78	(0.41)	0.85	(0.34)
Total fixation times (ms) (Std dev.)	20,134.38	(9664.29)	21,164.31	(8136.51)
Efforts (Std dev)	78.99	(6.63)	76.95	(6.81)

Table 6 Accuracies, total fixation times, and effort that our subjects spent on method names.

4.6 Case Study Results

By comparing Tables 4 and Table 5, we observe that if method names are highlighted, then subjects read immediately these method names and then read comments and, finally, class names. If class names are highlighted, then subjects read them first and then read method names, spending more time over the method names. In addition, the results show that changing font color/size might yield different AFD for each SCE. However, more case studies are required to explain and generalize this last observation.

Table 6 shows the difference of subjects' accuracies, total fixation times, and efforts on MiB and CiB code variants. We test our hypotheses to find any potential advantage of CiB over MiB. After applying the non-parametric unpaired Wilcoxon test with $\alpha = 0.05$, the p-values report that there is no significant difference between the two code variants for accuracy, total fixation time, and effort. We thus cannot reject any of our null hypotheses, H_{02}, H_{03}, or H_{03}. We use non-parametric unpaired Wilcoxon statistical test to determine significance because our data is not normally distributed.

We consider and compute the total fixation time that our subjects spent to perform the whole task and effort that are required for reading the code to understand it. Our analysis of the heatmaps of the subjects' fixations for each stimuli show that although we highlight class or method name, our subjects read the code completely and spent enough time on all types of SCEs to understand them and perform RT verification tasks. These results and observations emphasize that one types of SCEs does not lead and help our subjects to read the code and find the answers more effectively.

In conclusion, this second case study confirms the results of the previous one: method names are the most preferred SCEs. It also shows that whether method names or class names are highlighted, subjects spend more time on method names, with equivalent accuracy, total fixation time, and effort. It is indeed the content of the method names, and not just their presence, that subjects seek to perform the RT verification tasks.

However, these results suggest further case studies with longer pieces of code and more types of SCEs to understand the subjects' reading time and effort. Such study would require a different setting because eye-tracking systems are notoriously difficult to use when subjects can scroll text/images. Other case studies should also be performed with different means of highlighting class and method names because there is a complex interplay between the form and semantics of the SCEs. More sophisticated highlighting techniques may produce different results than observed in this study.

We cannot reject H_{02}, H_{02}, and H_{02} and answer RQ1 as follows: whether method names or class names are highlighted, subjects spend more time on method names, with equivalent accuracy, total fixation time, and effort. It is thus the content of the method names, and not just their presence, that subjects seek to perform the RT verification tasks.

5 *DPTF/IDF* Weighting Scheme

The results of the two previous case studies show that, while performing RT tasks, subjects preferred certain types of SCEs (method names) and that it is likely the semantic content of these SCEs, not their form or location that positively impact the subjects preferences. From this observation, we infer that the semantic content of the subjects' preferred type of SCEs could be used to improve an automated RT recovery technique as follows. Most RT recovery techniques use the semantics of words in the high-level and low-level documents to create links, such as LSI. Therefore, because subjects preferred certain types of SCEs based on their semantic content, we conjecture that giving more importance to these types of SCEs in a RT recovery technique, should improve the accuracy of the technique.

This section describes our weighting scheme *DPTF/IDF*, *i.e.*, **D**evelopers' **P**reference-based **T**erm **F**requency and **I**nverse **D**ocument **F**requency to improve the accuracy of an IR technique. The weighting scheme rewards a term based on its type whereas state-of-the-art IR-based weighting schemes give importance to a term based on its probability or frequency in a corpus. Generally, our proposed *DPTF/IDF* weighting scheme allows developers to tune the weighting scheme to increase or decrease the importance of any type of term in a corpus.

In our previous work [7], we proposed the *DOI/IDF* weighting scheme to calculate term weights based on their location in source code. To use this scheme, a developer must tune up eight parameters by manually selecting a value for each parameter. Thus, an oracle is required to actually compute its parameter values [7]. However, while dealing with real, legacy systems, no such oracle exists, *i.e.*, no priori-known set of traceability links exists. Moreover, using the same parameter values [7,38] or giving a fixed reward $\mathcal{R}_k$ to terms in different part of a system may not be beneficial for all recovered links. The *DOI/IDF* is only able to weight up to six types of SCEs. For example, if a developer wants to use each line of source code as a type of SCEs then *DOI/IDF* would not be able to handle so many SCEs. In the following, we propose an improved version of *DOI/IDF*, *i.e.*, *DPTF/IDF*, that can deal with any number of SCEs. The *DPTF/IDF* is easy to tune with the DynWing without developer's intervention.

5.1 Defining *DPTF/IDF*

State-of-the-art 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 (a document can either be a requirement or a piece of source code). 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:

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

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.

DPTF/IDF recognises that not all terms have equal importance. It checks for the type of a term in the source code to compute its importance, to reward the term. For example, if a term appears in method name, it could be given more importance than a term that appears in comments. A term would be highly rewarded if it appears in two (or more) preferred types of SCEs in the source code.

In *DPTF/IDF*, we represent a traceability link as a triple (source document, target document, similarity) and we use the following notations. Let $R = \{r_1, \ldots, r_N\}$ be a set of requirements and $\mathcal{C} = \{c_1, \ldots, c_M\}$ be a set of classes supposed to implement these requirements. Following Bunge's ontology [14], 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.

We consider a class as a collection of all types of SCEs, *i.e.*, all possible sources of information. Each source of information, such as comments, method names, or local variables possibly contribute in different ways to provide traceability evidence. To define the SCEs, let ψ_k be a family of functions $k = 1, \ldots N$. Each function selects a sub-set of C elements properties, for example, for a given class c_j (our document), a given ψ_k extracts the method names. In other words, each ψ_k function creates a new set of documents. By means of ψ_k, *DPTF/IDF* partitions X[5] properties into class name (CN), method name (MN), variables name (VN), comment (CMT), implementation-level (A), and domain-level (D) terms.

In the two previous case studies, we find that method names are the most important source of information for developers. However, given two links in two different classes for two different requirements, the relative importance of methods names with respect to, for example, class names may change. Let $\mathcal{R}_k$ be a set of reward functions assigning a positive integer when a term, t_i, in a class c_j belongs to the projection ψ_k. This is to say $\mathcal{R}_k(\psi_k(c_j), t_i)$ implements the ranking defined in Table 4: it returns values in N. For example, if a term appears in a method name, then $\mathcal{R}_k$ would use the value 6 as reward for that term. In this paper, we use two methods to calculate the reward: (1) we use developers preferences (see Table 4) to decide and choose the reward and (2) we use DynWing [6], *i.e.*, a dynamic reward calculator. However, it may also happen that a term belongs to comments as well as being used as a variable name. The *DPTF/IDF* implements an additive reward mechanism thus, if a term is a variable name and it is also included in a

[5] We consider any object X is a source code class, *i.e.*, c_i.

comment, it is rewarded by two $\mathcal{R}_k$ dynamically added on per link basis:

$$DPTF_{i,j} \;=\; TF_{i,j} \;\times\; \sum_k \lambda_k \; \mathcal{R}_k(\psi_k(c_j), t_i)$$

where λ_k are the parameters adjusted to maximize $DPTF/IDF$ and, thus, maximize $DPTF/IDF$, where IDF is the inverse document frequency, which weighs how often a term is encountered in the corpus 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.

Thus, we define $DPTF/IDF$ as follows:

$$DPTF/IDF_{i,j} = DPTF_{i,j} \times IDF_i$$

5.2 Calculating Parameter Values (λ_k)

To automatically calculate the values of the parameters, *i.e.*, λ_k, we apply our previous approach for dynamic tuning, DynWing [6]. We use DynWing to dynamically combine different sources of information, *e.g.*, software repositories [6]. We apply the same idea using DynWing to automatically tune parameters values, *i.e.*, λ_k, of $DPTF/IDF$.

Existing weighting schemes, *e.g.*, DOI/IDF [7], use static parameter values to compute term weights [5,38]. For example, MN could be chosen to have an importance of $\lambda = 0.4$ and CN to have an importance of $\lambda = 0.3$. These static weighting schemes require oracles to decide a "good" set of parameter values or an acceptable range of values. Furthermore, these static weighting schemes cannot dynamically reward terms of the same type in different ways: once the parameter value associated with class names is chosen, there is no possibility to adapt it to the class or document under analysis.

In $DPTF/IDF$, we consider each type of SCEs as an individual entity and dynamically assign importance to each of them. Choosing the right parameter value for each SCE is a problem that we formulate as a maximisation problem. Each SCE has its own importance in the source code. By maximizing the term importance value (and hence determining the optimal λ_k values), the DynWing-based approach automatically identifies the SCEs that are most trustworthy (highest λ_k values) and less trustworthy (lowest λ_k values) in source code.

The goal of our weighting scheme is to reward more to a term if it appears in a particular SCE, *e.g.*, method name. First, for each SCE, we compute term's weight using stat-of-the-art TF weighting scheme. Second, we pass all the TF values to DynWing to compute λ_k. DynWing uses different combinations of values to automatically adjust the λ_k, where it maximizes the term's weight. For example, if a term in a method name has 0.3 TF and a term in a class name has .4 TF then DynWing will assign 0.9 to method TF and 0.1 to class TF. DynWing will keep using different λ_k values until it finds a combination that gives high weight to the term in the method name. DynWing uses following constraints to calculate λ_k:

$$\max_{\lambda_1,\dots,\lambda_N} \{DPTF_{i,j}\} \tag{2}$$

with the following constraints:

$$\begin{aligned} 0 \le \lambda_k \le 1, k = 1, ..., N \\ \lambda_1 + \lambda_2 + ... + \lambda_N = 1 \\ \lambda_{k_1} \ge \lambda_{k_2} \ge \ldots \ge \lambda_{k_N} \end{aligned} \tag{3}$$

Given the three previous constraints, it is possible that the DynWing-based approach assigns $\lambda_k = 1$ to a single SCE. To avoid such an assignment, a developer can define a global term importance. For example, MN may be considered by the developer more trustworthy than CN. Therefore, the developer may constrain further Equation 2 by imposing the following constraint:

$$\lambda_{MN} \ge \lambda_{CN} > 0$$

6 Impact of *DPTF/IDF* - A RT Study

We perform an empirical study to analyse whether the $DPTF/IDF$ weighting scheme provides better results than the state-of-the-art TF/IDF weighting scheme and our previously proposed DOI/IDF weighting scheme. We perform a case study on three datasets, iTrust, Lucene, and Pooka, to analyse how much $DPTF/IDF$ improves the F-measure of an RT link recovery technique that uses it, as summarised in Table 2.

6.1 Case Study Design

To answer RQ2: Does using $DPTF/IDF$ allow developing a RT links recovery technique with better accuracy than a technique using TF/IDF? and RQ3: Does using $DPTF/IDF$ allow developing a RT links recovery technique with better accuracy than a technique using DOI/IDF?; we build three sets of traceability links using three different term weighting schemes, *i.e.*, TF/IDF, $DPTF/IDF$, and DOI/IDF.

We formulate the following two null hypotheses related to RQ2 and RQ3:

H_{04}*:* There is no difference between the accuracy of a LSI-based technique using TF/IDF and using $DPTF/IDF$ in terms of F-measure.

H_{05}*:* There is no difference between the accuracy of a LSI-based technique using DOI/IDF and using $DPTF/IDF$ in terms of F-measure.

Consequently, we define three sets of RT links: $LSI_{TF/IDF}$, $LSI_{DOI/IDF}$, and $LSI_{DPTF/IDF}$. The first set, $LSI_{TF/IDF}$, contains the RT links recovered by a technique based on LSI and using the standard TF/IDF weighting scheme. The second and third sets, $LSI_{DOI/IDF}$ and $LSI_{DPTF/IDF}$ are the sets of RT links obtained by a technique also based on LSI but using either the DOI/IDF or $DPTF/IDF$ weighting schemes.

Then, we perform two comparisons to assess the potential improvement in the accuracy of the recovered RT links brought by the $DPTF/IDF$ weighting scheme: (1) $LSI_{TF/IDF}$ vs. $LSI_{DPTF/IDF}$ and (2) $LSI_{DPTF/IDF}$ vs. $LSI_{DOI/IDF}$. With these comparisons, we can understand the importance of various types of SCEs for RT links recovery.

6.2 Objects and Oracles

We use three criteria to select the systems that are used in our case studies. First, we select open-source systems, *i.e.*, iTrust[6], Lucene[7], and Pooka[8], because their source code is freely available thus, other researchers can replicate our case studies. Second, we avoid small systems that do not represent real world systems usually handled by developers. Third, we choose small enough systems to manually create oracles ("true" RT links).

iTrust (v10.0) is a medical application that provides patients with the means to keep up with their medical history and records. It helps patients to communicate with their doctors, including selecting which doctors to be their primary care giver, and seeing and sharing satisfaction results. iTrust also allows the medical 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. iTrust is developed in Java and it has 526 source code files and 35 requirements. We did not create any traceability for iTrust. We used the same traceability matrix as provided online[6] by the original developer(s).

Lucene (v2.9) is an open-source, high-performance, text-based search-engine library that is written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. We select Lucene $v2.9$ because it contains more feature requests than other versions and these are linked to its SVN repository. Lucene has 413 source code files and 116 new feature requests. We download new feature requests and SVN logs from Lucene JIRA repository. Developers usually write feature IDs in SVN logs when they add–or modify a new feature [10]. We automatically extract feature IDs from SVN logs to link newly requested feature to the related classes. For example, if a developer committed some files in SVN and mentioned feature ID in the SVN log then we link particular feature to those committed files. We use new feature requests as new requirements. Using these requirements and knowing the classes that were added to the SVN to fulfill them, we build the set of "true" RT links, $Oracle_{Lucene}$, which consists of 744 links.

Pooka (v2.0) 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. Pooka version 2.0 has 298 source code files and 90 requirements. We reuse these 90 functional requirements recovered in our previous work [5]. We use

[6] `http://agile.csc.ncsu.edu/iTrust`

[7] `http://lucene.apache.org/`

[8] `http://www.suberic.net/pooka/`

these requirements to manually create traceability links between requirements and source code. Two of the authors created the RT links and the third author verified all the links to accept or reject them. The true RT links, $Oracle_{Pooka}$, consists of 546 verified links.

6.3 Dependent, Independent, and Mitigating Variables

We use the sets of recovered RT links using the three techniques as independent variables, this variable has three possible values, which correspond to the three sets of RT links: $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$.

We use the F-measure as a dependent variable to empirically attempt to reject the null hypotheses. We compute F-measure of the three sets of RT links in comparison to $Oracle_{Lucene}$, $Oracle_{Pooka}$ and $Oracle_{iTrust}$. We use the F-measure because we want to know if the proposed technique improves both precision and recall. The F-measure is the harmonic mean of precision and recall computed as:

$$F = \frac{2}{\frac{1}{R} + \frac{1}{P}}$$

where R is the recall, P is the precision, and F is the harmonic mean of R and P (thus, relative to the j^{th} document in the ranking). The function F assumes that values are in the interval $[0, 1]$. The value is equal to 0, when no relevant documents have been retrieved and it is equal to 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, finding the maximum value for the F is an attempt to find the best possible compromise between precision and recall [4].

6.4 Procedure

The following paragraphs detail the steps to obtain the RT links, $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$, and to compute their respective F-measure values with respect to the oracles, $Oracle_{iTrust}$, $Oracle_{Lucene}$, and $Oracle_{Pooka}$.

Extracting Concepts:

If a term belongs to the domain concepts then $\mathcal{R}_k$ (see Section 5.1) will assign more weight to that term than to term that is an implementation concept. In this paper, we use LDA [2,28,35] to separate terms likely representing domain concepts. 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. We consider two topics: domain-level terms and implementation-level terms. LDA takes three parameters α, β, and k to create the topics: α is the Dirichlet hyper-parameter for topic proportions, β is the Dirichlet hyper-parameter for topic multinomials, and k is the number of topics. More details on using LDA to extract domain concept are available in previous work [11,22,26,35,37].

In this case study, as in previous work [2], 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. LDA also uses a threshold to decide whether or not a term belongs to a topic. We use 0.01 as threshold as suggested by Abebe *et al.* [2] who used three different threshold points, *i.e.*, 0.01, 0.02, and 0.03, to assess the effect of terms having different levels of probabilities in representing the topics. Their results showed that 0.01 threshold provides better results than 0.02 and 0.03.

To concretely separate domain-level and implementation-level terms, we use an implementation of LDA in Java, *i.e.*, MALLET[9]. We look at each two topics to see if one was closer to domain concepts. We read the requirements and look at each two topics to see if one had more requirement-related terms. We consider the topic with the most requirement-related terms as containing domain-level terms.

LDA is not yet a perfect technique to accurately distinguish domain-level from implementation-level terms. However, to the best of our knowledge, LDA is a widely used and stable approach to extract concepts. Thus, we use the results of LDA as they were provided by the tool. The results of LDA could impact the accuracy of our proposed weighting technique. In future, other advanced domain concept extraction techniques could be integrated in our proposed term weighting scheme. However, better concept extraction techniques could only improve the quality of the results generated by our weighting scheme and, thus, we consider that our use of LDA is conservative.

Generating Corpora:

To process source code, we use Java parser [4] 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 types of SCEs and the textual information is stored in their respective source code files. We use these files to apply proposed weighting schemes to create RT links.

Pre-processing the Corpora:

We perform standard state-of-the-art [34] 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.

Setting the Global Constraints:

The proposed weighting scheme requires to choose some global constraints (see Equation 3). We use the ranking based on the fixation time (see Table 4), observed during eye-tracking case study, to tune Equation 3. We analyse that subjects give more preference (spent more time) to MN then D, CMT, VN, CN, and A. We also observed that our subjects looked at domain-level and implementation-level SCEs 74% and 26% of their time respectively. Thus, we define global constraint in Equation 3 is as follows:

$$\lambda_{MN} > \lambda_{D} > \lambda_{CMT} > \lambda_{VN} > \lambda_{CN} > \lambda_{A} > 0$$

Creating the RT Links:

We use LSI [34] to create the three sets $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$. LSI is an information retrieval technique based on the VSM. For

[9] http://mallet.cs.umass.edu

	$k = 30$	$k = 50$	$k = 100$
Pooka	9.02	9.52	9.05
iTrust	16.82	17.76	17.27
Lucene	14.54	17.01	16.13

Table 7 F-measure values using different k values for LSI.

each requirement, LSI generates a ranked list of classes. It assumes that there is an underlying or latent structure in word usage for every document in a corpus [34]. The processed corpus is transformed into a term-by-document matrix. The values of the matrix cells represent the weights of the terms in the documents, which are computed using the weighting schemes: TF/IDF, $DPTF/IDF$, and DOI/IDF.

The matrix is then decomposed using *Singular Value Decomposition (SVD)* [34] to derive a particular latent-semantic structure model from the term-by-document matrix. In SVD, each term and artifact could be represented by a vector in the k space. The choice of k value, *i.e.*, the SVD reduction of the latent structure, is critical and still an open issue in the natural-language processing literature. We want a value of k to be large enough to fit all the real structures in the data but small enough so we do not also fit the sampling error or unimportant details. The selection of a "good" k value requires an oracle of traceability links and, therefore, we use the oracle as reference to choose the k value. We find that $k = 50$ provides better accuracy than the other k values for all three datasets. Table 7 shows the F-measure results of different k values.

Once all requirements and source code documents have been represented in the LSI sub-space, we compute the similarities between requirements and classes using cosine similarity.

Computing the F-measure Values:

We use a threshold t to prune the set of traceability links, keeping only links whose similarity values are greater than t. 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, recall, and F-measure 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 at all the t. Then, we perform statistical test to assess whether the differences in F-measure values, in function of t, are statistically significant between $LSI_{TF/IDF}$ vs. $LSI_{DPTF/IDF}$ and $LSI_{DPTF/IDF}$ vs. $LSI_{DOI/IDF}$.

Analysing the Differences in F-measure Values:

To select an appropriate statistical test, we use the Shapiro-Wilk test to analyse the distributions of our data points. We observe that these distributions do not follow a normal distribution. Thus, we use a non-parametric test, *i.e.*, Wilcoxon test, to test our null hypotheses. We use the $\alpha = 0.05$ to accept or refute null hypotheses. Because we have three comparisons, we apply Bonferroni correction by dividing 0.05 by 3 and rounding α to 0.01

An improvement might be statistically significant but it is also important to estimate the magnitude of the difference between the accuracy levels achieved with $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$. We use a non-parametric effect size measure for ordinal data, *i.e.*, Cliff's d [18], to compute the magnitude of the effect

	Technique	Precision	Recall	F-Measure	Effect Size	p-Value
Pooka	$LSI_{TF/IDF}$	14.71	21.07	9.52		
	$LSI_{DOI/IDF}$	25.73	26.42	14.56	0.98	< 0.01
	$LSI_{DPTF/IDF}$	25.86	27.48	15.15	0.98	< 0.01
iTrust	$LSI_{TF/IDF}$	36.43	33.14	17.76		
	$LSI_{DOI/IDF}$	39.55	35.07	20.64	0.90	< 0.01
	$LSI_{DPTF/IDF}$	40.24	37.34	22.52	0.91	< 0.01
Lucene	$LSI_{TF/IDF}$	24.27	42.28	17.01		
	$LSI_{DOI/IDF}$	29.36	39.46	17.48	0.13	< 0.01
	$LSI_{DPTF/IDF}$	27.70	43.74	18.47	0.47	< 0.01

Table 8 Average values and Wilcoxon p-values. We performed Bonferroni correction due to multiple comparisons.

		F-Measure	Effect Size	p-Value
Pooka	$LSI_{DOI/IDF}$	14.56	0.67	< 0.01
	$LSI_{DPTF/IDF}$	15.15		
iTrust	$LSI_{DOI/IDF}$	20.64	0.61	< 0.01
	$LSI_{DPTF/IDF}$	22.52		
Lucene	$LSI_{DOI/IDF}$	17.48	0.11	< 0.01
	$LSI_{DPTF/IDF}$	18.47		

Table 9 Average F-Measure values of $LSI_{DOI/IDF}$ and $LSI_{DPTF/IDF}$.

of $DPTF/IDF$ on precision and recall as follows:

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

where x_1 and x_2 are F-measure values with $LSI_{TF/IDF}$, $LSI_{DPTF/IDF}$, and $LSI_{DOI/IDF}$, and n_1 and n_2 are the sizes of the sample groups. The effect size is considered small for $0.15 \leq d < 0.33$, medium for $0.33 \leq d < 0.47$ and large for $d \geq 0.47$ [33].

6.5 Results

Figure 5 shows the F-measure values of $LSI_{TF/IDF}$ and $LSI_{DPTF/IDF}$. It shows that $LSI_{DPTF/IDF}$ provides better F-measures at all the different values of threshold t. Figure 5 shows that giving different importance to SCEs, depending on their role and occurrence in source code, provides better accuracy. Table 8 shows that $LSI_{DPTF/IDF}$ statistically improves, on average, up to 11%, 6%, and 5.63% precision, recall, and F-measure, respectively.

We perform the Wilcoxon test on all F-measure values to analyse if the improvements are statistically significant. In all the cases, $DPTF/IDF$ statistically improves the F-measure values with large effect size.

We have statistically significant evidence to reject the H_{04} hypothesis for all datasets' results. Table 8 shows that the p-values, with Bonferroni correction, are below the standard significant value, *i.e.*, $\alpha = 0.05$. Thus, we answer RQ2 as follows: integrating the developers' knowledge, *i.e.*, SCEs preferences, in an IR technique statistically improves its accuracy.

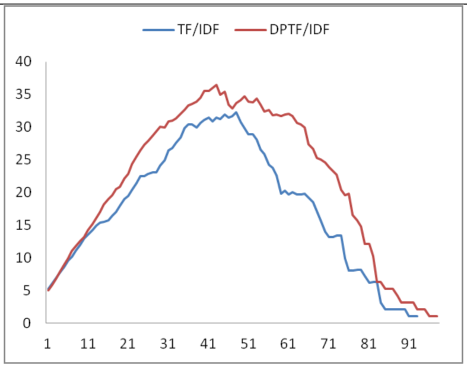
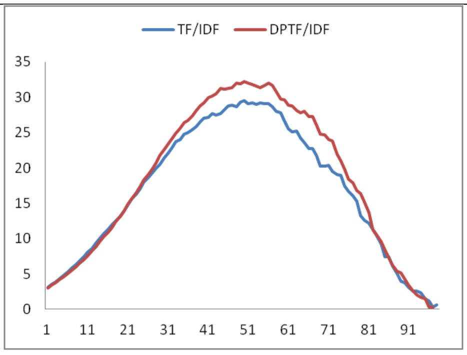
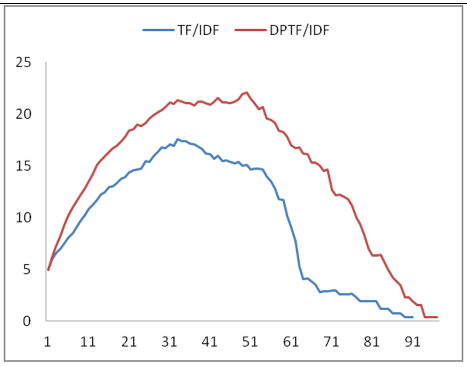
Dataset	
iTrust	
Lucene	
Pooka	

Fig. 5 F-measure values of $LSI_{TF/IDF}$ and $LSI_{DPTF/IDF}$.

Table 9 shows that results are slightly improved for all three datasets when compared to DOI/IDF. Figure 6 shows, that most of the times, $LSI_{DPTF/IDF}$ provides better F-measure values. In the case of Lucene, only in almost 34% of the cases, $LSI_{DPTF/IDF}$, was slightly lower then $LSI_{DOI/IDF}$. However, Table 9 shows that, on average, $LSI_{DPTF/IDF}$ provides better results than $LSI_{DOI/IDF}$.

We perform the Wilcoxon test to analyse the improvements in F-measure values using the $DPTF/IDF$ and DOI/IDF. Table 9 shows that $LSI_{DPTF/IDF}$ provides better results than $LSI_{DOI/IDF}$. Table 9 shows that using $LSI_{DPTF/IDF}$, for Lucene, the effect size goes from no improvement to large a improvement. For

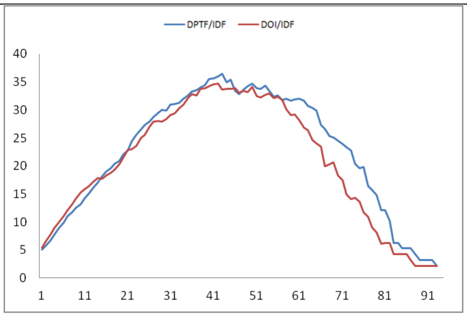
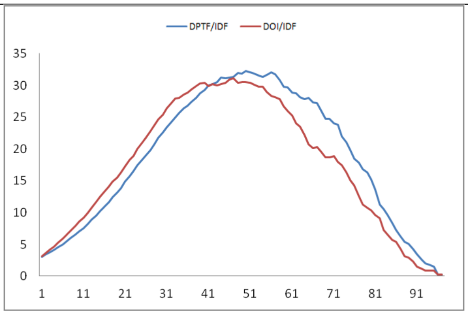
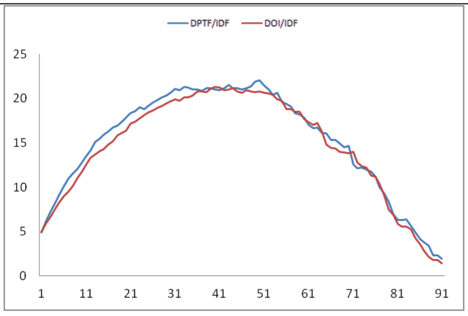
Dataset	
iTrust	
Lucene	
Pooka	

Fig. 6 F-measure values of $LSI_{DOI/IDF}$ and $LSI_{DPTF/IDF}$.

Pooka and iTrust, there is not much improvement in the effect size. It is possible that new and more advanced parameter tuning technique to compute the λ_k values would provide even better results.

We have statistically significant evidence to reject the H_{05} hypothesis for all datasets. Table 9 shows that the p-values are below the standard significant value, *i.e.*, $\alpha = 0.05$. Thus, we answer RQ3 as follows: $DPTF/IDF$ provides better results than DOI/IDF. DynWing automatically tune $DPTF/IDF$ parameters and it does not require a developer to tune the parameter for each dataset separately.

The understanding that we gained by answering RQ2 and RQ3 is important from the point of view of both researchers and practitioners. For researchers, our results bring further evidence to support our conjecture that different types of

SCEs must be given different importance, according to the developers' preferences. For practitioners, our results provide concrete evidence that they should pay attention to terms, in particular method names, to help developers perform RT tasks. In addition, we show that using domain-level terms in source code helps developers using an IR technique to create RT links with better accuracy than using implementation-level terms.

7 Discussion

In response to the research questions defined in Section 1, we observe that developers have different preferences for different types of SCEs. In summary, during RT manual verification tasks, developers pay more attention to method names and their semantic content. We observe that, as soon as developers read a requirement, they started looking for specific methods of interest and read their comments. We believe that developers least bothered with the class names because one class may contain several functionalities [4]. Similarly, we believe that developers paid less attention to variable names because variable names could be object names of other classes. Moreover, Figure 2 shows that a high number of fixations on variable names lead to more wrong answer.

To avoid the bias of eye-tracking observations, we ask our subjects through a post-experiment questionnaire about the type of SCEs that helped them to verify RT links. The results of cross verification questions are in agreement with the observations that we made with the eye-tracking studies. This step gives us confidence in our findings and mitigates the threats to validity.

In regular text documents' zoning, researchers mentioned that the title of a document is important [25,31,44]. If we map a document structure into source code structure then class name is equal to a document title. In our case study, we analyse that developers do not give preference to class name. We change our main constraint in Equation 3 and observe that it decreases the F-measure values. In some cases, putting comments more important than the other types of SCEs provides better recall but lower precision. To compare with normal regular textual documents' zoning, we give more importance to class name and observe that it decreases the F-measure value. Thus, we conclude that a document is not the same as the source code. In addition, MN and domain-level terms play an important role in manual and automatic RT links recovery tasks.

We used our eye-tracking observations in the $LSI_{DOI/IDF}$ and $LSI_{DPTF/IDF}$ techniques to analyse how much it improves its accuracy when compared to the $LSI_{TF/IDF}$ technique. Table 8 shows that observing developers during 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 RT techniques.

We observe that $DPTF/IDF$ is less effective on iTrust and Lucene than on Pooka. We investigate the reason, we find that in iTrust, many classes do not contain all types of SCEs, *i.e.*, class, method, variable name, and comments. In the case of Lucene, we observe that LDA identified 9% more implementation-level SCEs than domain-level ones. In the case of iTrust, LDA identified 3% more domain-level SCEs terms. In the case of Pooka, LDA identified 14% more domain-level SCEs than implementation-level SCEs. Thus, we could summarise that if a dataset contains all types of SCEs and more domain-level SCEs then, the proposed $DPTF/IDF$ weighting scheme performs better accuracy in terms of F-measure. However, even in the case of iTrust and Lucene, it has less domain-level SCEs and fewer classes containing all types of SCEs; the proposed weighting scheme still provides better accuracy.

We set the parameter values (λ_k) of our weighting schemes based on the observations that we made during our eye-tracking case study. $DPTF/IDF$ is completely automatic once you set preferences for the global constraints defined in 3. It is quite possible that using other values for λ_k may yield different results. However, with the current parameters, we observe that proposed weighting schemes outperform traditional TF/IDF.

We also made some interesting observations as follows.

7.1 Identifiers' Vocabulary

We observe that when a similar term appears in both requirement and source code, it helps developers to provide better accuracy than when different terms refer to the same idea. RT link verification and recovery techniques are not 100% perfect yet and a developer must verify the recovered links that are provided by automated techniques and–or create the missing links. We observe that if the method names are not meaningful, then a developer may skip the whole method. However, it is quite possible that the method is implementing the requirement. We only observe this situation for method names because there is only one class per requirement. It is quite possible that, if there are meaningless class names, then a developer may skip the whole class as well without reading the whole source code. We thus suggest that developers must work closely with requirement engineers to understand the requirements and use the requirements' terms to name the SCEs.

The identifiers' vocabulary does not make any threat to the validity of our study. In our study, the meaningless SCEs were implementation-level SCEs that we take into account in our case study.

We observe that domain-level SCEs are developers' focal points during RT creation tasks. We record more fixations on domain-level terms. Using implementation-level terms as identifier names may help developers to remember them and–or their data type. However, implementation-level identifiers do not help a lot in RT manual and automated tasks. Thus, we give more importance on implementation-level identifiers and observed decrease in F-measure value.

7.2 Male vs. Female

The main purpose of the study was not to compare the male and female subjects during RT recovery. However, we make interesting observations that may lead to future case studies on gender effect on RT tasks. There are 7 female subjects and 17 male subjects in our study. We observed that female subjects were more interested in details than male subjects. For example, female subjects directly look at the implementation of the source code without reading the comments or

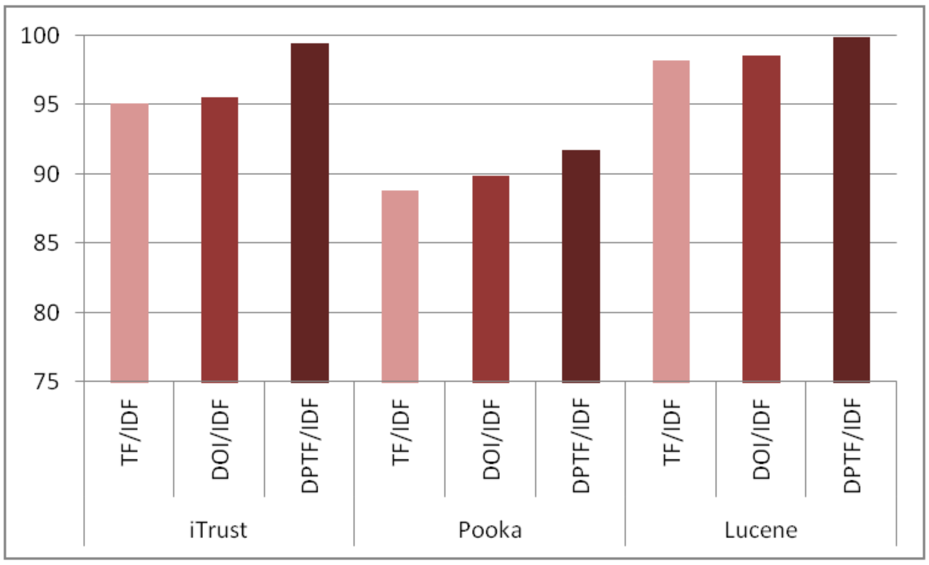


Fig. 7 New Links Recovered using $DPTF/IDF$.

method names. They spent more time reading variable names and understanding them. Whereas, male subjects are more concerned about the high-level structure of the source code. They preferred more method names and comments. However, we cannot generalise this observation. More detailed studies are required to analyse gender difference on RT tasks.

7.3 Missing Links Analysis

It could be relatively easy to verify a link than recovering a missing link. Thus, recall plays an important role in RT. We observe that $DPTF/IDF$ recovers up to 4.37% more links that have been missed by TF/IDF. Figure 7 shows the recall of TF/IDF and $DPTF/IDF$. In the case of iTrust and Pooka, we find that $DPTF/IDF$ tries to recover more missing links. For Lucene, the percentage of missing links is already less than 2%. However, $DPTF/IDF$ still tries to recover almost 100% links. $DPTF/IDF$ supplements the results of TF/IDF with additional correct links. We did not observe any case where $DPTF/IDF$ missed a link that TF/IDF recovered. Figure 7 shows that $DPTF/IDF$ also recovers more links than DOI/IDF.

7.4 Static Weights vs. DynWing

Manually tuning the parameters (λ_k) requires a labelled corpus, *i.e.*, an oracle, that is almost never available [6]. Consequently, previous techniques [5,7,38] used a range of parameters (λ_k) values to tune their techniques. However, it is possible that these manually-tuned parameters may work on one dataset but not on the other. In our previous study [7], we found that iTrust had less good results than Pooka. One reason could be that the parameter values were better for Pooka.

We use DOI/IDF, with static parameter tuning, as previously used in [7], and use $DPTF/IDF$ combined with DynWing. Then, we compute the precision, recall, and F-measure of the obtained links in comparison to our three oracles: ($Oracle_{iTrust}$, $Oracle_{Lucene}$, and $Oracle_{Pooka}$).

We observe in Table 9 that $DPTF/IDF$ provides slightly better results than DOI/IDF. For example, in the case of iTrust, DynWing provides up to on average 1.88% better F-measure than DOI/IDF. In the case of Lucene, Table 8 shows that $DPTF/IDF$ provides better precision and recall than baseline, *i.e.*, TF/IDF, links, whereas DOI/IDF provides better precision but lower recall than baseline. $DPTF/IDF$ tends to provide a better trade-off between precision and recall. We conclude that $DPTF/IDF$ combined with DynWing does not require any previous knowledge of an oracle and developers intervention for tuning parameters for every dataset. The achieved improvements are statistically significant and with large effect sizes. In the case of Lucene, Table 8 shows that DOI/IDF has no effect and that using $DPTF/IDF$ provides a large effect size on improvement.

8 Threats to Validity

Several threats limit the validity of our case studies. 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, threats to the construct validity could be due to measurement errors. We use the time spent on each AOI and the percentages of correct answer to measure the subjects' efficiency during the eye-tracking case studies. These measures are objective, even small variations due to external factors, such as fatigue, could impact their values. We minimise this threat by using small source code and all the subjects finished each case study within, on average, 30 minutes. We also use the widely-adopted F-measure to assess our new weighting scheme, $DPTF/IDF$, as well as its improvement.

The oracle used to evaluate the accuracy of the RT links could also impact our results. To mitigate this threat, two authors manually created traceability links for Pooka and the third author verified the links. In addition, the oracle for Pooka was not specifically built for this empirical study. We used this oracle in our previous studies [4, 5]. We used iTrust traceability oracle developed independently by developers who did not know the goal of our empirical study. For Lucene, we parsed the content of its JIRA version-control repository to create an oracle. It is possible that developers mentioned a wrong feature ID in some log message but we manually verified all the RT links in the oracle to discard any false positive links to minimise this threat.

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 any specific case study because we provide two independent sets of six different pieces of source code to the subjects. Moreover, the subjects did not know the goal of the case study explicitly.

A threat could occur due to the subjects' selection and impact our study results due to natural differences among the subjects' abilities. In our two eye-tracking case studies, 24 and 14 subjects performed tasks related to RT links. The total number of subjects, 38, mitigates the selection threat. The 14 subjects of the second eye-tracking include 10 subjects from the first eye-tracking case study. Therefore, their could be a learning effect: these subjects could have learned how to perform RT link verification tasks. We mitigate this threat by using two different sets of stimuli and by having a six months gap between the two case studies.

Subjects' Java experience and background in object-oriented programming could lead subjects to focus on some types of SCEs just because they are available, not for their semantic content. We minimise this threat in two ways. First, we ask all subjects for their preferred types of SCEs during source code comprehension at the end of the first case study. Second, we perform a second case study highlighting certain types of SCEs, described in Section 4. The post-experiment questionnaire and the second case study results confirm the findings of our first case study.

The selection of the λ_k parameters could also impact the results that we report in this paper. We minimise this threat by using two different parameters tuning approaches, *i.e.*, static and dynamic. In both cases, results were statistically improved over the baseline results. However, using different values to tune λ_k could provide better or slightly different results.

LDA cannot yet identify domain-level terms perfectly. Thus, our use of LDA to distinguish domain-level terms from implementation-level terms could impact the results of RQ2 and RQ3. However, better and more precise approaches would positively impact our results, thus our use of LDA can be considered as conservative. We would need more case study to analyse the impact of different domain-level terms extraction approaches on $DPTF/IDF$.

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 software professionals is one of the threats to generalize our results. Our subjects were all graduate students with an average of 3.39 years of Java experience, *i.e.*, they all had good knowledge of Java. In addition, some of the subjects had industrial experience. Thus, we believe that, as mentioned by Kitchenham *et al.* [30], "using students as subjects is not a major issue as long as [we] are interested in evaluating the use of a technique by novice or non-expert software engineers. Students are the next generation of software professionals and, so, are relatively close to the population of interest."

The other external threat could be the size of the source code in our case study. It is possible that in real development environment, *e.g.*, Eclipse, subjects may have different SCEs preferences. However, using development environment with eye-tracking system would weaken our control over the case study. To mitigate this threat, we used a post-experiment questionnaire and perform the second case study to confirm our subjects' preferred types of SCEs.

Our empirical study was limited to three systems: iTrust, Lucene, and Pooka. They are not comparable to industrial systems, but the datasets used by other authors [9,34,38] to compare different IR techniques had comparable sizes. We cannot claim that we would achieve similar results with other systems. Different systems with different identifiers' quality, requirements, implementations, or different artifacts may lead to different results [3]. However, the three selected systems have different source code quality and requirements. Thus, 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 verify the data distribution before selecting an appropriate parametric or non-parametric test and effect size.

9 Conclusion and Future Work

In this paper, we conjectured 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.* To support this conjecture, we ask four research questions (RQs) pertaining to (1) the type of source code entities (SCEs) used by developers to verify RT links (RQ0 and RQ1) and (2) the use of these to propose a new weighting scheme, $DPTF/IDF$, to use with LSI to recover RT links (RQ2 and RQ3). Tables 1 and 2 summarise the RQs, the case studies performed to answer them, and the answers.

We answer the RQs as follows. For RQ0, we analyse the eye movements of 24 subjects to identify their preferred SCEs when they read the source code to verify RT links using an eye-tracking system. 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 report that, as soon as developers read a requirement, they paid immediately attention to method names and then comments. We also observe that subjects give more importance to domain-level SCEs.

For RQ1, we highlight the subjects' preferred types of SCEs to assess whether their preferences are due to the location of the SCEs or due to the semantic contents. Such a preference would be similar to the observation that subjects trust a piece of information by virtue if it being shown rather than by its intrinsic value in the results returned by the Google search engine [36]. We show that it is indeed the semantic content of the SCEs that developers seek.

For RQ2 and RQ3, we propose a new weighting scheme, $DPTF/IDF$, that takes into account the subjects' preferred types of SCEs and compared it with DOI/IDF. We use LDA to distinguish domain-level terms from implementation-level ones. We use a LSI technique with $DPTF/IDF$ on three datasets, namely iTrust, Lucene, and Pooka to show that, in general, the proposed scheme has a better accuracy than TF/IDF in terms of F-measure. In all three systems, the new weighting scheme statistically improves the accuracy of the IR-based technique.

Thus, we conclude that taking into account source code elements is important to improve the accuracy of RT techniques. Consequently, there are several ways in which we are planning to continue this work. We will use more advanced techniques to distinguish domain-level and implementation-level terms [2] and advanced LDA parameter tuning techniques [37]. We will apply $DPTF/IDF$ on different datasets, if possible from the industry. We will also analyse in which type of SCEs a requirement term plays a more important role. We will apply the proposed weighting scheme on heterogeneous artifacts to analyse the improvement of the accuracy. We also plan to use *DPTF/IDF* to improve feature location techniques. We may also perform more case studies on the role of highlighting SCEs in program comprehension. We plan to replicate this study with industrial developers in real development environments.

Acknowledgements The authors would like to thank all the participants of the case studies 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, and by Fonds de recherche du Québec – Nature et technologies (FRQNT).

References

1. A. Abadi, M. Nisenson, and Y. Simionovici. A traceability technique for specifications. In *Proceeing of 16th IEEE International Conference on Program Comprehension*, pages 103 –112, June 2008.
2. S.L. Abebe and P. Tonella. Towards the extraction of domain concepts from the identifiers. In *Proceeding of 18th Working Conference on Reverse Engineering (WCRE)*, pages 77 –86, 2011.
3. Nasir Ali, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Factors impacting the inputs of traceability recovery approaches. In Andrea Zisman, Jane Cleland-Huang, and Olly Gotel, editors, *Software and Systems Traceability*, chapter 7. Springer-Verlag, New York, 2011.
4. Nasir Ali, Yann-Gaël Gueheneuc, and Giuliano Antoniol. Requirements traceability for object oriented systems by partitioning source code. In *Proceedings of 18th Working Conference on Reverse Engineering*, WCRE '11, pages 45–54, Washington, DC, USA, 2011. IEEE Computer Society.
5. Nasir Ali, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Trust-based requirements traceability. In *Proceeding of 19th IEEE International Conference on Program Comprehension*, page 10, Washington, DC, USA, 2011. IEEE Computer Society.
6. Nasir Ali, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *IEEE Transactions on Software Engineering*, 99(PrePrints):1, 2012.
7. Nasir Ali, Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An empirical study on requirements traceability using eye-tracking. In *Proceedings of IEEE International Conference on Software Maintenance*, pages 191–200, 2012.
8. G. Antoniol, B. Caprile, A. Potrich, and P. Tonella. Design-code traceability for object-oriented systems. *Annals of Software Engineering*, 9(1):35–58, 2000.
9. Guiliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.
10. Adrian Bachmann, Christian Bird, Foyzur Rahman, Premkumar Devanbu, and Abraham Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the 18th ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 97–106, New York, NY, USA, 2010. ACM.
11. Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead, and Sushil K. Bajracharya. A theory of aspects as latent topics. *Sigplan Notices*, 43(10):543–562, October 2008.
12. Roman Bednarik and Markku Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, ETRA '06, pages 125–132, New York, NY, USA, 2006. ACM.
13. Roman Bednarik and Markku Tukiainen. An eye-tracking methodology for characterizing program comprehension processes. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, pages 125–132, New York, NY, USA, 2006. ACM.
14. M. Bunge. *Treatise on Basic Philosophy: Vol. 3: Ontology I: The Furniture of the World.* Reidel, Boston MA, 1977.

15. Teresa Busjahn, Carsten Schulte, and Andreas Busjahn. Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, Koli Calling '11, pages 1–9, New York, NY, USA, 2011. ACM.
16. Gerardo Cepeda Porras and Yann-Gaël Guéhéneuc. An empirical study on the efficiency of different design pattern representations in uml class diagrams. *Empirical Softw. Engg.*, 15:493–522, October 2010.
17. Barthélémy Dagenais, Harold Ossher, Rachel K. E. Bellamy, Martin P. Robillard, and Jacqueline P. de Vries. Moving into a new software project landscape. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 275–284, New York, NY, USA, 2010. ACM.
18. A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. Improving ir-based traceability recovery using smoothing filters. In *Proceeding of 19th IEEE International Conference on Program Comprehension*, pages 21 –30, june 2011.
19. Andrea De Lucia, Massimiliano Di Penta, and Rocco Oliveto. Improving source code lexicon via traceability and information retrieval. *IEEE Transactions on Software Engineering*, 37:205–227, 2011.
20. Andrea De Lucia, Fausto Fasano, Rocco Oliveto, and Genoveffa Tortora. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, 16(4), 2007.
21. Benoît De Smet, Lorent Lempereur, Zohreh Sharafi, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Naji Habra. Taupe: Visualizing and analyzing eye-tracking data. *Science of Computer Programming*, 2012.
22. Bogdan Dit, Annibale Panichella, Evan Moritz, Rocco Oliveto, Massimilano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. Configuring topic models for software engineering tasks in tracelab. *Proceedings of 7th ACM/IEEE International Conference in Software Engineering*, 13:105–109.
23. A.T. Duchowski. A breadth-first survey of eye-tracking applications. *Behavior Research Methods*, 34(4):455–470, 2002.
24. A.T. Duchowski. *Eye tracking methodology: Theory and practice*. Springer-Verlag New York Inc, 2007.
25. Berna Erol, Kathrin Berkner, and Siddharth Joshi. Multimedia thumbnails for documents. In *Proceedings of the 14th annual ACM international conference on Multimedia*, MULTIMEDIA '06, pages 231–240, New York, NY, USA, 2006. ACM.
26. Malcom Gethers, Trevor Savage, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Codetopics: which topic am i coding now? In *Proceedings of the 33rd International Conference on Software Engineering*, pages 1034–1036. ACM, 2011.
27. O. C. Z. Gotel and C. W. Finkelstein. An analysis of the requirements traceability problem. *1st International Conference on Requirements Engineering*, pages 94–101, April 1994.
28. T.L. Griffiths and M. Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences of the United States of America*, 101(Suppl 1):5228–5235, 2004.
29. Y.G. Guéhéneuc. Taupe: towards understanding program comprehension. In *Proceedings of conference of the Center for Advanced Studies on Collaborative research*, pages 1–13. ACM, 2006.
30. Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *Transactions on Software Engineering*, 28(8):721–734, August 2002.
31. G. Kowalski. *Information retrieval architecture and algorithms*. Springer-Verlag New York Inc, 2010.
32. Andrea De Lucia, Andrian Marcus, Rocco Oliveto, and Denys Poshyvanyk. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability*, pages 71–98. 2012.
33. Guillermo Macbeth, Eugenia Razumiejczyk, and Rubén Daniel Ledesma. Cliff's delta calculator: A non-parametric effect size program for two groups of observations. *Universitas Psychologica*, 10(2):545–555, 2011.
34. 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*, pages 125–135, Portland Oregon USA, 2003. IEEE CS Press.

35. Girish Maskeri, Santonu Sarkar, and Kenneth Heafield. Mining business topics in source code using latent dirichlet allocation. In *Proceedings of the 1st India software engineering conference*, ISEC '08, pages 113–120, New York, NY, USA, 2008. ACM.
36. Bing Pan, Helene Hembrooke, Thorsten Joachims, Lori Lorigo, Geri Gay, and Laura Granka. In Google we trust: Users' decisions on rank, position, and relevance. *Journal of Computer-Mediated Communication*, 12(3):801–823, April 2007.
37. Annibale Panichella, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 522–531. IEEE Press, 2013.
38. Denys Poshyvanyk, Yann-Gaël Guéhéneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Transactions on Software Engineering*, 33(6):420–432, 2007.
39. K. Rayner. Eye movements in reading and information processing: 20 years of research. *Psychological bulletin*, 124(3):372, 1998.
40. Seeing Machine. Seeing Machine's website - FaceLAB. `http://www.seeingmachines.com/product/facelab/`, 2012. Accessed July 13 in 2012.
41. B. Sharif and J.I. Maletic. An eye tracking study on camelcase and under_score identifier styles. In *Proceedings of 18th International Conference on Program Comprehension (ICPC)*, pages 196–205. IEEE, 2010.
42. Bonita Sharif, Michael Falcone, and Jonathan 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*, ETRA '12, pages 381–384, New York, NY, USA, 2012. ACM.
43. Bonita Sharif and Huzefa 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)*, pages 67–70, NY, USA, 2011.
44. Y.H. Sun, P.L. He, and Z.G. Chen. An improved term weighting scheme for vector space model. In *Proceedings of 2004 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1692–1695. IEEE, 2004.
45. Hidetake Uwano, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the 2006 symposium on Eye tracking research & applications (ETRA)*, pages 133–140, New York, NY, USA, 2006. ACM.
46. Jinshui Wang, Xin Peng, Zhenchang Xing, and Wenyun 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)*, pages 213 –222, 2011.
47. S. Yusuf, H. Kagdi, and J.I. Maletic. Assessing the comprehension of uml class diagrams via eye tracking. In *Proceedings of 15th IEEE International Conference on Program Comprehension (ICPC)*, pages 113–122. IEEE, 2007.