UNIVERSIDAD
NACIONAL
DE COLOMBIA

Analyzing the Impact of Information Retrieval and Relevance Feedback Techniques on Concept Location Tasks

Andrés Fernando Wilches Riaño

Universidad Nacional de Colombia
Engineering Faculty, Department of Systems and Industrial Engineering
Bogotá, D.C., Colombia
2017

Analyzing the Impact of Information Retrieval and Relevance Feedback Techniques on Concept Location Tasks

Andrés Fernando Wilches Riaño

Thesis presented as a partial requirement for the degree of:
Master in Systems Engineering and Computer Science

Advised by:
Ph.D. Jairo Hernán Aponte Melo

Research Lines:
Software Maintenance and Evolution
Research Group:
Colectivo de Investigación en Ingeniería de Software (ColSWE)

Universidad Nacional de Colombia
Engineering Faculty, Department of Systems and Industrial Engineering
Bogotá D.C., Colombia
2017

To my mom and my family.

Acknowledgements

Firstly, I want to express my gratitude to the teacher Jairo Aponte, for advising and guiding me from the start to the end of this thesis.

Secondly, to the students of the software engineering course who participated in the development of the controlled experiment, specially to Andrés Rene Gutierrez and Camilo Dajer.

Thirstly, to Mario Arrieta for supporting me on the statistical analysis of the controlled experiment results.

Finally, to my mom for her patience and unconditional help.

Abstract

Concept location is the process by which a programmer determines the place, within a system codebase, where a change is to start in response to a modification request. It is a usual and fundamental process performed as part of software maintenance tasks such as bug fixing, refactoring, and in some cases, new feature implementation. One of the recent approaches proposed to support that process augments information retrieval (IR) based concept location via an explicit relevance feedback (RF) mechanism.

In this thesis, we present an Eclipse plugin that implements the IR+RF approach and a controlled experiment aimed at assessing the impact of that approach on bug fixing tasks. Within the experiment, five bug fixing tasks were performed by 40 undergraduate software engineering students. The efficiency of the participants, the completion time, and the correctness of their responses were measured. The results indicate that the IR+RF approach surpasses in effectiveness and efficiency the default searching functionalities provided by the Eclipse IDE. On the other hand, it does not reduce the completion time in bug fixing tasks.
Keywords: Concept Location, Information Retrieval, Relevance Feedback, Controlled Experiment.

Resumen

La localización de conceptos es el proceso por el cual un programador determina el lugar dentro de un sistema, donde un cambio inicia en respuesta una solicitud de modificación. este es un proceso usual y fundamental que se realiza como parte de las tareas de mantenimiento de software como reparación de errores, refactorización de código y en algunos casos, implementación de nuevas características. Uno de los enfoques recientes propuestos para apoyar este proceso mejora la localización de conceptos basada en recuperación de la información (IR) a través de un mecanismo explícito de retroalimentación relevante (RF).

En esta tesis nosotros presentamos un plugin de Eclipse que implementa el enfoque IR+RF y un experimento controlado destinado a evaluar el impacto de este enfoque en tareas de reparación de errores. Dentro del experimento, cinco tareas de reparación de errores fueron realizadas por 40 estudiantes del curso de ingeniería de software. Se midió la eficiencia de los participantes, el tiempo de terminación, y la correctitud de sus repuestas. Los resultados indican que el enfoque IR+RF sobrepaso en correctitud y eficiencia las funcionalidades de búsqueda proporcionadas por defecto por el IDE Eclipse- Por otro lado, no se reduce el tiempo de terminación en tareas de reparación de errores.
Palabras clave: Localización de Conceptos, Recuperación de la Información, Retroalimentación Relevante, Experimento Controlado.

List of Tables

List of Figures

Contents

1. Introduction

Finding the part of a software system in which a problem domain concept is implemented is an essential task in software maintenance. For instance, this task appears when a developer needs to determine the source code location where a change is to start in response to a modification request. Nowadays Concept Location is defined as the process that identifies where a software system implements a specific concept. For developers the process of changing a feature of a determined software application could be a tricky issue since they have to look for domain concepts inside the source code in order to find where the feature is implemented. This process begins when a change request or bug report are requested and ends when the developer found the concept or feature that developer needs either to change or fix.

Information retrieval methods have been used to address the concept location problem[4–9, 15, 17, 18]. Information retrieval (IR) is a discipline that deals with the retrieval of unstructured data, especially text documents. [2] defines IR as *"finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."* The general goal of IR systems is to minimize the required time to trace the information that a user needs [3]. In the context of concept location, the IR method is used to map domain concepts expressed as developer queries to source code artifacts. Once the resulting list of artifacts is calculated by the IR method, the developer inspects and evaluates those results. This querying-inspecting cycle normally requires being repeated several times until the developer finds the desired software artifact.

On the other hand, Relevance Feedback (RF) is an automatic approach that aims to improve query formulation within information retrieval processes. RF is a two-part process: first, the developer judges and assesses the relevance of existing results returned by a search. Then, the information retrieval system uses this feedback information to perform a new search, and (hopefully) return more relevant results to the developer[4]. The classic algorithm for implementing RF is the Rocchio algorithm [13]. Rocchio was developed using the Vector Space Model (VSM) and it is based on the assumption that the user knows which documents should be denoted as relevant or non-relevant [2].

The Vector Space Model (VSM) algorithm stores a mathematical representation of documents as vectors; these vectors capture the importance of a term in a document as a weight [2]. The weights of terms are calculated based on *term frequency - inverse document frequency (tf-idf)*, which is an indexing statistic that indicates the importance of a term in the document. The similarity of two documents is computed with the cosine of the vectors that represent both documents[15].

Gay et al. [4] proposed an approach for concept location that combines an IR method and an RF algorithm. The novelty of that approach relies on the use of RF to enhance an IR method in two ways: (i) improving the developer's ability to write better queries throughout the entire concept location process, and (ii) allowing the developer to use the knowledge gained when inspecting a particular resulting list by incorporating it in the next query. With the aim of empirically evaluating that approach, we carried out a controlled experiment [14]. Thus, we collected evidence to determine whether the usage of an embedded information retrieval and relevance feedback tool increases the developers' performance when they have to carry out concept location tasks.

Currently, developers often perform concept location manually. This circumstance implies that the time spent performing this task and the effort could be so much. Consequently, we think an embedded tool (Eclipse plug-in) is needed to support this process in large and complex programs, unfamiliar for the developers.

IR and RF techniques applied to concept location will allow making this process easier since they can reduce the time required and the effort to locate certain features inside the source code. Consequently, these techniques may rise the developers' productivity during software maintenance and evolution stages. By the development of an embedded tool (Eclipse plug-in) that support concept location, we want to provide developers an aid for doing such process. The main purpose of the research is to observe a reduction in the time and effort required by a software development team to successfully accomplish concept location tasks, especially when they are dealing with large and complex unfamiliar software applications.

In this thesis, we evaluate and analyze the impact of Information Retrieval and Relevance Feedback techniques on the development of maintenance tasks as fault localization. So we presented an Eclipse plug-in that implements the IR+RF approach for concept location and a controlled experiment designed to compare the performance of this approach with the standard Eclipse searching capabilities.

2. Background and Related Work

2.1. Concept Location

Biggerstaff et al.[1] defined concept assignment problem as *"discovering human oriented concepts and assigning them to their implementation instances within a program"*, it is currently known as concern, feature or concept location.Software evolution and maintenance involves tasks as adding new features, improving existing functionalities and removing or fixing bugs [16]. Concept location is a crucial activity because it is one of the most important and frequent activity during the incremental change process.

During concept location, the developers have to face several challenges as finding the snippet of code related to a feature, understanding and modifying unfamiliar code. In many of cases, this search is performed manually and even with the help of a tool. It implies a big amount of time consuming and effort by developers. After concept location a impact analysis task has to be executed in order to find the code affected by a change[16].

Concept Location starts in response of a change request or bug report and ends when the developer finds the location of a feature or functionality where either a change has to be made in order to fix a bug or a new feature has to be implemented so as to respond a change request.

Several approaches to deal with concept location have been developed. These can be broadly classified into static, dynamic and combined analysis based approaches. Dit et al. [16] carried out a survey to discuss in detail the existing approaches.

Dynamic analysis refers to examine the execution of a software system. It is focused in the observation of the features invocation during runtime. The most common approach used in dynamic concept location is the observation of execution trace and scenarios [16].

On the other hand, static techniques for concept location are focused on structural information such as either data or control dependencies[16].

Another approach used to deal with concept location is textual through the analysis of words or terms present in the source code, the main objective of this approach is map pieces of source code to features and domain using identifiers and comments. The main techniques used in textual concept location are pattern matching, information retrieval (IR) and natural language processing (NLP) [16].

Concept location is not limited to dynamic, static or textual analysis, some techniques use a combination of these approaches to cope concept location tasks[16], others do not use any of these analyses, such as historical revision of control version systems to identify lines of code and examination of viewed code by developers during maintenance tasks in order to derive what was important[16].

2.2. Information Retrieval for Concept Location

The main goal of Information Retrieval techniques used for concept location tasks is to map domain concepts expressed as queries performed by developers to software artifacts or source code snippets that satisfy these queries.

IR techniques overcome the limitations of string-matching techniques by showing a ranked-list of results since techniques such as grep or regular expressions show the results of a search to the user in no a particular order. While when the users use an IR techniques they can investigate in a top-ranked list according to the similarity of the terms of a query and the documents where they are looking for[17].

Another advantage of IR techniques is that the developers do not need to review the execution trace, know the statical or structural information of the software application or make debug process to perform concept location. Developers can look for source code through text extracted from a bug description or a change request and keywords they considered as important or relevant.

Marcus and Haiduc introduced several important concepts of text retrieval in [17]. It worths to delve some of them:

Document is the basic unit retrieved as a result of a search; in the concept location scope a document is a source code artifact such as methods, classes, lines of code, etc.

Term is the unit of discourse used to express meaning in a document; in concept location, the terms are extracted from identifiers and comments.

Corpus is a collection of documents e.g. projects, classes, methods.

Indexing is the process of transforming the corpus into an intermediary mathematical representation that depends on a text retrieval technique. The outcome of the indexing process is called *index*.

Query is an expression as a set of one or more words in a specific format that represents the information needed by users.

Relevance expresses the degree to which a result retrieved by a query satisfies the information needed by users.

Marcus and Haiduc [17] presented and discussed the application of several Text Retrieval techniques to support concept location.

Marcus et al. [5] applied an information retrieval method called Latent Semantic Indexing (LSI) to map concepts, expressed by developers in natural language, into relevant code fragments. Marcus et al.[6] used static concept location techniques in Object-Oriented code such as regular expression matching, static program dependences, or information retrieval. They compared these techniques to determine their strengths and weaknesses.

Poshyvanyk and Marcus[7]augmented an existing technique based on information retrieval for concept location with the automatic organization of the searching results using a combination of Formal Concept Analysis (FCA) and LSI.

Later, Scanniello and Marcus [8] presented a new approach to static based concept location that combines structural and lexical information. They used an IR-based approach which formulates concept location as a text retrieval problem.

Poshyvanyk et al. [9] enhanced an existing IR-based technique for concept location with an automatic clustering of the search results through Formal Concept Analysis. This approach selected the most relevant attributes of the best-ranked documents from a ranked list and clustered them.

The main difference between these approaches and our is that we used the VSM algorithm to index source code at methods granularity level. We developed an Eclipse plug-in as an embedded tool that allows to perform queries, analyze the results in a ranked list and examine the results (methods) directly in the Eclipse IDE and where they are implemented within the source code of the application.

2.3. Relevance Feedback and Query Expansion for Concept Location

The main goal of Relevance Feedback is to improve the results of a search based on the information given by users of documents considered as relevant and no relevant. In concept location, RF is used within IR process. The developer judges the relevance of the results retrieved by a search then this information is used to perform a new search that will return more relevant results to developers.

Gay et al. [4] proposed an approach to augment IR-based concept location with an explicit relevance feedback mechanism. They performed a case of study with one developer to determine the impact of IR and Relevance Feedback (RF) on finding buggy methods, based on bug reports, on open source projects. The main objective of this case of study focused on analyzing the effect of RF in tasks such as fault localization. This case of study revealed that this approach reduced developer effort since manual query reformulation was absent. On the other hand, we performed a controlled experiment between two groups, one used the IR+RF approach (baseline) and the other one used the default eclipse features for searching.

A crucial activity for searching effectiveness during the concept location process is the formulation and improving of queries. Haiduc et al. [10–12] presented several approaches to assess and reformulate queries and improve their quality. These are useful in software engineering tasks, such as concept location, impact analysis, or when developers are looking for source code artifacts. Text Retrieval (TR) techniques were used to evaluate the quality of queries and to predict and reformulate them in order to improve the results, On the other hand, we implemented a relevance feedback mechanism in order to reformulate and improve the query and the results returned by the IR mechanism.

Mills et al. [24] implemented and evaluated two applications for automatic query quality prediction in the context of concept location and traceability link recovery among solutions. They found that their approach was able to predict the results through the evaluation of the the queries' text it represented a reduction in time and effort.

Chaparro et al. [25] used the Observed Behavior (OB) descriptions to reduce low-quality queries in TR-Based bug localization. Besides, they conducted an empirical study by comparing four TR-based bug localization approaches versus OB only. They conclude that reformulated queries improved TR-based bug localization for all approaches.

Sisman et al.[26] built a framework to automatic query reformulation by enriching the initial query with specific additional terms from the highest-ranked artifacts retrieved by a user query. They demonstrated the superiority of their framework for bug localization tasks compared with well-known QR frameworks in two large software projects and using more than 4000 queries.

2.4. Empirical Studies for Concept Location

Empirical software engineering aims at identifying and understanding the relationships among factors and variables, one of the research paradigms in empirical software engineering are the empirical strategies, it refers to study the objects in their natural state and finding the results through observation. Among empirical strategies of research are surveys, case of studies, controlled experiments and quasi-experiments[14].

Controlled experiments[14] are launched when the control over a situation is desired or researchers want to manipulate a behavior directly. It involves the comparison of outcomes of one or more treatments. Controlled experiments are performed in a controlled environment under controlled circumstances, the events are organized to simulate actual situations of real world. Controlled experiments can be human-oriented or technology oriented. Researchers have less control of human-oriented experiments, since the human behavior changes according the situations and it is hard to control. According to Wohlin et al.[14] the controlled experiments help to investigate aspects like confirm theories, explore relationships evaluate the accuracy of models, or validate measures. In controlled experiments are applied to subjects and objects. The subjects are the people that apply the treatments, and the characteristics of this subjects can be considered as independent variables. The experiments consist of a set of tests or tasks in combination of treatments, objects and subjects.

Poshyvanyk and Marcus[7] evaluated this approach through a case of study and concluded that it outperformed traditional ranking technique. Later, Scanniello and Marcus [8] appraised their approach and concluded that it surpassed a text search baseline. Poshyvanyk et al. [9] assessed their approach through a large case of study, using features and bugs from six open source systems. They concluded that their proposed approach outperformed a standalone IR-based concept location techniques since it reduced irrelevant search results.

Gay et al. [4]performed several tasks with N (1, 3 and 5) marked methods in each round of feedback, they measure the number of methods marked to reach the target method and the number of feedback rounds. The baseline is the position (according to the rank) of the target method in initial query and they measured whether the RF mechanism improved the position of target method in a ranked-list after marking N methods in each feedback round and several feedback rounds. On the other hand, we measured for each task the effectiveness, time spent, marked and reviewed methods, formulated queries and number feedback rounds (for the Tool group). So Gay et al. metrics can be compared with our work in terms of efficiency and correctness because they reported the marked methods and feedback rounds and when the feedback mechanism helped to reach the target method or failed.

Wang et al. [15] conducted a user study in order to determine how developers behave when using an IR-based tool to perform fault localization tasks, based on bug reports. They concluded that there was not a meaningful difference in benefit between high-quality bug reports and IR-based techniques i.e., IR techniques are useful to find the faulty file but these do not reduce the time for understanding and fixing that file.

Scanniello and Marcus [18] implemented a concept location solution using the PageRank algorithm. They conducted a controlled experiment and replications to assess if their new technique helps the user to find the places where a change has to be made in the code. The evaluation concluded that their approach yielded better retrieval performance that previous text retrieval and clustering baselines.

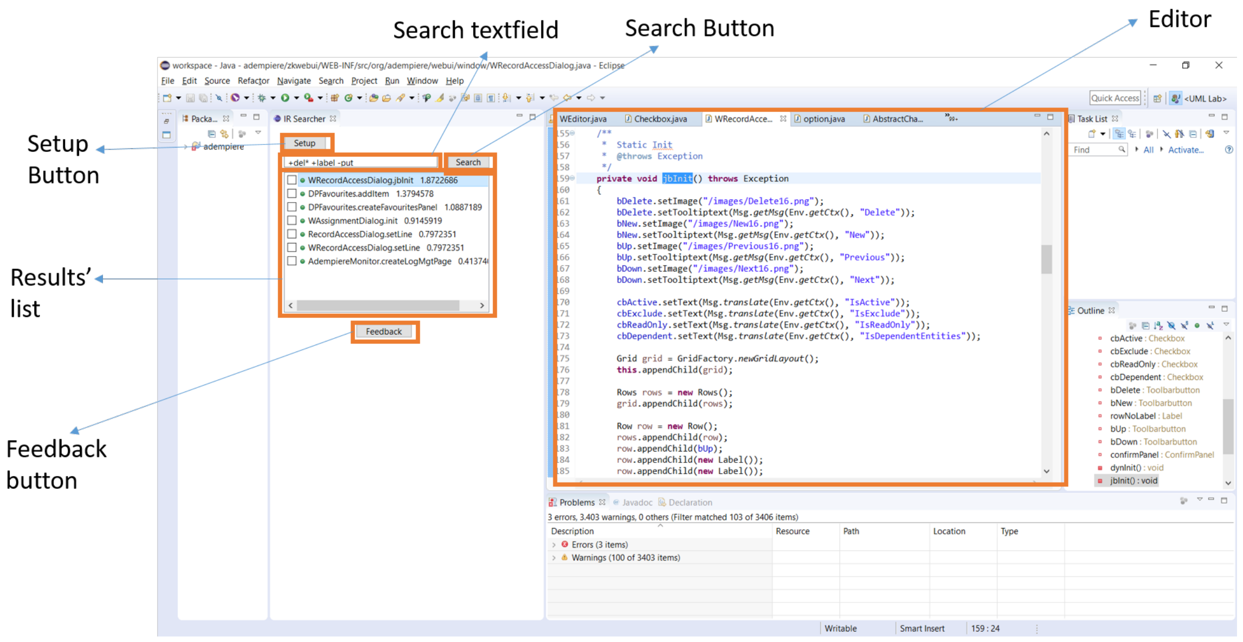
3. Eclipse Plug-in and the IR+RF approach for Concept Location

In this chapter, we present how the Eclipse plug-in is internally built and how we used the IR+RF based approach to create the tool.

3.1. Eclipse Plug-in

We built an Eclipse plug-in based on guidelines staged in [23]. It implements the IR+RF approach for concept location. The plug-in allows the users to make queries and explore source code within the Java default perspective.

Figure 3-1.: Eclipse plug-in GUI



```
+del* +label -put
```

```
WRecordAccessDialog.jbInit   1.8722686
DPFavourites.addItem   1.3794578
DPFavourites.createFavouritesPanel   1.0887189
WAssignmentDialog.init   0.9145919
RecordAccessDialog.setLine   0.7972351
WRecordAccessDialog.setLine   0.7972351
AdempiereMonitor.createLogMgtPage   0.41374
```

```java
    /**
     *  Static Init
     *  @throws Exception
     */
    private void jbInit() throws Exception
    {
        bDelete.setImage("/images/Delete16.png");
        bDelete.setTooltiptext(Msg.getMsg(Env.getCtx(), "Delete"));
        bNew.setImage("/images/New16.png");
        bNew.setTooltiptext(Msg.getMsg(Env.getCtx(), "New"));
        bUp.setImage("/images/Previous16.png");
        bUp.setTooltiptext(Msg.getMsg(Env.getCtx(), "Previous"));
        bDown.setImage("/images/Next16.png");
        bDown.setTooltiptext(Msg.getMsg(Env.getCtx(), "Next"));

        cbActive.setText(Msg.translate(Env.getCtx(), "IsActive"));
        cbExclude.setText(Msg.translate(Env.getCtx(), "IsExclude"));
        cbReadOnly.setText(Msg.translate(Env.getCtx(), "IsReadOnly"));
        cbDependent.setText(Msg.translate(Env.getCtx(), "IsDependentEntities"));

        Grid grid = GridFactory.newGridLayout();
        this.appendChild(grid);

        Rows rows = new Rows();
        grid.appendChild(rows);

        Row row = new Row();
        rows.appendChild(row);
        row.appendChild(bUp);
        row.appendChild(new Label());
        row.appendChild(new Label());
```

```
cbActive : Checkbox
cbExclude : Checkbox
cbReadOnly : Checkbox
cbDependent : Checkbox
bDelete : Toolbarbutton
bNew : Toolbarbutton
rowNoLabel : Label
bUp : Toolbarbutton
bDown : Toolbarbutton
confirmPanel : ConfirmPanel
dynInit() : void
jbInit() : void
```

```
3 errors, 3.403 warnings, 0 others (Filter matched 103 of 3406 items)
Description                          Resource   Path   Location   Type
> Errors (3 items)
> Warnings (100 of 3403 items)
```

We created a new view and called it “IR Searcher”. As shown in Figure **3-1**. The main components of this view are:

Search textfield: This component allows users to write text queries. The query format will be discussed ahead.

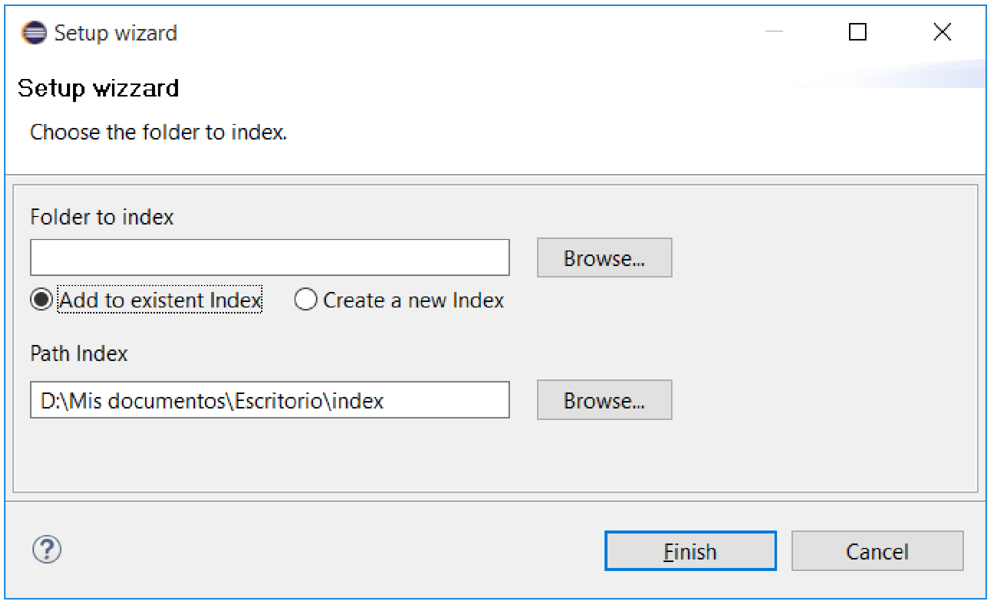
Search button: It performs a search based on the query written in the search text field. If the search retrieved results, the result list is filled; on the contrary, a message is shown to inform that the search did not find any result that satisfies the query. The plugin informs the users through a message in case that the query format is not valid and consequently, the search is not executed.

Results list: It contains the list of results (methods) of a search. Users can examine and explore the results of a search that satisfies a query. By clicking on one item of the resulting list, users can see where and how it is implemented, in the Java editor of the Eclipse IDE. Besides, users can select the method they considered as relevant so as to use the feedback mechanism to automatically reformulate the query.

Setup button: By clicking this component the setup wizard is opened.

Setup wizard: Through this component (Figure **3-2**) users are able to select the folder to index, i.e. the path of the project that will be the corpus, and a folder to store the index mathematical representation of Java code after applying IR method. Besides, it allows either to create a new index with the selected project or add it to an existing index. Users can change the path of the index by selecting the folder of another index path and leaving empty the folder to index field.

Figure 3-2.: Setup Wizard


Setup wizard

Setup wizzard

Choose the folder to index.

Folder to index

[✓ Add to existent Index] Create a new Index

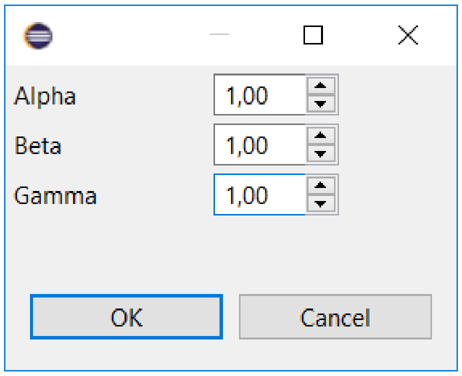
Path Index

D:\Mis documentos\Escritorio\index

Feedback button: This component automatically formulates a new query based on previous query and the relevant methods that were selected from the list of results. The feedback mechanism will not work if no document is selected.

Feedback parameters dialog: When the user clicks the feedback button a dialog is shown (Figure **3-3**). In this component, the user can change feedback parameters (we will present it further) alpha α, beta β and gamma γ. If the user clicks the OK button, the feedback is performed. On the other hand, if the cancel button is clicked the feedback is not executed.

Figure 3-3.: Feedback parameters dialog



Alpha 1,00
Beta 1,00
Gamma 1,00

As shown in Figure **3-1**, In the search textfield users are able to write queries; by clicking the search button a search is performed; the search results are shown in the resulting list of methods where users can examine and explore the results that satisfies a query. By clicking on one item (method) of the resulting list, users can see where and how it is implemented in the Java editor of the Eclipse IDE. The users can select the methods they considered as relevant and use the feedback mechanism to automatically reformulate the query, by clicking the feedback button the resulting list is refreshed with the results of the reformulated query.

3.2. Indexing Process

We used Apache Lucene[1]text search engine, written in Java, to index the corpus and made every task concerning text retrieval.

The indexing process needs a file stream, so we need to select the path of a Java project to build the index. First, the plug-in recursively visits all the files starting from project root of selected project path and check if the file is a folder and then if it is a Java file. When a Java file is found, we used the Visitor pattern to inspect the Abstract Syntax Tree (AST) representation of the Java file, to extract the methods. Secondly, an instance of the *Analyzer* class is responsible for extracting tokens (terms) out of the method's body and header, removing English and Java stop words

[1]`https://lucene.apache.org/core/`

(public, private, protected, interface, abstract, implements, extends, null, new, switch, case, default, synchronized, do, if, else, break, continue, this, assert, for, instance of, transient, final, static, void, catch, try, throws, throw, class, finally, return, const, native, super, while, import, package, true, false), and doing stemming (stemming is the process of reducing a transformed word to its corresponding word stem), We used Porter Stemmer algorithm included with Lucene. Furthermore, Lucene Javadoc defines the *term Vectors* as "a list of the document's terms and their number of occurrences in that document." We decided to store in the *document* instance the *term vectors* of the header and the body of methods, it will be useful for the feedback process. The other information of the method (file path, parameters, name, declaration, etc) are kept without changes i.e., English stop words and Java stop words removal and stemming was not applied to this information. For each visited method an instance of *Document* class of Lucene is created. In this document object is gathered the header and body processed by the analyzer, the number of parameters, the file path, the name, and the declaration of the method. Finally, an instance of the *IndexWriter* class stores the mathematical representation of the method represented in the document object, in the folder selected in the index path of the setup wizard. Figure **3-4** shows a flowchart of the indexing process.
When the choice "create a new index" is selected in the setup wizard, an index is created and stored in the selected folder. If the "add to existent index" option is chosen, the *IndexWriter* saves the new documents in the picked index and updates the existent documents that were in that index.

3.3. Search Process

The search process starts when the user creates a query and click on the search button, and finishes when the resulting list is filled with the documents that satisfies the formulated query. The internal search process implemented in the plug-in works as follows:

Firstly, the query in text format is parsed to Lucene format since the query must have a valid format to be parsed. Table **3-1** shows examples of valid query expressions [19, 20]; users can combine one or more query expressions so as to improve the query formulation. If the query format is not valid the process can not continue. The query received the same preprocessing as the methods in indexing process (English and Java stop words removal, stemming, etc) through an instance of the *Analyzer* class.

Secondly, an instance of the *IndexSearcher* class accesses the index directory and performs the search with the parsed query.

Finally, the documents (methods) that satisfy the query are retrieved and ranked according to their tf-idf score. Lucene uses a combination of the VSM of information retrieval and the boolean model [2] to determine the relevance of a document according to a user's query. Once the search retrieved the search results, the user is able to judge their relevance in accordance with their needs. The documents retrieved by a search are descendingly organized according to their score (rank) in the resulting list.

Figure 3-4.: Flowchart of the indexing process

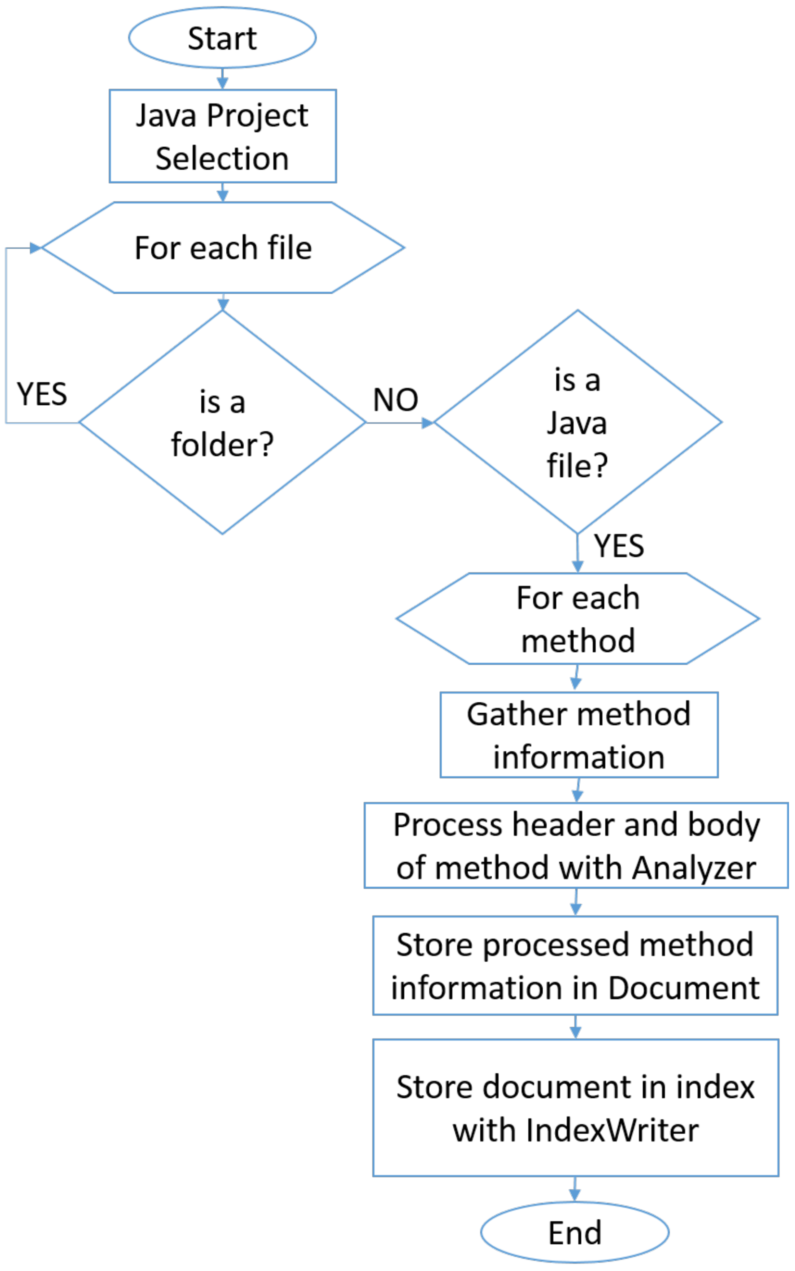

Start
Java Project Selection
For each file
YES
is a folder?
NO
is a Java file?
YES
For each method
Gather method information
Process header and body of method with Analyzer
Store processed method information in Document
Store document in index with IndexWriter
End

Table 3-1.: Queries expressions of Lucene [19, 20]

Query expression	**Matches documents that...**
java	Contain the term *java* in the default field.
java junit java OR junit java || junit	Contain the term *java*,*junit*, or both.
+java +junit java AND junit java && junit	Contain both *java* and *junit*.
java NOT junit java AND NOT junit java –junit java !junit	The NOT operator excludes documents that contain the term after NOT. **Note:** The NOT operator cannot be used with just one term.
(jakarta OR apache) AND website	Using parentheses to group clauses to form sub-queries.
java*	Contain terms that begin with *java*, like *javaspaces*, *javaserver*, and *java.net*.
java∼	Contain terms that are close to the word *java* such as *lava*.
"junit in action"	Contain the exact phrase "junit in action".
te?t test* te*t	To perform a single character wildcard search use the "?" symbol. To perform a multiple character wildcard search use the "*" symbol. *Note:* You can not use a * or ? symbol as the first character of a search.

3.4. Relevance Feedback Process

The relevance feedback process starts when the user selects the documents he considered as relevant and ends with a query automatically reformulated. The plug-in implemented the feedback process as follows. First, the methods marked as relevant and those considered as non-relevant as stored in a collection (vector of *QueryTermVector*) separately. Second, We extract the terms and the terms frequencies of the relevant and non-relevant methods and query terms, Third, We modified the tf-idf score assigned to each document according to the similarity of each document with the original query by multiplying that score with the weighted parameters of rocchio algorithm (alpha,gamma and beta). Finally, a new query is built by adding to the original query, as or terms, the terms of relevant documents. and removing the terms from non-relevant documents. The results' list is replenished with the documents retrieved by the new query.

We used a modified implementation of Rocchio algorithm introduced by Gay et al. [4]. The factors alpha, beta, and gamma are used as weighting parameters. These factors are setup as Figure **3-3** shows.

3.5. The IR+RF approach for Concept Location

The tool followed most of the IR based concept location principles established in [4]:

(I) ***Corpus creation:*** The code was parsed using methods as granularity level. The user should choose the root folder of the project that he wants to look for. We use an Abstract Syntax Tree (AST) to visit all the Java methods in the project. The documents indexed are the methods visited with the AST. We process the text of the header and body of the methods and apply text retrieval techniques like stop words removal, Java stop words (while, for, if, new, etc.) removal, identifier splitting (we kept original identifiers), stemming, etc. The user can select another project and either add to an existing index or create a new one.

(II) ***Corpus indexing:*** We use Apache Lucene for indexing the corpus [19]. It uses the Vector Space Model (VSM) algorithm to store a mathematical representation of the corpus. Each method has its representation in the index. In VSM, each document is represented as a weighted vector that spans over the words of corpus [4]. VSM was chosen because RF techniques were developed specifically for this IR algorithm.

(III) ***Query formulation:*** The developer uses a set of words within specific syntax rules defined by Lucene [20] to formulate queries. Lucene parses the query and returns the documents, i.e., Java methods that match the query. Queries receive the same text retrieval pre-processing treatment as the methods in corpus creation.

(IV) ***Ranking documents:*** Lucene sorts documents retrieved by a query according the similarity score between the query and each document. The rank is assigned according to how closely each result matches the query [19].

(V) ***Results examination:*** The developer can review a ranked list of Java methods that satisfy a query. The programmer can visit the source code by clicking on the method that wants to investigate in the resulting list.

(VI) ***Relevance feedback:*** The tool uses an explicit feedback mechanism. That means that the developer has to select the methods he considered relevant, and based on that information, the query is rebuilt. To do that, the tool considers the original terms and the terms of documents considered relevant and removes the terms of documents considered as not relevant. We use a modified version of original Rocchio Relevance feedback algorithm introduced in Equation 3-1 by Gay et al.[4]. A new query Q' is built with the terms of original query Q multiply by the alpha α factor; plus the beta β weight factor divided by the number of documents retrieved by the query Q marked as relevant, multiply by the sum of terms of the set of documents d judged as relevant R_q; minus gamma γ factor divided by the number of documents pondered as irrelevant from the original query, multiply by the sum of the terms of the documents deemed as irrelevant I_q.

$$Q' = \alpha Q + \frac{\beta}{|R_q|} \sum_{d \in R_q} d - \frac{\gamma}{|I_q|} \sum_{d \in I_q} d \tag{3-1}$$

4. Controlled Experiment Design

The goal of our experiment is to provide empirical evidence of the efficiency and effectiveness of the IR+RF approach when compared to standard searching features provided by the Eclipse IDE.

4.1. Research Questions

We formulated the following research questions:

RQ1: Does the IR+RF approach reduce the time needed by developers and improve the performance of them when they perform concept location tasks, compared to default Eclipse IDE searching features?

RQ2: Does the IR+RF approach improve the effectiveness of the solutions for concept location tasks, compared to default Eclipse IDE searching features?

The main factor being analyzed is the use of the IR+RF approach implemented as an Eclipse plugin. A *between-subjects* design was utilized, where the subjects were divided into two disjointed groups, one that used the Eclipse plug-in, and the other one used the standard Eclipse features. To assess the impact of the IR+RF approach we measure three dependent variables: time, efficiency and effectiveness.

4.2. Hypotheses

Based on the two research questions, we formulated the hypotheses of Table **4-1**:

4.3. Dependent and Independent Variables

The experiment has only one independent variable: the tool used to solve the concept location tasks. Thus, the tool variable has two treatments, i.e., the IR+RF approach (Tool) and a baseline which is the default features provided by the Eclipse IDE (No Tool). To measure the effect of using the IR+RF approach, implemented as an Eclipse plugin, to support concept location tasks against the default Eclipse IDE features, we consider three dependent variables: effort, effectiveness, and efficiency.

Effort (completion time) refers to the time required by a developer to complete a concept location task, e.g. localizing a fault. *effectiveness* is a binary variable that indicates if the user successfully completed the concept location task. *Efficiency* is the number of methods reviewed and marked, and the queries formulated by the developer when performing the concept location task.

Table 4-1.: Hypotheses formulation

Null hypothesis	**Alternative hypothesis**
$H0_0$: There is no significant difference in effectiveness between the IR+RF and Eclipse groups for concept location tasks	$H0$: There is significant difference in effectiveness between the IR+RF and Eclipse groups for concept location tasks.
$H1_0$: There is no significant difference in time to complete between the IR+RF and Eclipse groups for concept location tasks	$H1$: There is significant difference in time to complete between the IR+RF and Eclipse groups for concept location tasks.
$H2_0$: There is no significant difference in the number of revised methods between the IR+RF and Eclipse groups for concept location tasks	$H2$: There is significant difference in the number of revised methods between the IR+RF and Eclipse groups for concept location tasks
$H3_0$: There is no significant difference in the number of marked methods between the IR+RF and Eclipse groups for concept location tasks	$H3$: There is significant difference in the number of marked methods between the IR+RF and Eclipse groups for concept location tasks
$H4_0$: There is no significant difference in the number of formulated queries between the IR+RF and Eclipse groups for concept location tasks	$H4$: There is significant difference in the number of formulated queries between the IR+RF and Eclipse groups for concept location tasks

4.4. Control Variables

We use the experience level and Java knowledge as control variables since it allows us to divide the participants into two balanced groups and assign them to the treatments (Tool and No Tool). Thus, before performing the controlled experiment we examined the experience level of participants through a demographic survey and divided them according to it.

4.5. Subject System

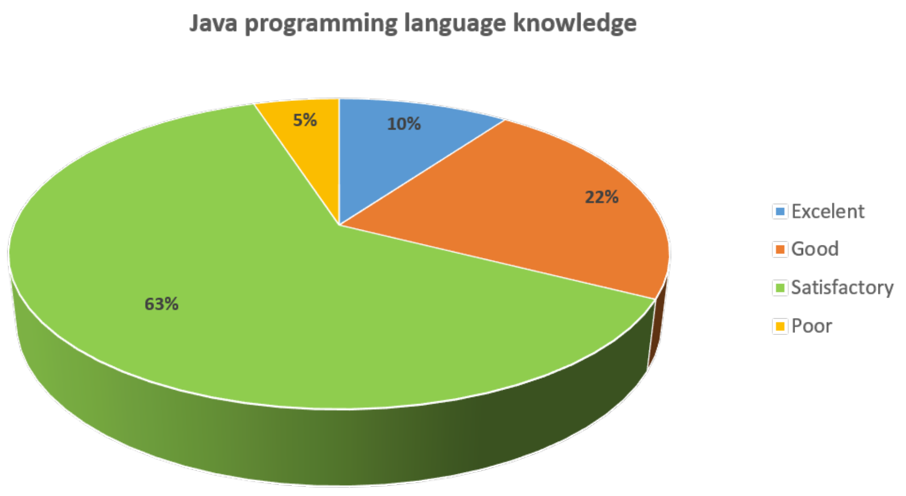
We chose ADempiere[1]Business Suite, an industrial open-source software solution that combines ERP, CRM and SCM support for business process. This system is a large size project with a well documented bug report system and an active community. For the controlled experiment we considered the version 3.8 launched on March 1th 2015, which has approximately 11 millions LOC, 68 % of which are written in Java, 5.234 classes, 376 packages, and 89.335 methods.

4.6. Participants

The participants were 40 undergraduate students enrolled in a software engineering course at Universidad Nacional de Colombia. We divided them into two groups according to their knowledge and experience with the Java programming language, Eclipse IDE, and Eclipse searching features.

Figure **4-1** shows that 63 % of participants considered that their Java knowledge is satisfactory, while 22 % of them said that it is good, and 10 % expressed that they have an excellent knowledge of Java. We conclude that most of the participants have an intermediate knowledge of Java programming language so that they are competent enough to participate in the experiment.

Figure 4-1.: Java knowledge of participants

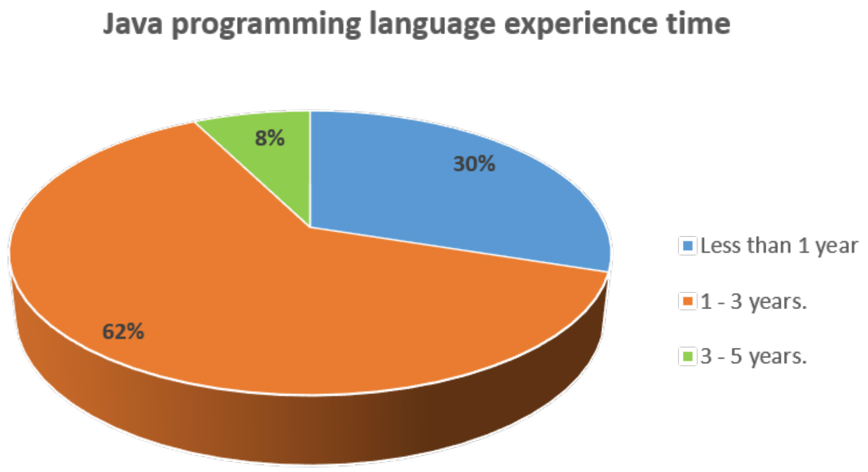


Regarding programming experience, most of the participants (62 %), as shown in Figure **4-2**, said they have between 1 and 3 years of Java programming experience. 30 % of participants expressed

[1]www.adempiere.net

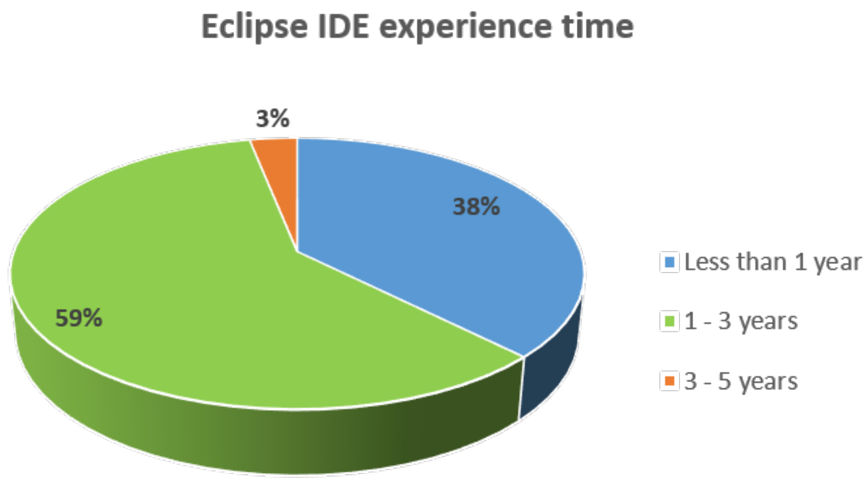
that their programming experience is less than 1 year, and only the 8 % of subjects say that they have from 3 to 5 years of experience time in Java programming.

Figure 4-2.: Java programming experience of participants



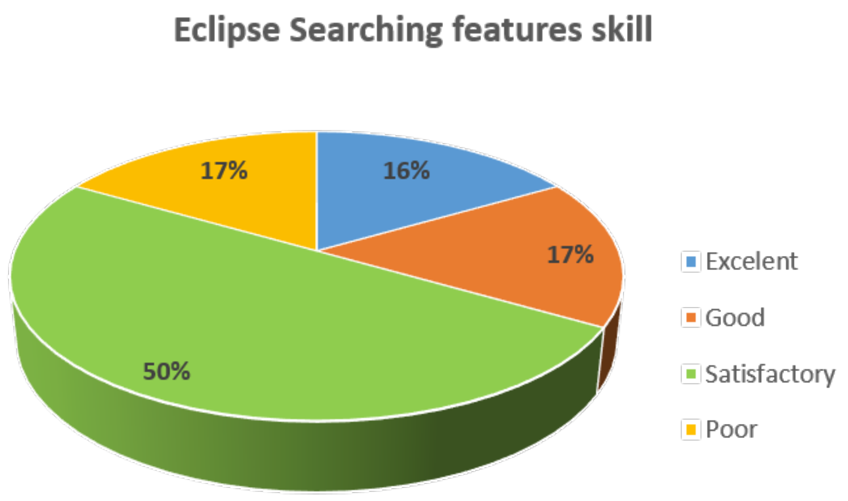
Only 13 % of participants indicated that they are currently working as developers in an enterprise. 80 % of participants expressed they have experience using the Eclipse IDE. Figure **4-3** shows how much experience the participants that have used the Eclipse IDE.

Figure 4-3.: Eclipse IDE experience of participants



We asked to participants who have experience using Eclipse IDE if they are familiarized with the searching features of Eclipse. The 19 % of Eclipse experienced users express that they have used this feature. Then, we requested to that participants to evaluate their skill using Eclipse searching functionalities. Figure **4-4** shows the results to this question.

Figure 4-4.: Eclipse searching feature skills of participants



We did maintain a balance between the groups with respect that expertise as much as possible, according to the demographical survey results. One group, named as henceforth the ***Tool group***, used the Eclipse plug-in that implements the IR+RF approach for concept location. The other group, that we named the ***No-Tool group***, used the default Eclipse features for searching.

4.7. Tasks

We chose five bugs reports from the ADempiere issue tracker system [2]and reviewed approved patches to version 3.8.0. By contrasting the correction reported in further versions for each bug, we determined the method that was modified in order to fix the bug. This approach was used by Gay et al. [4]. Table **4-2** shows the title, description, the buggy or target method, and the URL for each bug. We chose bugs where only one target method was modified in order to fix it. We kept titles and descriptions as appear in the ADempiere bug tracker. There were poor quality explanations and spelling and grammatical errors in some titles and bug descriptions. We decided not to fix them and use original text so that participants dealt with some of the actual problems of software engineering industry. Those bugs were used for the pilot test and the controlled experiment.

[2]https://github.com/adempiere/adempiere/issues/

Table 4-2.: Relevant information of bugs

Bug	Title	Description	Target Method	URL
1	Process Class Generator not get parameters type correctly.	When a parameter is of type FilePathOrName, its created like Object on getter method on Process abstract generated.	Class: DisplayType Method: boolean isText(int displayType)	`https://github.com/adempiere/adempiere/issues/677`
2	DRP not generate distribution order because business partner not link organization.	DRP not generate order , the issue that business partner are try find is based on source locator organization. The system should use of organization from target locator.	Class : MRP Method: void createDDOrder(int AD_Org_ID, int PP_MRP_ID, MProduct product,BigDecimal QtyPlanned ,Timestamp DemandDateStartSchedule, String trxName)	`https://github.com/adempiere/adempiere/issues/83`
3	The tables mark like IsDocument is deleteable.	Currently the tables mark like IsDocument are deleteables, it is bad for the document, all document that handle document number must be no deleteable.	Class: MTable Method: boolean beforeSave (boolean newRecord).	`https://github.com/adempiere/adempiere/issues/657`
4	Error when try merge product entity.	When try merge from a duplicate product , the error throw exception because not can delete product because have a dependence with cost dimension.	Class: Merge Methdod: int mergeTable (String TableName, String ColumnName, int from_ID, int to_ID)	`https://github.com/adempiere/adempiere/issues/595`
5	Get all employees for payroll process, should get all active or not active.		Class: MHREmployee Method:MBPartner[] getEmployees (MHRProcess process)	`https://github.com/adempiere/adempiere/issues/634`

4.8. Procedure

We conducted a pilot test of the controlled experiment before applying the controlled experiment with the participants. Two undergraduate students with knowledge and expertise comparable to the participants helped us to measure the difficulty of tasks and estimate the time required to complete the experiment, among other circumstances that we could control. One student used the IR+RF plug-in and another used the default Eclipse searching features. After running the pilot test experiment, we concluded that it is necessary to prepare a tutorial session about the subject system, the IR+RF plug-in, and the Eclipse search features. Moreover, we prepared a demographic survey to assess Java and programming skills and balance both groups of the experiment. Furthermore, the tasks must be randomly shown to the participants, since in that way we are able to get enough information from all tasks and mitigate learning effects during the experiment.

Two days before conducting the experiment, the participants attended a tutorial session where we presented the ADempiere system, the plugin features (usage and query formulation syntax), and the Eclipse search features. Then, they responded the demographical survey, which allowed us to divide the participants into two groups as homogeneously as possible based on their previous experience with Java and Eclipse.

Before running the controlled experiment we verified that Eclipse Java Neon[3]was properly installed in computers that participants used to perform the experiment. We installed and tested the Eclipse plugin that subjects of Tool group used and an extra plugin that keep track of the actions of participants of the No-Tool group. Besides, we made a summary of the topics given in the tutorial session that we delivered in the controlled experiment to subjects according to the group they were assigned.

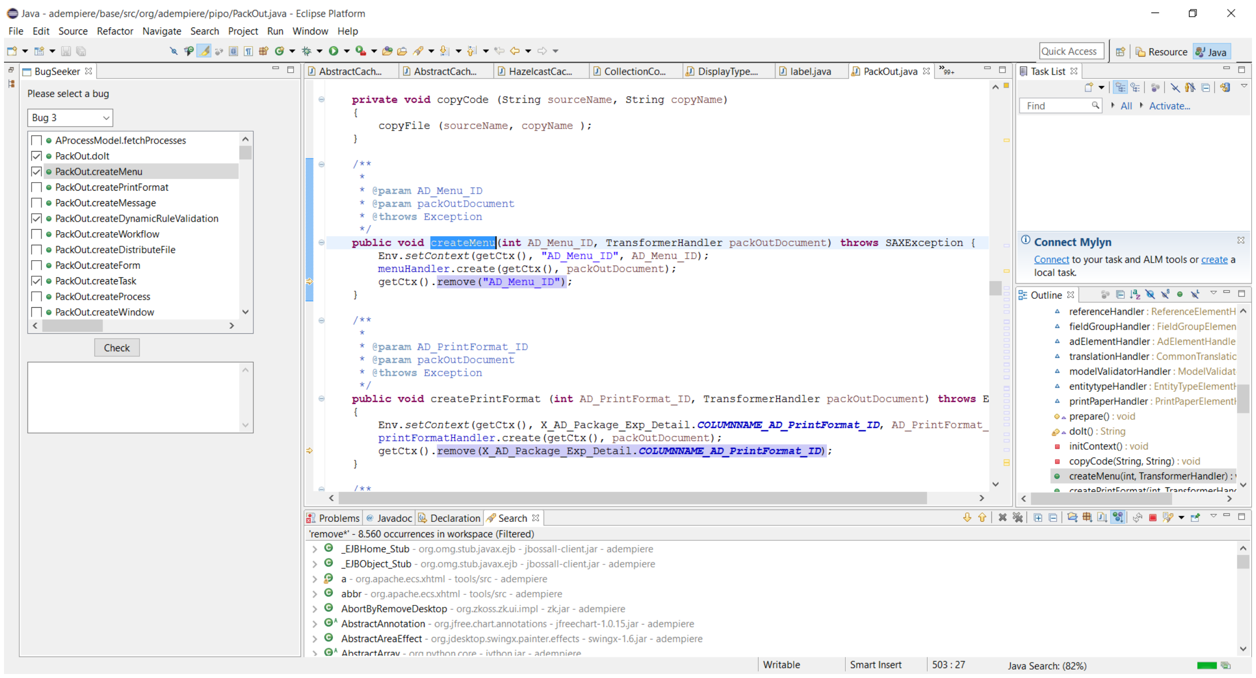
In the experimental session, once the subjects were divided into the two experimental groups, they received the title and description of five bugs, which were shown randomly in a Qualtrics4survey. We measured the time spent by each participant since the bug was shown until she finished the task. Depending on the group, the participants use either the tool or the Eclipse IDE default search features to find the target method where the bug needs to be fixed.

For the No-Tool group, we modified Eclipse in order to track the results of each Java search and allow subjects to select and explore the methods retrieved by a Java Search in the resulting list. Furthermore, we stored how many methods were revised and marked by participants, how many queries or searches were made, and we also monitored the results so that we know when the target method of a specific bug was found. As Figure **4-5** shows the subjects of the N-Tool group

[3]`http://www.eclipse.org/downloads/packages/eclipse-ide-java-developers/neon3`
4`www.qualtrics.com`

can choose the bug for what they are looking for the target method, they have a resulting list similar to the IR+RF that shows the methods retrieved by a Java search that satisfies a search. They can click on a method and examine in the Eclipse Editor how and where it is implemented. With the check button, they can know whether a selected method is the target method for the bug at hand. At the end of each task, in the textbox is reported the marked and revised methods, the number of queries formulated and the effectiveness of task. We did the same in the Tool group to keep track of the actions of both groups. Figure **4-6** displays the IR+RF plugin modified so as to track actions of the Tool group. It looks like the Eclipse plugin for the No-Tool group with the difference that we stored how many times the feedback mechanism was used.

Figure 4-5.: Eclipse plugin for tracking the No-Tool participants' actions



Please select a bug

Bug 3

- [ ] AProcessModel.fetchProcesses
- [x] PackOut.doIt
- [x] PackOut.createMenu
- [ ] PackOut.createPrintFormat
- [ ] PackOut.createMessage
- [x] PackOut.createDynamicRuleValidation
- [ ] PackOut.createWorkflow
- [ ] PackOut.createDistributeFile
- [ ] PackOut.createForm
- [x] PackOut.createTask
- [ ] PackOut.createProcess
- [ ] PackOut.createWindow

```java
    private void copyCode (String sourceName, String copyName)
    {
        copyFile (sourceName, copyName );
    }

    /**
     *
     * @param AD_Menu_ID
     * @param packOutDocument
     * @throws Exception
     */
    public void createMenu(int AD_Menu_ID, TransformerHandler packOutDocument) throws SAXException {
        Env.setContext(getCtx(), "AD_Menu_ID", AD_Menu_ID);
        menuHandler.create(getCtx(), packOutDocument);
        getCtx().remove("AD_Menu_ID");
    }

    /**
     *
     * @param AD_PrintFormat_ID
     * @param packOutDocument
     * @throws Exception
     */
    public void createPrintFormat (int AD_PrintFormat_ID, TransformerHandler packOutDocument) throws E
    {
        Env.setContext(getCtx(), X_AD_Package_Exp_Detail.COLUMNNAME_AD_PrintFormat_ID, AD_PrintFormat_
        printFormatHandler.create(getCtx(), packOutDocument);
        getCtx().remove(X_AD_Package_Exp_Detail.COLUMNNAME_AD_PrintFormat_ID);
    }

    /**
```

Connect Mylyn

Connect to your task and ALM tools or create a local task.

Outline

```
referenceHandler : ReferenceElementH
fieldGroupHandler : FieldGroupElemen
adElementHandler : AdElementHandle
translationHandler : CommonTranslatic
modelValidatorHandler : ModelValidat
entitytypeHandler : EntityTypeElementH
printPaperHandler : PrintPaperElementH
prepare() : void
doIt() : String
initContext() : void
copyCode(String, String) : void
createMenu(int, TransformerHandler) :
```

```
'remove*' - 8.560 occurrences in workspace (Filtered)
> _EJBHome_Stub - org.omg.stub.javax.ejb - jbossall-client.jar - adempiere
> _EJBObject_Stub - org.omg.stub.javax.ejb - jbossall-client.jar - adempiere
> a - org.apache.ecs.xhtml - tools/src - adempiere
> abbr - org.apache.ecs.xhtml - tools/src - adempiere
> AbortByRemoveDesktop - org.zkoss.zk.ui.impl - zk.jar - adempiere
> AbstractAnnotation - org.jfree.chart.annotations - jfreechart-1.0.15.jar - adempiere
> AbstractAreaEffect - org.jdesktop.swingx.painter.effects - swingx-1.6.jar - adempiere
> AbstractArray - org.python.core - jython.jar - adempiere
```

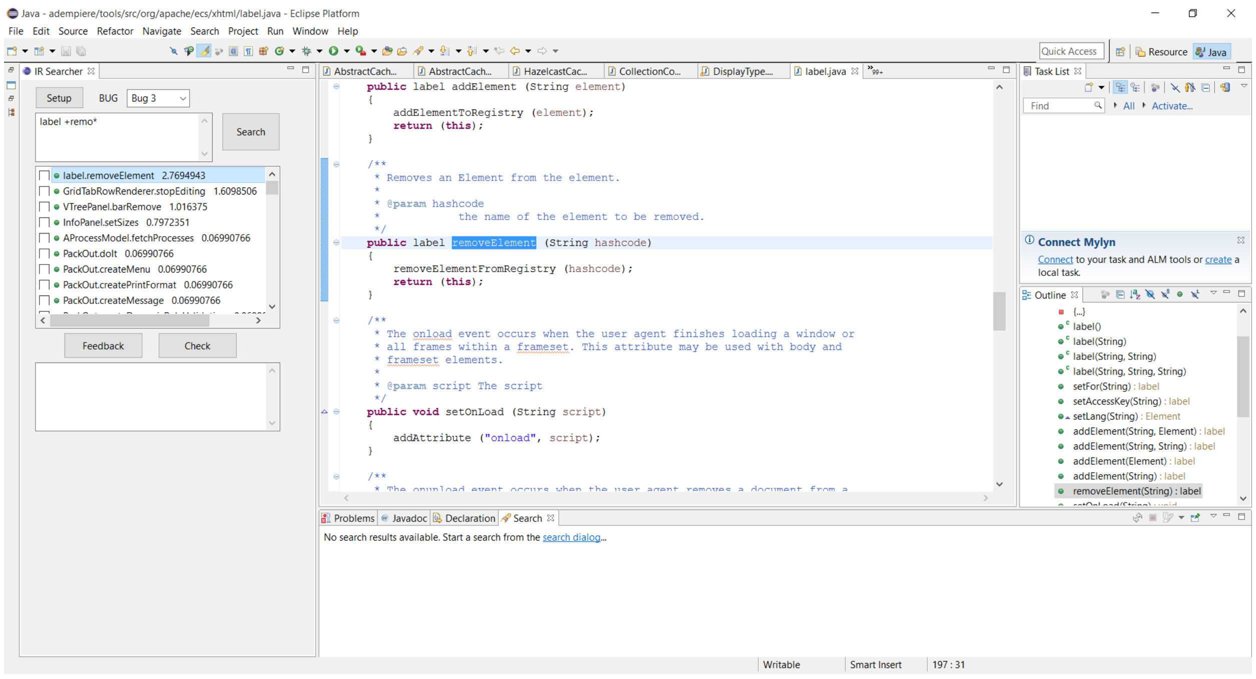
The experimental session was limited to two hours. The following instructions were given to the participants at the beginning of the session:

- In this experiment, you will find the description of five bugs. For each bug, you have to find what method needs to be modified in order to fix the bug. That method is called *the Target Method*. There is only one target method for each bug.
- In order to make the initial query, you can use terms from the bug title or description as keywords.
- You are able to formulate the queries and freely revise the resulting list of methods. Once you have obtained the results of a search, you have to mark the methods that you believe have to be modified in order to fix the bug, and then, click the check button in order to

verify whether the target method has been found. You can mark as relevant maximum 30 methods in total.

- If you have not found the target method (after using the check button), you have three options: i) use the feedback (this option is only available for the Tool group), ii) review and select other methods, or iii) reformulate the query to start a new search.
- You can continue until either you have marked 30 methods or the target method is found. In the last case, the task was successfully completed for the bug at hand.
- Once a task is completed, Eclipse shows relevant information such as effectiveness, revised and marked methods, and formulated queries. You must copy this information and paste it in the Qualtrics survey.

Figure 4-6.: IR+RF plug-in for tracking the Tool participants' actions



Java - adempiere/tools/src/org/apache/ecs/xhtml/label.java - Eclipse Platform

BUG: Bug 3

label +remo*

label.removeElement 2.7694943
GridTabRowRenderer.stopEditing 1.6098506
VTreePanel.barRemove 1.016375
InfoPanel.setSizes 0.7972351
AProcessModel.fetchProcesses 0.06990766
PackOut.doIt 0.06990766
PackOut.createMenu 0.06990766
PackOut.createPrintFormat 0.06990766
PackOut.createMessage 0.06990766

```
    public label addElement (String element)
    {
        addElementToRegistry (element);
        return (this);
    }

    /**
     * Removes an Element from the element.
     *
     * @param hashcode
     *            the name of the element to be removed.
     */
    public label removeElement (String hashcode)
    {
        removeElementFromRegistry (hashcode);
        return (this);
    }

    /**
     * The onload event occurs when the user agent finishes loading a window or
     * all frames within a frameset. This attribute may be used with body and
     * frameset elements.
     *
     * @param script The script
     */
    public void setOnLoad (String script)
    {
        addAttribute ("onload", script);
    }

    /**
```

Connect Mylyn

Connect to your task and ALM tools or create a local task.

{...}
label()
label(String)
label(String, String)
label(String, String, String)
setFor(String) : label
setAccessKey(String) : label
setLang(String) : Element
addElement(String, Element) : label
addElement(String, String) : label
addElement(Element) : label
addElement(String) : label
removeElement(String) : label

No search results available. Start a search from the search dialog...

For the experiment, we used the feedback parameters as $\alpha = 1$, $\beta = 1$ and $\gamma = 1$, since according to Gay et al. [4] these seem to yield the best performance.

Once the five tasks were completed a post-experiment questionnaire was used to collect participants' thoughts about relevance feedback as a code searching support, and their opinions about the experiment itself.

5. Results Analysis and Discussion

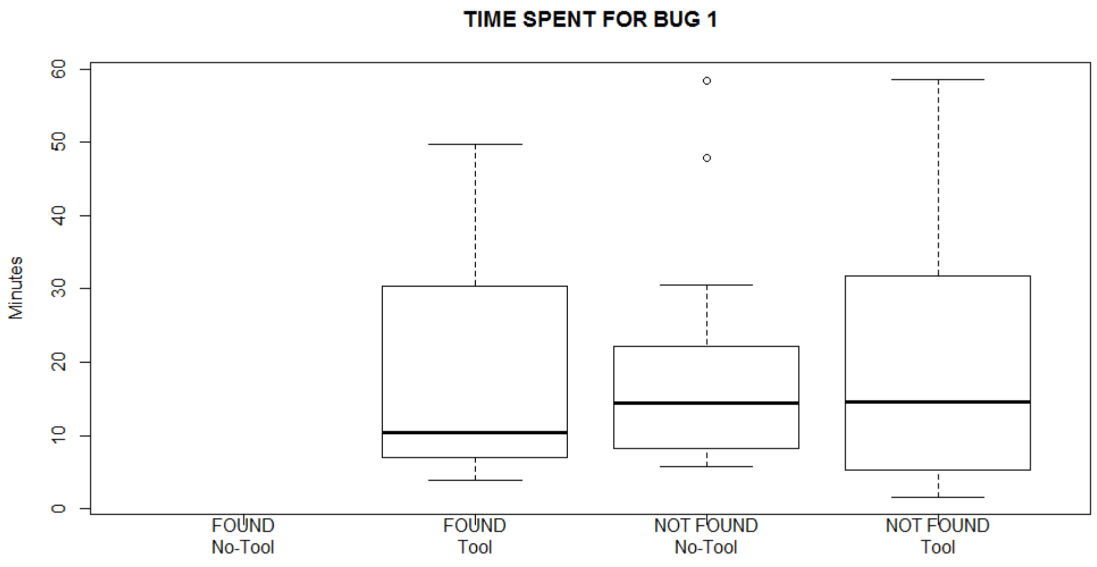
5.1. Analysis of Results

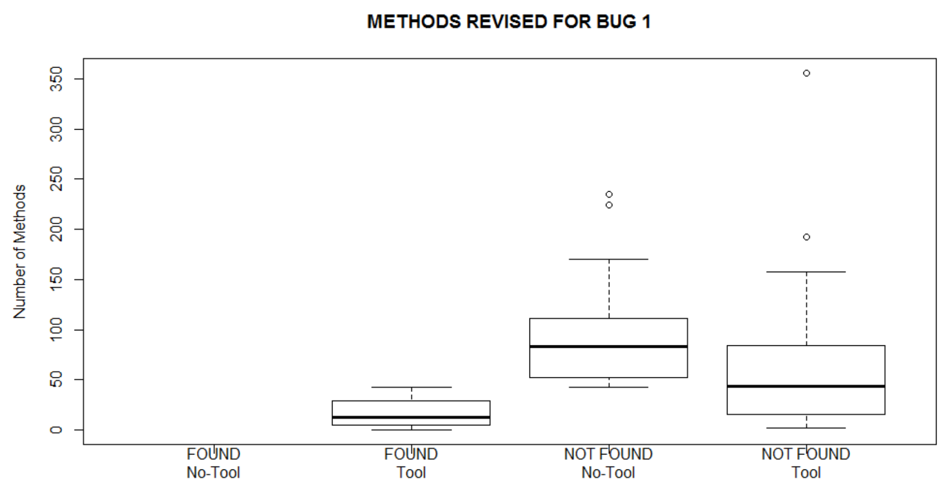
We did a per task graphical analysis so as to see what was the performance of developers, since we measured time spent, revised and marked methods, and queries formulated by developers that successfully completed the tasks.

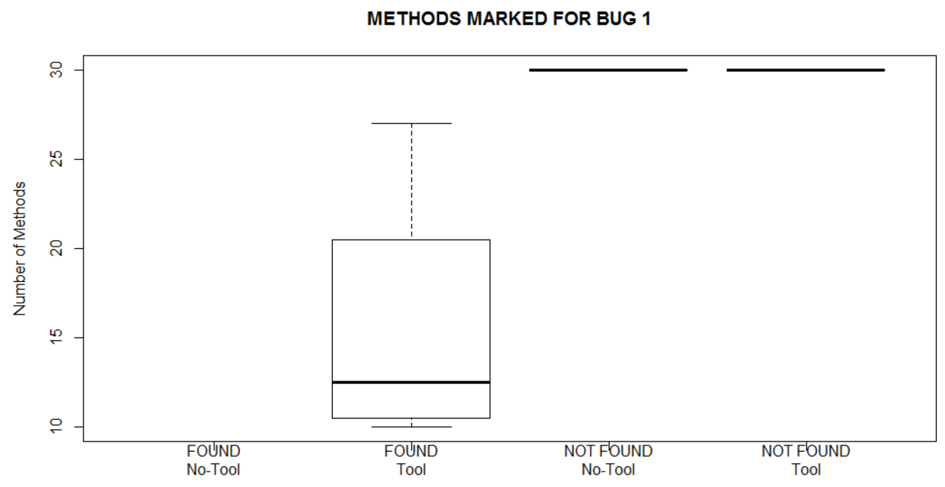
BUG 1

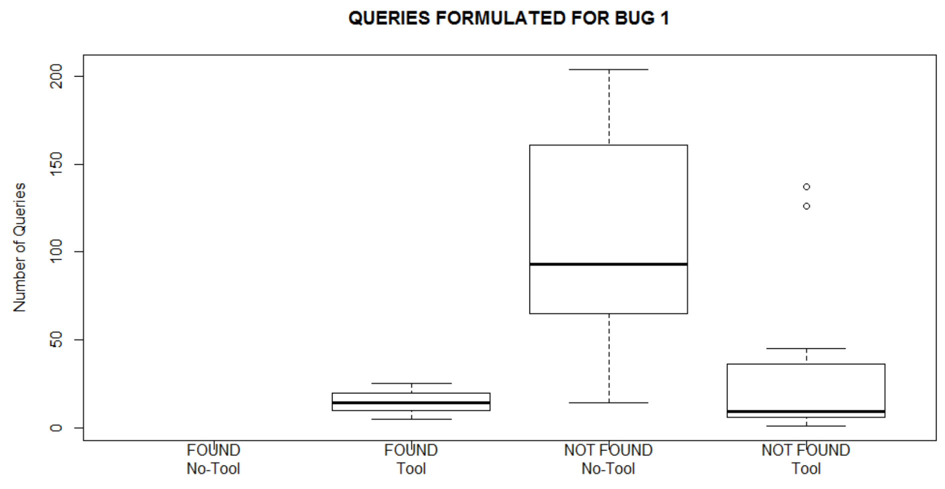
Since in bug 1 none of the participants in the No-Tool group successfully completed the task, we concluded that the Tool group outperformed the No-Tool group. Figure **5-1** shows the performance variables and time spent by participants for this task.

Figure 5-1.: Box plots of time spent, marked and revised methods, and queries formulated for BUG 1




METHODS REVISED FOR BUG 1
Number of Methods
0
50
100
150
200
250
300
350
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

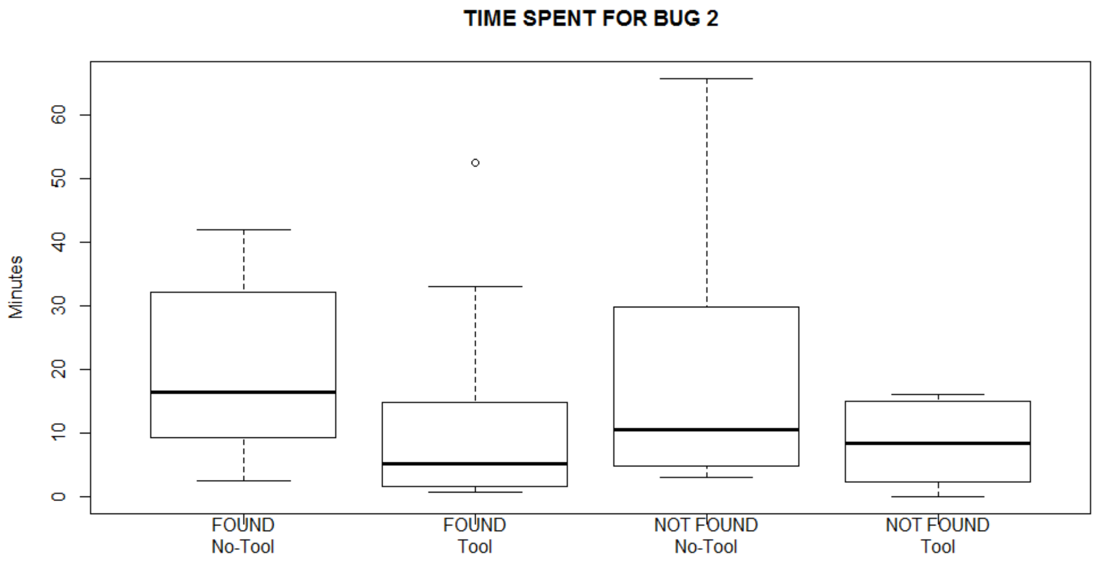

METHODS MARKED FOR BUG 1
Number of Methods
10
15
20
25
30
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

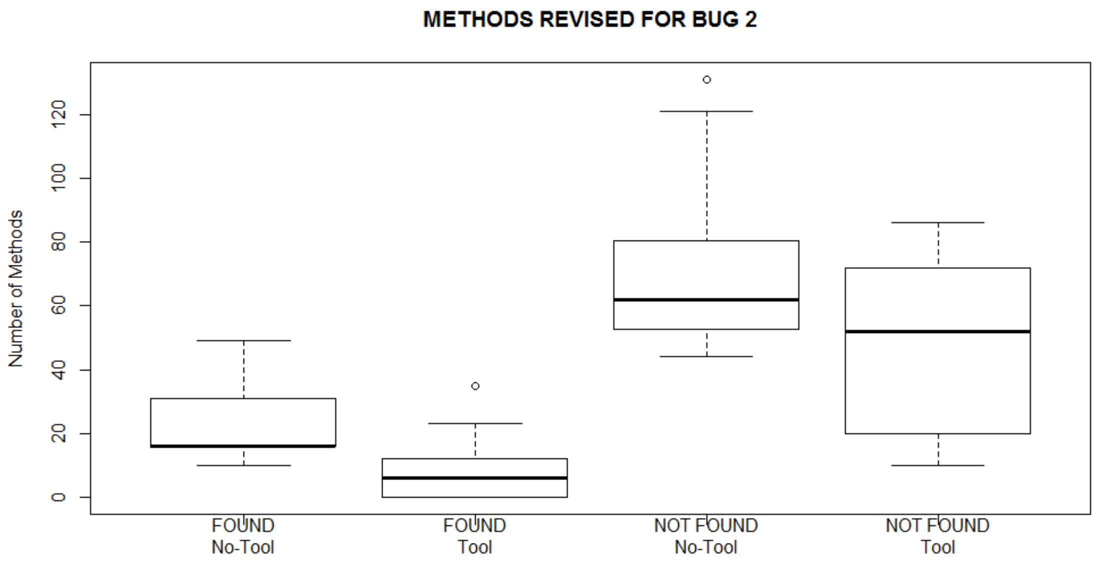

QUERIES FORMULATED FOR BUG 1
Number of Queries
0
50
100
150
200
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

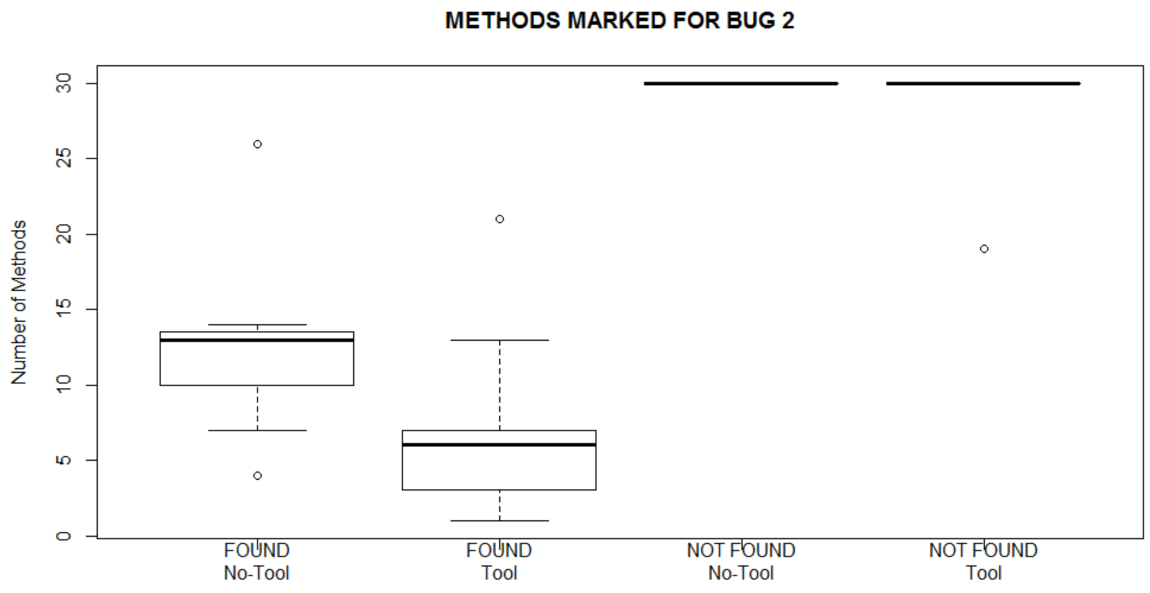
BUG 2

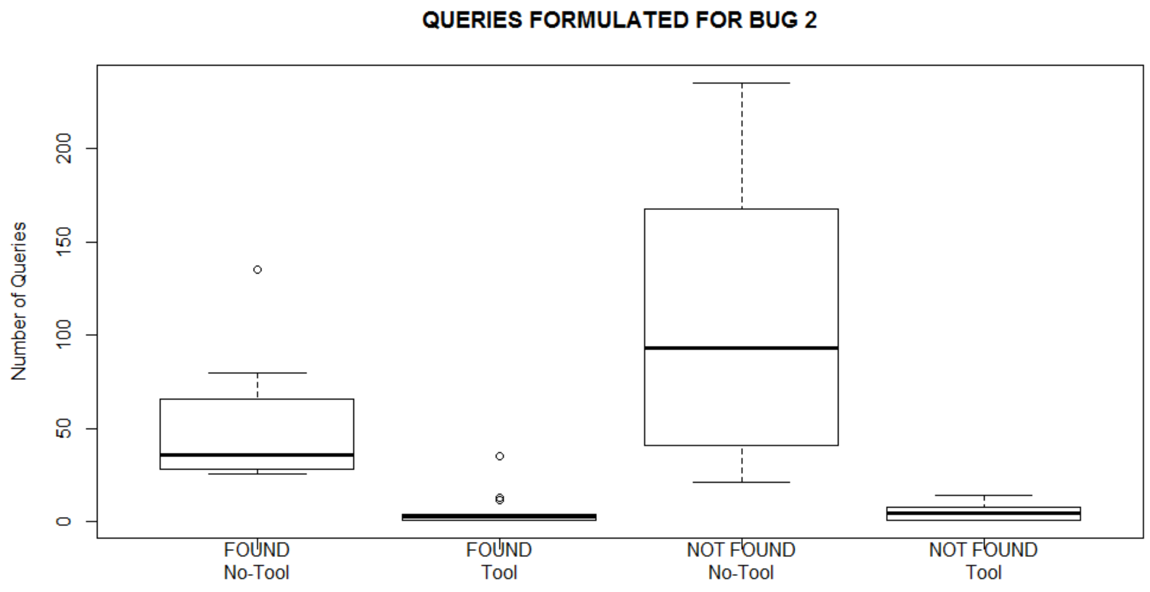
In this task, the effectiveness is greater in the Tool group than in the No-Tool group (11 vs 7). As we can see in Figure **5-2**, in other performance variables the subjects of the Tool group that successfully completed the task surpassed in average to the No-Tool group.

Figure 5-2.: Box plots of time spent, marked and revised methods, and queries formulated for BUG 2





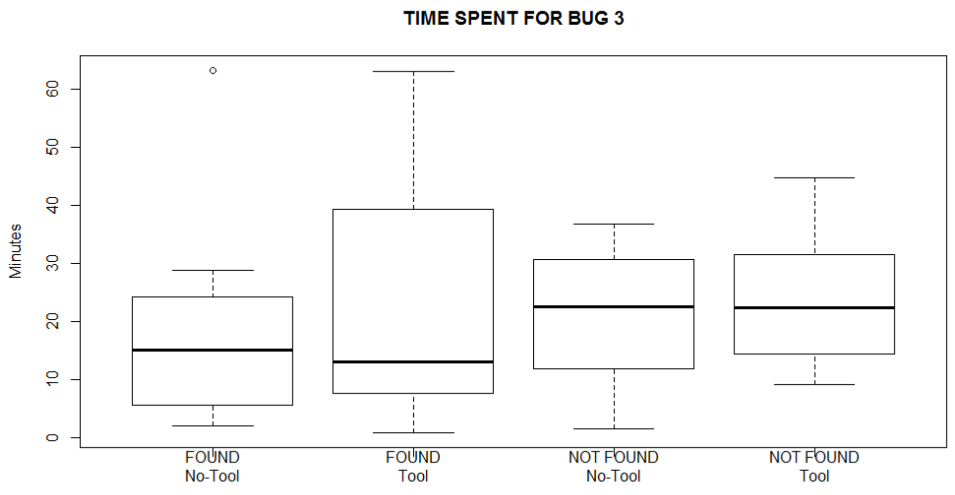


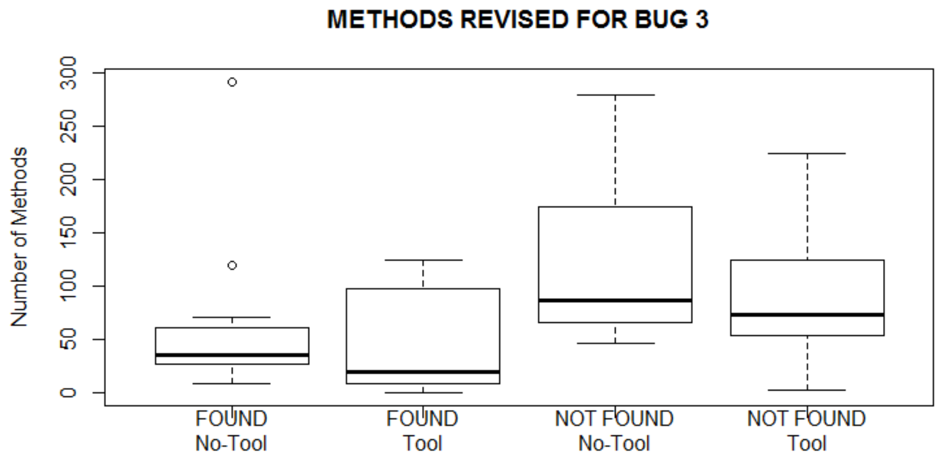


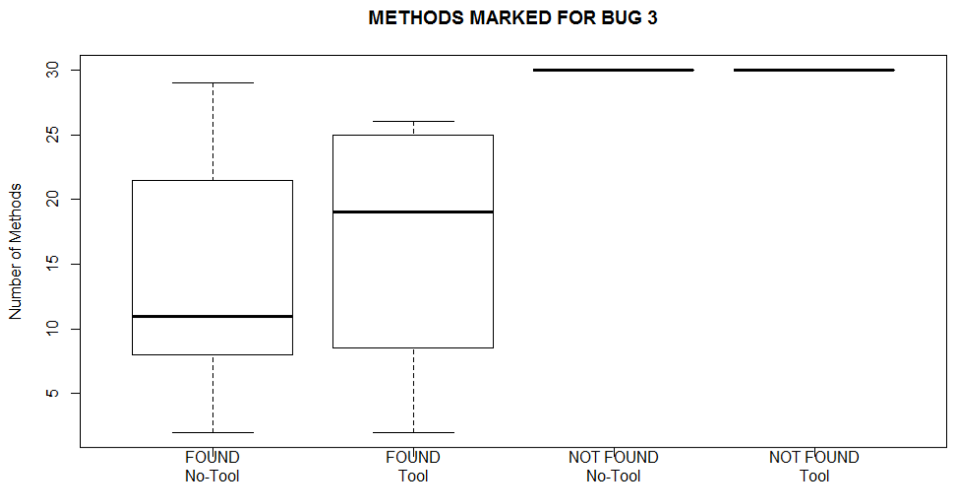
BUG 3

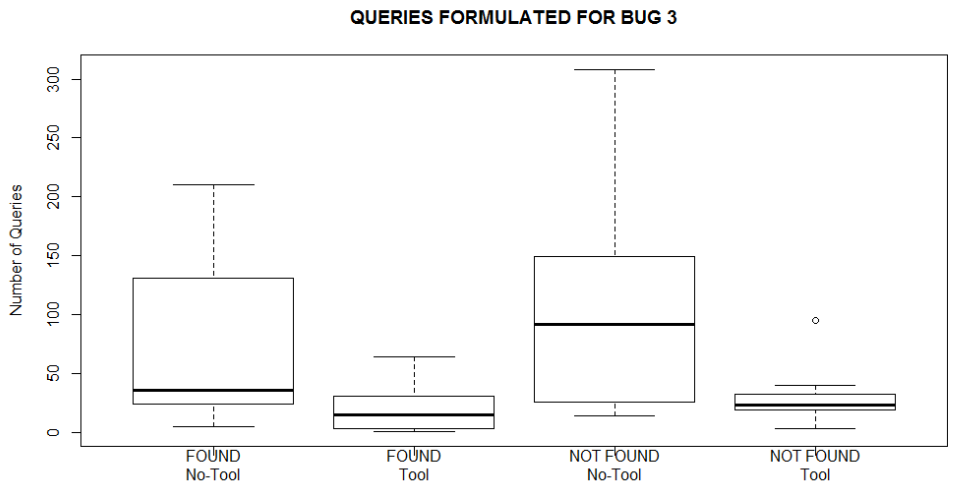
Figure **5-3** shows that the performance of participants of the No-Tool group that found the target method slightly surpassed to the subjects of the Tool group. Only in the number of formulated queries, the difference seems to be significant. We believe that the effectiveness of the Tool group could be seem affected because some terms of the title and bug description benefit the No-Tool group, since the query formulation is different for this group.

Figure 5-3.: Box plots of time spent, marked and revised methods, and queries formulated for BUG 3





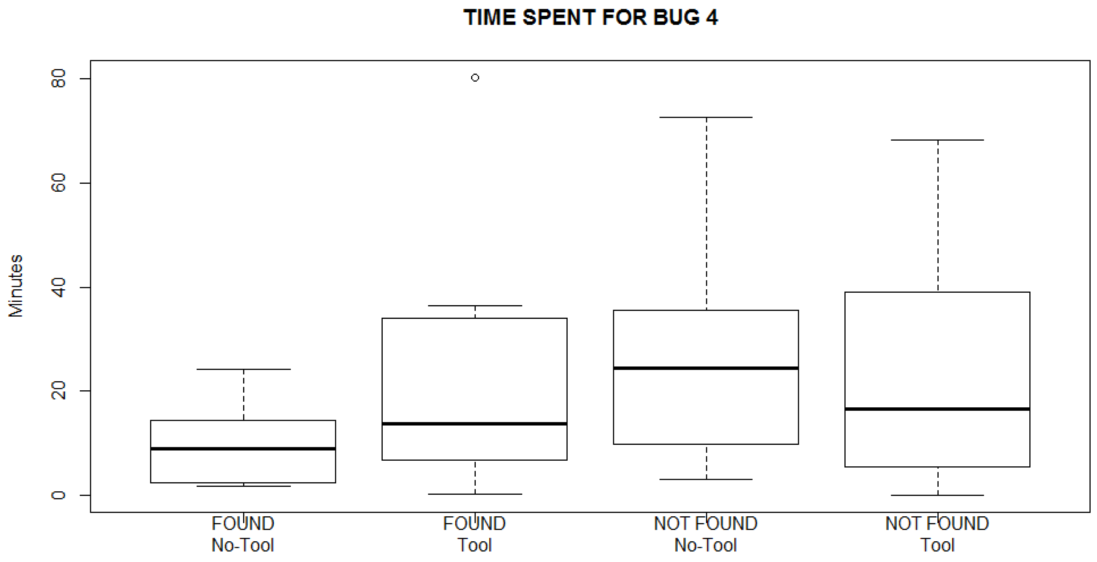


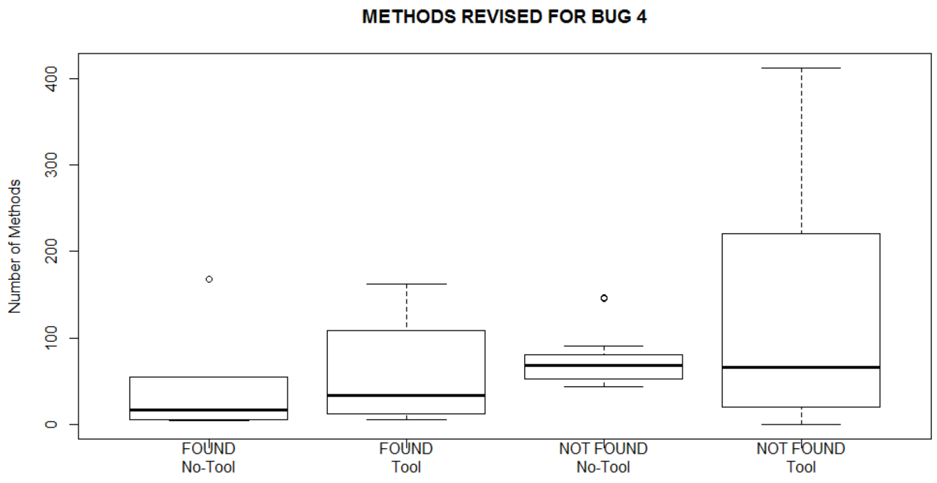


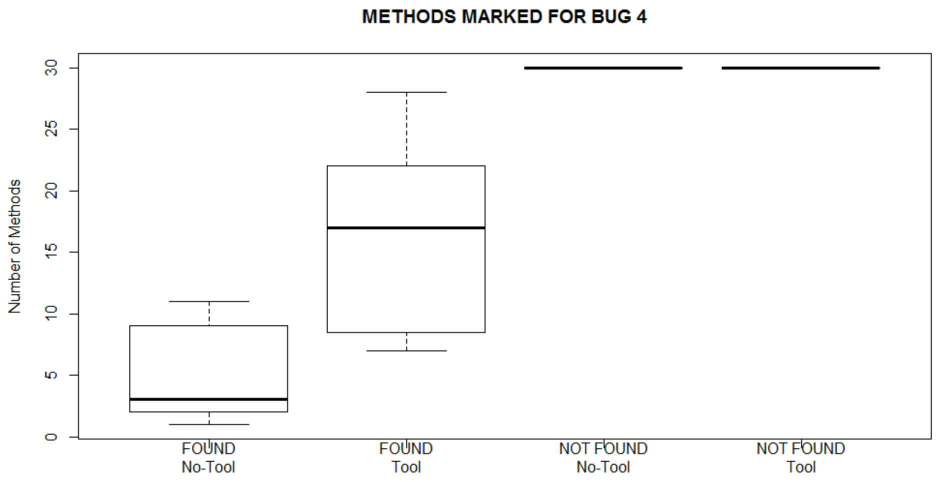
BUG 4

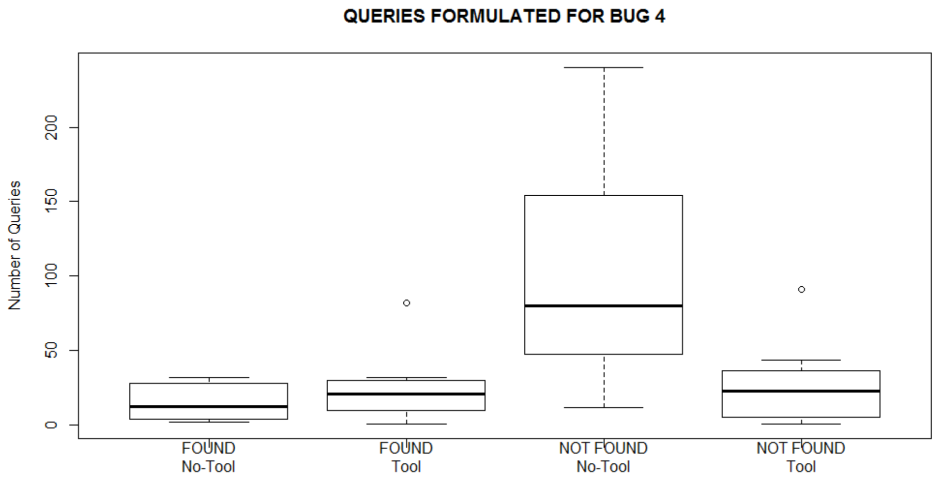
As regards performance variables, Figure **5-4** shows that there is a small difference in favor of the No-Tool group, Respect to the formulated queries measure we see that the difference did not seem to be significant between both groups. Nevertheless, the effectiveness of the Tool group is greater than the No-Tool group.

Figure 5-4.: Box plots of time spent, marked and revised methods, and queries formulated for BUG 4




METHODS REVISED FOR BUG 4
Number of Methods
0
100
200
300
400
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

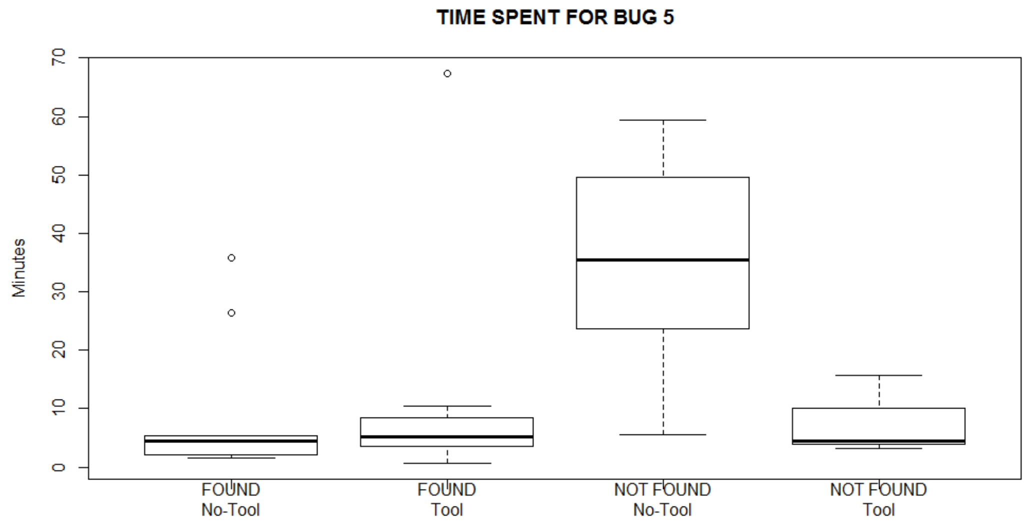

METHODS MARKED FOR BUG 4
Number of Methods
0
5
10
15
20
25
30
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

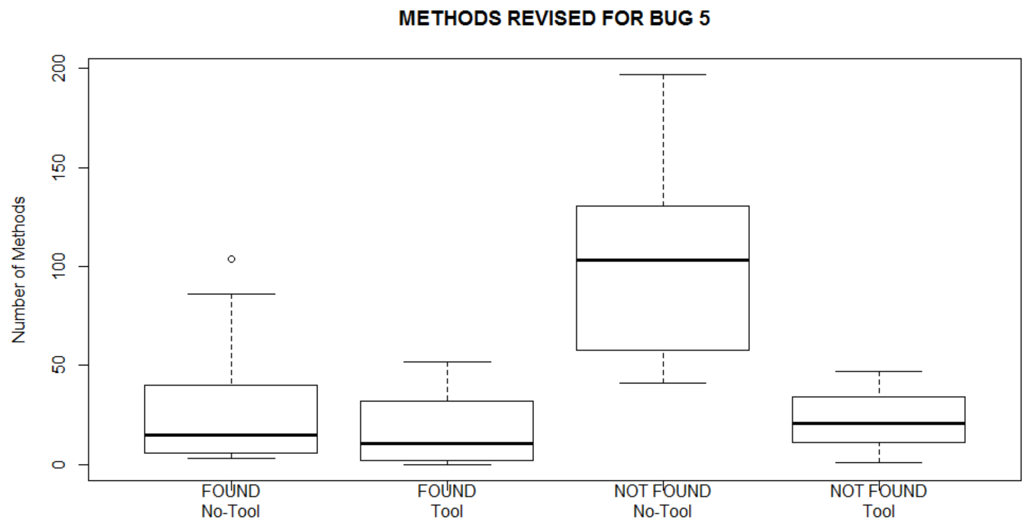

QUERIES FORMULATED FOR BUG 4
Number of Queries
0
50
100
150
200
FOUND No-Tool
FOUND Tool
NOT FOUND No-Tool
NOT FOUND Tool

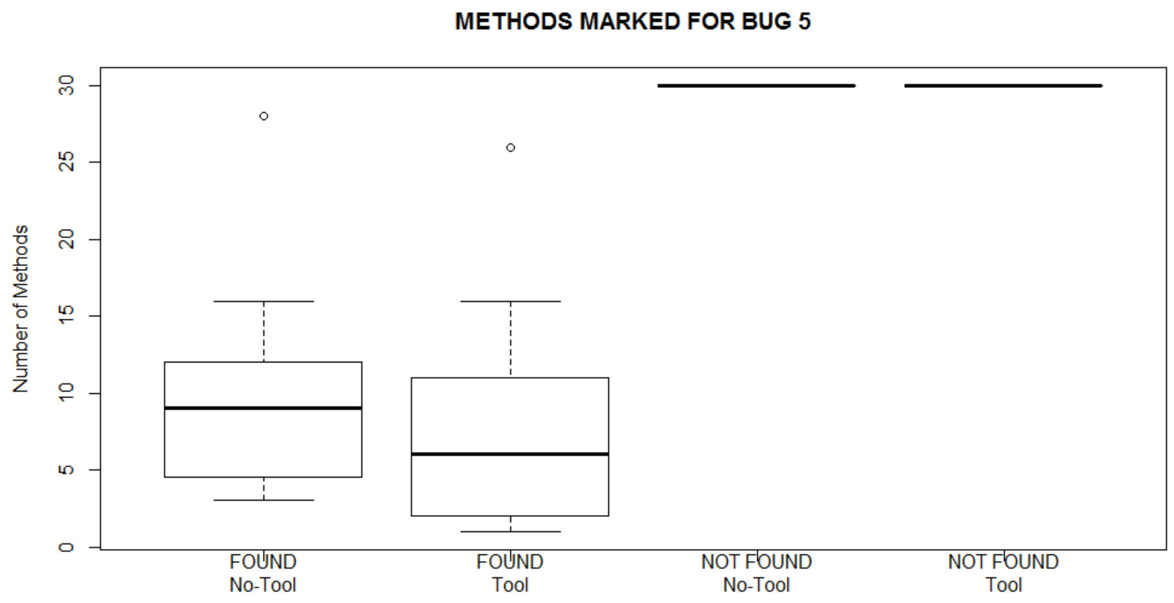
BUG 5

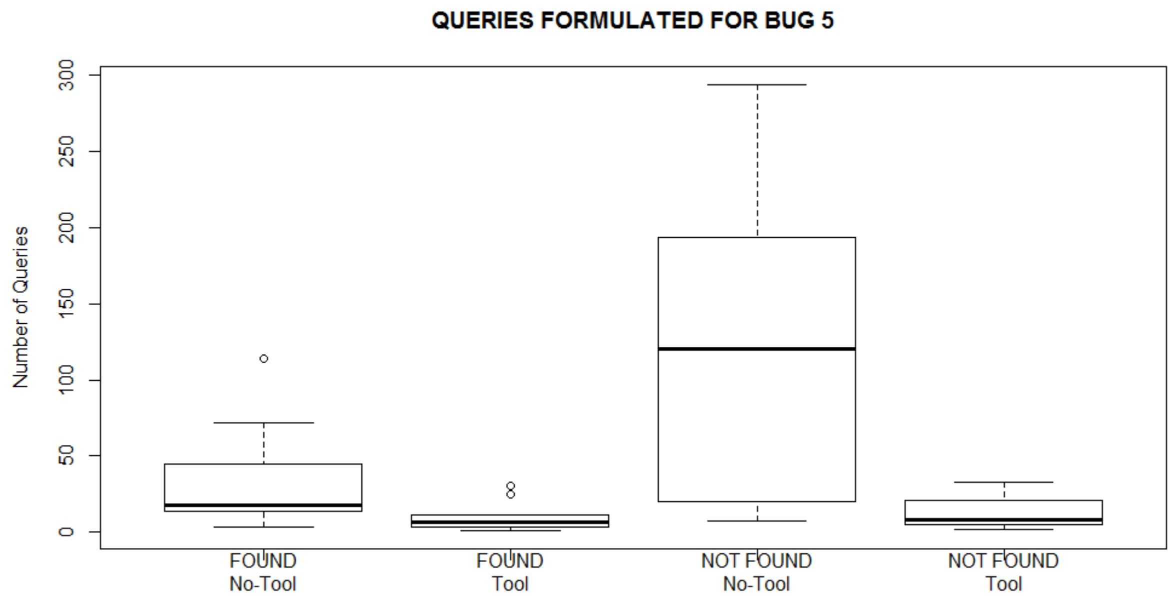
As we can see in Figure **5-5**, the Tool group slightly outperformed the No-Tool group in performance variables such as the number of revised and marked methods. Regarding time spent, there was a small difference between both groups. With respect to queries formulated, the Tool group surpassed the No-Tool group, since in this performance variable the subjects of the Tool group had to make fewer queries to successfully complete the tasks.

Figure 5-5.: Box plots of time spent, marked and revised methods, and queries formulated for BUG 5









5.2. Research Question 1

In order to check if there is a significant difference between the performance variables of both groups we had to perform a hypothesis test for each one of this variables. Before making hypothesis testing we performed several statistical normality tests to check whether there is normality in the results. We applied five tests available in 'nortest' package[21] of R[1]. For each numerical variable collected for each bug (time spent, marked methods, revised methods and queries formulated) we performed Anderson-Darling Cramer-von Mises, Lilliefors (Kolmogorov-Smirnov), Pearson chi-square, and Shapiro-Francia tests for normality (with $\alpha = 0{,}05$) and found that the p-values of each test are lower than 0.05. Thus, none of the variables is normally distributed.

Therefore, we applied Mann-Whitney test [22], a well known non-parametric test. We compared the results of the Tool and No-Tool groups who completed the tasks, i.e. the participants who found the target method of each bug. First, we perform the hypothesis tests that we stated in Table **4-1**. Since for bug 1 none of the No-Tool participants completed the task, we concluded that in bug 1 the Tool group outperformed the No-Tool group. Table **5-1** shows that there are meaningful differences in highlighted performance variables for each bug, since p-values are less than 0.05, and therefore, the null hypothesis must be rejected.

Table 5-1.: P-values of Mann-Whitney hypothesis tests (Tool vs. No-Tool)

Bug	Time Spent	Revised Methods	Marked Methods	Formulated Queries
2	0.114600	0.008218	0.031100	0.000820
3	0.791400	0.525900	0.617500	0.049790
4	0.413600	0.365600	0.016680	0.516300
5	0.291500	0.323800	0.271800	0.009190

We need to know what group has better performance, so we performed a hypothesis test where H_0 : the mean of performance variables of No-Tool group is less or equal than the mean of performance variables of Tool group, and H_1 : the mean of the performance variables of No-Tool group is greater than the mean of the performance variables of Tool group. We only did this hypothesis test on performance variables where the previous test showed meaningful differences.

Table 5-2.: P-values of Mann-Whitney hypothesis tests (Tool vs. No-Tool)

Bug	Time Spent	Revised Methods	Marked Methods	Formulated Queries
2		0.004109	0.015550	0.000410
3				0.025400
4			0.994200	
5				0.004595

Table **5-2** shows that highlighted p-values are less 0.05 so that the null hypothesis should be rejected in favor of the alternative hypothesis. It means that with exception of bug 4, for the

[1]`https://www.r-project.org/`

number of formulated queries by bug, the Tool group outperformed the No-Tool group, i.e., the members of the Tool group had to formulate fewer queries than the participants of the No-Tool group to successfully complete the tasks. Regarding the number of revised and marked methods, only in bug 2 the subjects of the Tool group had to revise and mark fewer methods than the No-Tool group members to complete this task. With respect to time spent, we did not find statistical evidence of a significant difference between the Tool group and No-Tool Group. On the other hand, the No-Tool group participants outperformed the Tool group in bug 4, since they had to mark fewer methods for completing the task, that is to say, it was the only one performance measure where there is statistical evidence that the No-Tool group surpassed the Tool group.

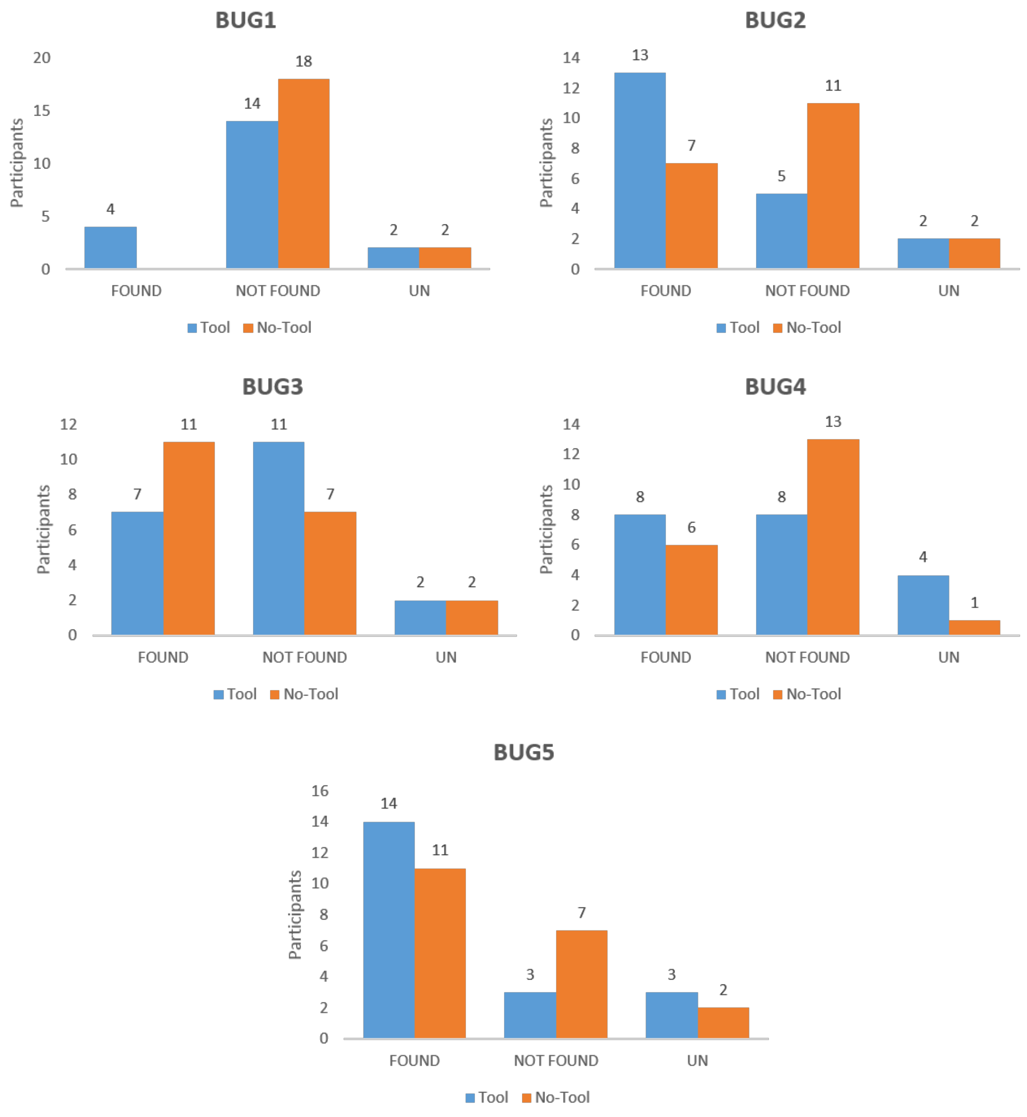
5.3. Research Question 2

First of all, we count how many participants (not) found the target method for each bug. Some participants did not have enough time to complete all tasks so that some bugs are considered as unexplored (UN).

Regarding effectiveness, we found that the tool group surpassed the No-Tool group because in almost all tasks the effectiveness is greater. As shown in Figure **5-6**, for almost all bugs the participants in the Tool group had better performance than those in the No-Tool group, in terms of target methods found. In 4 out of 5 bugs, more participants were able to find the buggy method by using the tool vs the participants that used Eclipse search. For bug1 no one in the No-Tool group found the target method. Moreover, in average the Tool group subjects found 2.3 buggy methods while the No-Tool group participants found 1.75. Bug 3 was an exceptional case where the No-Tool group outperformed the Tool group. We think some terms in the title or bug description may negatively influence the performance of the Tool group since participants took keywords from there to write queries. In conclusion, the Tool group outperformed No-Tool group since, in almost all tasks the effectiveness is greater.

Finally, we made a histogram for the number of target methods found by participants of each group. Figure **5-7** shows that subjects in the Tool group found at least one target method, while in the No-Tool group three subjects did not find any buggy method. Furthermore, the maximum number of target methods found by the No-Tool group was 3, while 4 and 5 target methods were found by some of the participants in the Tool group.
The low effectiveness of both groups is either due to lack of experience of participants or the big size and complexity of the subject system. Mostly of participants have between 1 and 3 years of experience as developers in academic scope. Besides, ADempiere business suite is a large and complex system to deal with. That is why the majority of participants did not find the target methods with or without the eclipse plug-in.

Figure 5-6.: effectiveness of tasks by participants



Finally, we analyzed the results of the post-experiment questionnaire. It collects thoughts of participants about how difficult was to participants to find the target method, the prominence of relevance feedback mechanism for the Tool group, what difficulties had the participants during the experiments and comments about the experiment itself.

Figure **5-8** shows that most of the participants of the No-Tool group think that finding the target method was either very hard or hard, while for subjects of the Tool group the difficulty is distributed homogeneously between very hard, hard and moderate. It was revealed in effectiveness of the tasks since it was greater in participants of the Tool group.

Figure 5-7.: Number of target methods found by participants of each group.

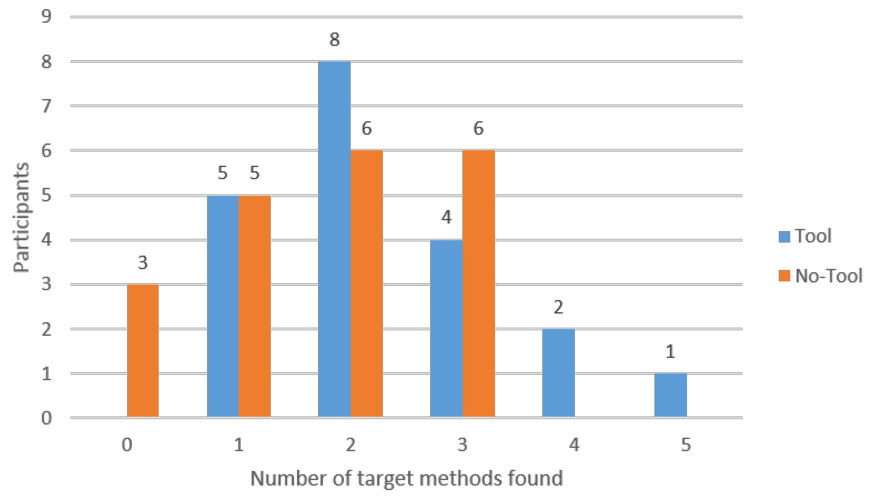
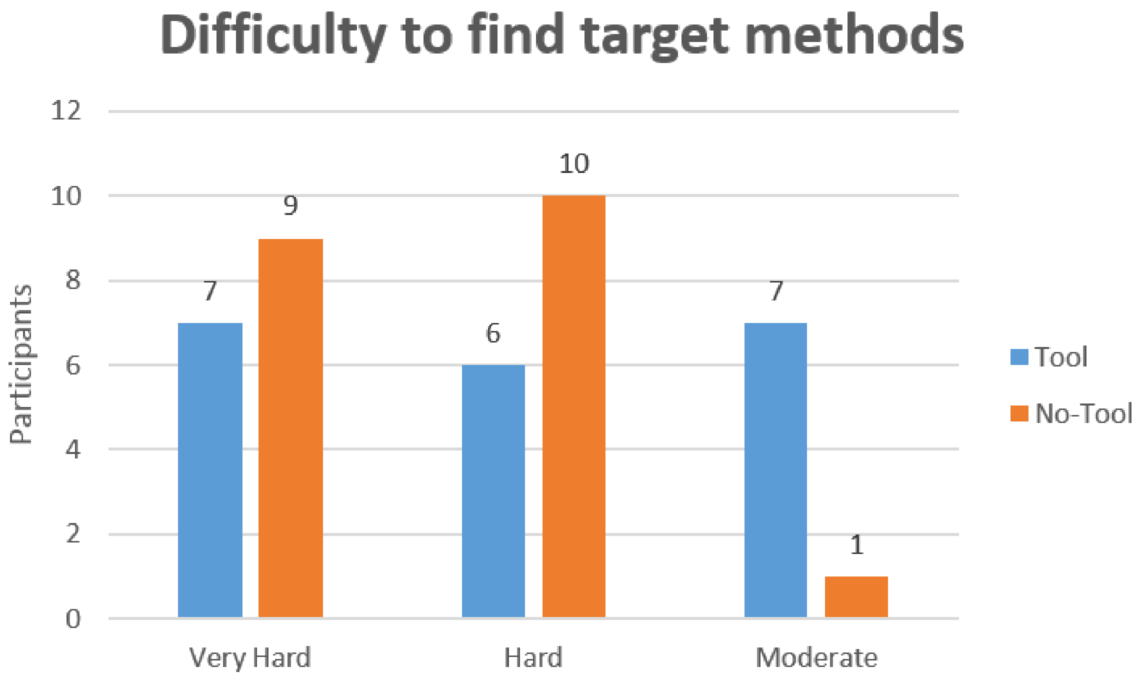


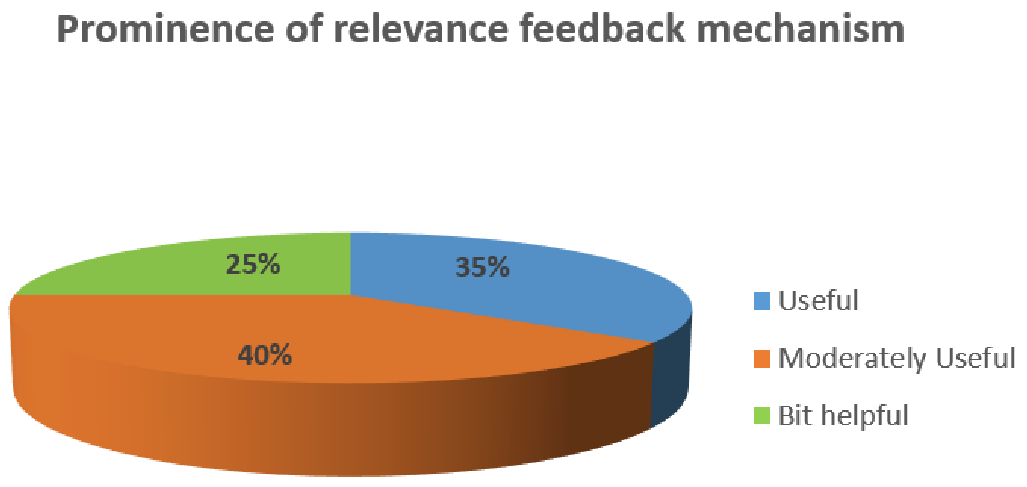
Figure 5-8.: Difficulty to find target methods by participants of each group.



Then we asked the participants of the Tool group about the prominence of the relevance feedback mechanism. Figure **5-9** reveals that the feedback mechanism was either useful or moderate useful for 75 % of participants, while only the remaining 25 % of subjects thought that it was little helpful. We argue that the relevance feedback was very useful to the Tool group since it helps them to increase the effectiveness of their tasks and their performance.

Most of the participants reported that they did not have any drawback during the controlled experiment. Some of the participants reported that they had confusion with the titles and descriptions of the bugs. As we said in the experiment design, the titles and descriptions of bugs were not changed (even they had grammatical of spelling mistakes), since we wanted the participants to deal with real circumstances of the software engineering industry. Other comments are regarding the big size and high complexity of the subject system, as we said before we tried to mitigate this risk through the tutorial session. Nevertheless, the results were good even though the participants did not have enough knowledge as is desired about the subject system.

Figure 5-9.: Prominence of relevance feedback mechanism by participants of the Tool group.



5.4. Threats to validity

In this section, we discuss the threats to the validity of the experiment results. The internal validity refers to unexpected factors that may compete with our independent variable and affect the values obtained for the dependent variables. First of all, to reduce the threat that the participants may not be competent to perform the tasks, we conduct a tutorial session where they learned how to use the Eclipse plug-in for concept location purposes. Moreover, we familiarized them with the main features and modules of the ADempiere Business Suite and reviewed the main search features offered by the Eclipse IDE. Second, based on the answers to the demographic survey, we grouped the subjects such that both groups would have participants with fairly similar programming skills and experience with the Eclipse IDE. In this way, we moderated the threat of an unbalanced distribution of the subjects' expertise across the two groups. Third, we conducted a pilot study with two students that allowed us to calibrate the tasks' difficulty and estimate the time required to carry out each of them. Based on this information, we selected the five bugs and decided to give them two hours to find the target methods. Fourth, the participants were not informed about the study objectives, and they did not know which group they were before performing the study. Lastly, we limited the learning effect by presenting the five tasks in random order. Another factor that could affect the internal validity is the way the participants used the check button to verify if they found the target or buggy method since they could mark the methods randomly without inspecting and reviewing them and get the target or the buggy method by guessing. It could affect the results because the participants could have found the target method with less effort.

The external validity refers to the degree of generalization of the experiment's results across individuals, settings, and times. As in all controlled experiments, the degree of generalization is low since in real situations there would be a lack of control over all the manipulated factors. First of all, we have to admit that the experiment results may have been different if the subjects had been experienced professional developers. Thus, for comparison purposes, a replication of this experiment with professional developers is more than desirable. Secondly, although the selected tasks are real tasks extracted from the bug tracker system of the ADempiere project, we restricted the experiment to include only bug fixing tasks where only one method needs to be changed. This fact reduces the representativeness of our tasks, and therefore, it makes difficult to extrapolate the results to other types of concept location tasks.

Construct validity refers to the validity of the measures used to quantify the dependent variables of the experiment. With respect to effectiveness, we identified from the ADempiere source code repositories the method that was modified in order to fix each bug so that the solution corresponds exactly to what was done by the real developers of ADempiere. Regarding the effort, the Qualtrics features to automatically measure and save the time spent by each participant were used. Finally, to measure the efficiency related variables, we implemented the required functionality to to keep track of that kind of information.

Conclusion validity refers to the ability to draw the correct conclusions based on the data collected. Before making hypothesis testing we performed several statistical normality tests to check whether there is normality in the results. Since we found that the distribution is non-normal, we applied Mann-Whitney test[22], a well known non-parametric test. Then, we compared the results of the Tool and No-Tool groups who complete the tasks, i.e. the participants who found the target method of each bug.

6. Conclusions and Future Work

We present a controlled experiment for the empirical evaluation of an IR+RF approach for concept location. Moreover, we developed an Eclipse plugin that implements that approach and was a fundamental tool to achieve the goals of the experiment. In the experimental session, five bug fixing tasks were carried out by 40 students enrolled in a Software Engineering course. The subject system was ADempiere Business Suite, an industrial open-source software solution that combines ERP, CRM and SCM support for business process. For each bug fixing task we measured the time required by a developer to complete the task, its effectiveness, and the number of methods reviewed and marked, as well as the queries formulated by the developer when performing the task.

The IR+RF approach outperforms the Eclipse default searching feature because it only limits to extract exact matching of one identifier with or without using regular expressions into Java files. So to look for something the user has to know either the name or part of an identifier, While the IR+RF approach allows to group several terms and to use some operators that enhance query formulation.

Results indicate that the use of an IR+RF based tool has a positive impact when developers are dealing with concept location tasks such as bug localization. As far as performance measures are concerned, in order to answer the RQ1, we proved that in almost all tasks the Tool group outperformed No-Tool group in at least one of these measures.

Moreover, we found that the tool increases the effectiveness of solutions when developers carry out fault localization tasks. For almost all tasks, more participants in the Tool group successfully completed them than those who used Eclipse IDE default features, considering the size and little knowledge of the subject system.

Perhaps we did not have enough statistical evidence with other performance measures, so that we need to consider other experimental settings such as industry people, graduated students, open source developers, etc. in order to collect more evidence about the effect of the IR+RF approach.

As a future work, we plan to evaluate the impact of our IR+RF based tool with other concept location tasks, as in the controlled experiment we only used it for bugs' localization problems.

A. Appendix: Paper Analyzing the Impact of Information Retrieval and Relevance Feedback on Concept Location Tasks: A Controlled Experiment.

This paper was accepted and presented at the 12th Colombian Conference on Computing (12CCC).

Analyzing the Impact of Information Retrieval and Relevance Feedback on Concept Location Tasks: A Controlled Experiment.

Abstract. Concept location is the process by which a programmer determines the place, within a system codebase, where a change is to start in response to a modification request. It is a usual and fundamental process performed as part of software maintenance tasks such as bug fixing, refactoring, and in some cases, new feature implementation. One of the recent approaches proposed to support that process augments information retrieval (IR) based concept location via an explicit relevance feedback (RF) mechanism. This paper presents a tool that implements the IR+RF approach and a controlled experiment aimed at assessing the impact of that approach on bug fixing tasks. Within the experiment, five bug fixing tasks were performed by 40 undergraduate software engineering students. The efficiency of the participants, the completion time, and the correctness of their responses were measured. The results indicate that the IR+RF approach surpasses in correctness and efficiency the default searching functionalities provided by the Eclipse IDE. On the other hand, it does not reduce the completion time in bug fixing tasks.

Keywords: Concept Location,Information Retrieval, Relevance Feedback, Controlled Experiment

1 Introduction

Finding the part of a software system in which a problem domain concept is implemented is an essential task in software maintenance. For instance, this task appears when a developer needs to determine the source code location where a change is to start in response to a modification request. Nowadays *Concept Location* is defined as the process that identifies where a software system implements a specific concept.

Information retrieval methods have been used to address the concept location problem [4–9, 15, 17, 18]. Information retrieval (IR) is a discipline that deals with the retrieval of unstructured data, especially text documents. [2] defines IR as finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). The general goal of IR systems is to minimize the required time to trace the information that a user needs [3]. In the context of concept location, the IR method is used to map domain concepts expressed as developer queries to source code artifacts. Once the resulting list of artifacts is calculated by the IR method, the developer inspects and evaluates those results. This querying-inspecting cycle normally requires being repeated several times until the developer finds the desired software artifact.

On the other hand, relevance Feedback (RF) is an automatic approach that aims to improve query formulation within information retrieval processes. RF is a two-part process: first, the developer judges and assesses the relevance of existing results returned by a search. Then, the information retrieval system uses this feedback information to perform a new search, and (hopefully) return more relevant results to the developer [4]. The classic algorithm for implementing RF is the Rocchio algorithm [13]. *Rocchio* was developed using the Vector Space Model and it is based on the assumption that the user knows which documents should be denoted as relevant or non-relevant [2].

Gay et al [4] proposed an approach for concept location that combines an IR method and a RF algorithm. The novelty of that approach relies in the use of RF to enhance an IR method in two ways: (i) improving the developer's ability to write better queries throughout the entire concept location process, and (ii) allowing the developer to use the knowledge gained when inspecting a particular resulting list by incorporating it in the next query. With the aim of empirically evaluating that approach, we carried out a controlled experiment [14]. Thus, we collected evidence to determine whether the usage of an embedded information retrieval and relevance feedback tool increases the developers' performance when they have to carry out concept location tasks.

This paper presents the Eclipse plug-in that implements the IR+RF approach and the controlled experiment designed to compare the performance of the approach with the standard Eclipse searching functionalities.

2 Related Work

Several approaches to deal with concept location have been developed. These can be broadly classified into static, dynamic and combined analysis based approaches. In [16], a survey was carried out to discuss in detail the existing approaches. Marcus and Haiduc [17] present and discuss the application of several Text Retrieval techniques to support concept location. In [5] an information retrieval method called Latent Semantic Indexing (LSI) was applied to map concepts, expressed by developers in natural language, into relevant code fragments. Marcus et al. [6] used static concept location techniques in Object-Oriented code such as regular expression matching, static program dependences, or information retrieval. They compared these techniques to determine their strengths and weaknesses.

Poshyvanyk and Marcus [7] augmented an existing technique based on information retrieval for concept location with the automatic organization of the searching results using Formal Concept Analysis (FCA). Gay et al. [4] proposed an approach to augment IR-based concept location with an explicit relevance feedback mechanism. They performed af case of study with one developer to determine the impact of IR and Relevance Feedback (RF) on finding buggy methods, based on bug reports, on open source projects The main objective of this case of study focused on analysing the effect of RF in tasks as fault localization.

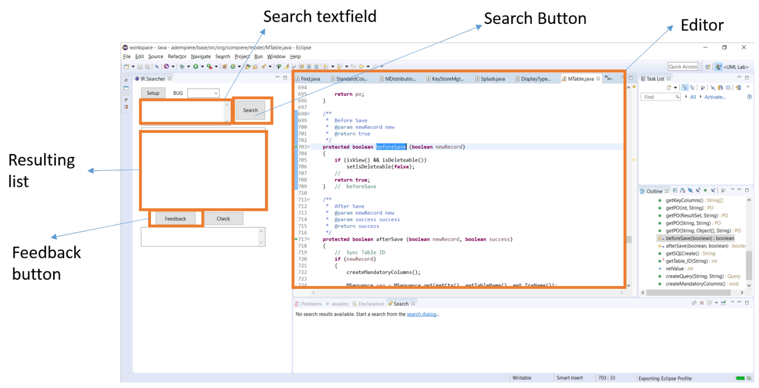
Later, Scanniello and Marcus [8] presented a new approach to static based concept location that combines structural and lexical information. They used an

IR-based approach which formulates concept location as a text retrieval problem. Poshyvanyk et al. [9] enhanced an existing IR-based technique for concept location with an automatic clustering of the search results through Formal Concept Analysis. In [15] a user study was conducted in order to determine how developers behave when using an Information Retrieval tool to perform fault localization tasks, based on bug reports. Scanniello and Marcus [18] implemented a concept location solution using the PageRank algorithm. They conducted a controlled experiment and replications to assess if their new technique helps the user to find the places where a change has to be made in the code.

A crucial activity for searching effectiveness during the concept location process is the formulation and improving of queries. Haiduc et al. [10–12] presented several approaches to assess and reformulate queries and improve their quality. These are useful in software engineering tasks, such as concept location or impact analysis, when developers are looking for source code artifacts. Text Retrieval (TR) techniques were used to evaluate the quality of queries and to predict and reformulate them in order to improve the results that allow developers to reach their goals when they are searching in a source code.

3 Eclipse Plug-in

We built an Eclipse plug-in that implements the IR+RF approach for concept location. The plug-in [23] allows the users to make queries and explore source code within the Java default perspective. We created a new view and called it IR Searcher. As shown in Figure 1, this view has a textfield to write queries; a button to perform searches; a resulting list of methods where users can examine and explore the results that satisfies a query. By clicking on one item of the result-ing list, users can see where and how it is implemented, in the Java editor of the Eclipse IDE. The users can select the methods they considered as relevant and use the feedback mechanism to reformulate the query.



```
        return po;
    }

    /**
     *  Before Save
     *  @param newRecord new
     *  @return true
     */
    protected boolean beforeSave (boolean newRecord)
    {
        if (isView() && isDeleteable())
            setIsDeleteable(false);
        //
        return true;
    }   //  beforeSave

    /**
     *  After Save
     *  @param newRecord new
     *  @param success success
     *  @return success
     */
    protected boolean afterSave (boolean newRecord, boolean success)
    {
        //  Sync Table ID
        if (newRecord)
        {
            createMandatoryColumns();

            MSequence seq = MSequence.get(getCtx(), getTableName(), get_TrxName());
```

No search results available. Start a search from the search dialog...

Fig. 1. Eclipse plug-in GUI

The user can select a Java project to index and a folder to store the index (mathematical representation of Java code after applying IR method) with the setup button. Besides, it allows either to create new index with the selected project or add it to an existing index.

4 The IR+RF Approach for Concept Location

The tool followed most of the IR based concept location principles, established in [4]:

1. ***Corpus creation:*** The code was parsed using methods as granularity level. The user should choose the root folder of the project that he wants to look for. We use a Abstract Syntax Tree (AST) to visit all the Java methods in the project. The documents indexed are the methods visited with the AST. We process the text of the header and body of the methods and apply text retrieval techniques like stop words removal, Java stop words (while, for, if, new, etc.) removal, identifier splitting (we kept original identifiers), stemming, etc. The user can select another project and either add to existing index or create a new one.
2. ***Corpus indexing:*** We use Apache Lucene for indexing the corpus [19]. It uses the Vector Space Model (VSM) algorithm to store a mathematical representation of the corpus. Each method has its representation in the index. In VSM each document has a vector model as a vector of weights that span over the words of corpus [4]. VSM was chosen because RF techniques were developed specifically for this IR algorithm.
3. ***Query formulation:*** The developer uses a set of words within specific syntax rules defined by Lucene [20] to formulate queries. Lucene parses the query and returns the documents, i.e., Java methods that match the query. Queries receive the same text retrieval pre-processing treatment as the methods in corpus creation.
4. ***Ranking documents:*** Lucene sorts documents retrieved by a query according its rank. The rank is assigned according to how closely each result match the query [19].
5. ***Results examination:*** The developer can review a ranked list of Java methods that satisfy a query. The programmer can visit the source code by clicking on the method that wants to investigate in the resulting list.
6. ***Relevance feedback:*** The tool uses an explicit feedback mechanism. That means that the developer has to select the methods he considered relevant, and based on that information, the query is rebuilt. To do that, the tool considers the original terms and the terms of documents considered relevant, and removes the terms of documents considered as not relevant. We use a modified version of original Rocchio Relevance feedback algorithm introduced in [4].

5 Experimental Design

The aim of the experiment is to provide empirical evidence of the efficiency and effectiveness of the IR+RF approach when compared to standard searching features provided by the Eclipse IDE.

5.1 Research Questions

We formulated the following research questions:
RQ1: Does the IR+RF approach reduce the time needed by developers and improve performance of them when they perform concept location tasks, compared to default Eclipse IDE searching features?
RQ2: Does the IR+RF approach improve the correctness of the solutions for concept location tasks, compared to default Eclipse IDE searching features?

The main factor being analyzed is the use of the IR+RF approach implemented as an Eclipse plugin. A *between subjects* design was utilized, where the subjects were divided into two disjointed groups, one that used the Eclipse plug-in, and the other one used the standard Eclipse features. To assess the impact of the IR+RF approach we measure three dependent variables: time, efficiency and correctness.

5.2 Hypotheses

Based on the two research questions, we formulated the following hypotheses:

Null hypothesis	**Alternative hypothesis**
$H1_0$: There is no significant difference in time to complete between the IR+RF and Eclipse groups for concept location tasks	$H1$: There is significant difference in time to complete between the IR+RF and Eclipse groups for concept location tasks.
$H2_0$: There is no significant difference in the number of revised methods between the IR+RF and Eclipse groups for concept location tasks	$H2$: There is significant difference in the number of revised methods between the IR+RF and Eclipse groups for concept location tasks
$H3_0$: There is no significant difference in the number of marked methods between the IR+RF and Eclipse groups for concept location tasks	$H3$: There is significant difference in the number of marked methods between the IR+RF and Eclipse groups for concept location tasks
$H4_0$: There is no significant difference in the number of formulated queries between the IR+RF and Eclipse groups for concept location tasks	$H4$: There is significant difference in the number of formulated queries between the IR+RF and Eclipse groups for concept location tasks

Table 1. Hypotheses formulation

5.3 Dependent and Independent Variables

The experiment has only one independent variable: the tool used to solve the concept location tasks. Thus, the tool variable has two treatments, i.e., the IR+RF approach (Tool) and a baseline which is the default features provided by the Eclipse IDE (No Tool). To measure the effect of using the IR+RF approach, implemented as an Eclipse plugin, to support concept location tasks against the default Eclipse IDE features, we consider three dependent variables: effort, correctness, and efficiency.
Effort (completion time) refers to the time required by a developer to complete a concept location task e.g. localizing a fault. *Correctness* is a binary variable that indicates if the user successfully completed the concept location task. *Efficiency* is the number of methods reviewed and marked, and the queries formulated by the developer when performing the concept location task.

5.4 Control Variables

We use the experience level and Java knowledge as control variables, since it allows us to divide the participants into two balanced groups and assign them to the treatments (Tool and No tool). Thus, before performing the controlled experiment we examined the experience level of participants through a demographic survey and divide them according to it.

5.5 Subject System

We chose ADempiere[1]Business Suite, an industrial open-source software solution that combines ERP, CRM and SCM support for business process. This system is a largue size project with a well documented bug report system and an active community. For the controlled experiment we considered the version 3.8 launched on March 1th 2015, which has approximately 11 millions LOC, 68% of which are written in Java, 5.234 classes, 376 packages, and 89.335 methods.

5.6 Participants

The participants were 40 undergraduate students enrolled in a software engineering course. We divided them into two groups according their knowledge and experience with the Java programming language, Eclipse IDE, and Eclipse searching features. We did maintain a balance between the groups with respect that expertise as much as possible. One group, named as henceforth the Tool group, used the Eclipse plug-in that implements the IR+RF approach for concept location. The other group, that we named the No-Tool group, used the default features of Eclipse for searching.

[1] `www.adempiere.net`

5.7 Tasks

Bug	Title	Description	Target Method
1	Process Class Generator not get parameters type correctly.	When a parameter is of type FilePathOrName, its created like Object on getter method on Process abstract generated.	Class: DisplayType Method: boolean isText(int displayType)
2	DRP not generate distribution order because business partner not link organization.	DRP not generate order , the issue that business partner are try find is based on source locator organization. The system should use of organization from target locator.	Class : MRP Method: void createDDOrder(int AD_Org_ID, int PP_MRP_ID, MProduct product,BigDecimal QtyPlanned ,Timestamp DemandDateStartSchedule, String trxName)
3	The tables mark like IsDocument is deleteable.	Currently the tables mark like IsDocument are deleteables, it is bad for the document, all document that handle document number must be no deleteable.	Class: MTable Method: boolean beforeSave (boolean newRecord).
4	Error when try merge product entity.	When try merge from a duplicate product , the error throw exception because not can delete product because have a dependence with cost dimension.	Class: Merge Methdod: int mergeTable (String TableName, String ColumnName, int from_ID, int to_ID)
5	Get all employees for payroll process, should get all active or not active.		Class: MHREmployee Method:MBPartner[] getEmployees (MHRProcess process)

Table 2. Relevant information of bugs

We chose five bugs from the ADempiere issue tracker system[2]and reviewed approved patches for them. By contrasting the correction reported in further versions for each bug, we determined the method that was modified in order to fix the bug. This approach was used in [4]. Table 2 shows the title, description, and buggy or target method for each bug. We chose bugs where only one target method was modified in order to fix it. We kept titles and descriptions as appear in ADempiere bug tracker. There are poor quality explanations and spelling and grammatical errors in some titles and bug descriptions. We decided not to fix them and use original text so that participants dealt with some of the actual problems of software engineering industry.

5.8 Procedure

Two days before conducting the experiment, the participants attended a tutorial session where we presented the ADempiere system, the plug-in features (usage and query formulation syntax), and the Eclipse search features. Then, they responded the demographical survey, which allowed us to divide the participants in two groups as homogeneously as possible based on their previous experience with Java and Eclipse.

In the experimental session, once the subjects were divided in the two experimental groups, they received the title and description of five bugs, which were shown randomly in a Qualtrics[3]survey. We measured the time spent by each participant since the bug was shown until she finished the task. Depending on the group, the participants use either the tool or the Eclipse IDE default search features to find the target method that fix the bug.

For the No-Tool group we modified Eclipse so as to track the results of each Java search and allow subjects to select and explore the methods in the resulting list. Furthermore, we stored how many methods were revised and marked by participants, how many queries or searches were made, and we also monitored the results so that we know when the target method of a specific bug was found. We did the same in the Tool group to keep track of the actions of both groups.

The experimental session was limited to two hours. The following instructions were given to the participants at the beginning of session:

- In this experiment you will find the description of five bugs. For each bug, you have to find what method needs to be modified in order to fix the bug. That method is called the Target Method. There is only one target method for each bug.
- In order to make the initial query, you can use terms from the bug title or description as keywords.
- You are able to formulate the queries and freely revise the resulting list of methods. Once you have obtained the results of a search, you have to mark the methods that you believe have to be modified in order to fix the bug, and then, click the check button in order to verify whether the target method has been found. You can mark as relevant maximum 30 methods in total.
- If you have not found the target method (after using the check button), you have three options: i) use the feedback (this option is only available for the Tool group), ii) review and select other methods, or iii) reformulate the query to start a new search.

[2] `https://github.com/adempiere/adempiere/issues/`
[3] `www.qualtrics.com`

- You can continue until either you have marked 30 methods or the target method is found. In the last case, the task was successfully completed for the bug at hand.
- Once a task is completed, Eclipse shows relevant information such as correctness, revised and marked methods, and formulated queries. You must copy this information and paste it in the Qualtrics survey.

Once the five tasks were completed a post-experiment questionnaire was used to collect participants' thoughts about relevance feedback as a code searching support, and their opinions about the experiment itself.

6 Results Analysis and Discussion

First of all, in order to analyze the results, we see how many participants (not) found the target method for each bug. Some participants did not have enough time to complete all tasks, so that some bugs are considered as unexplored (UN).

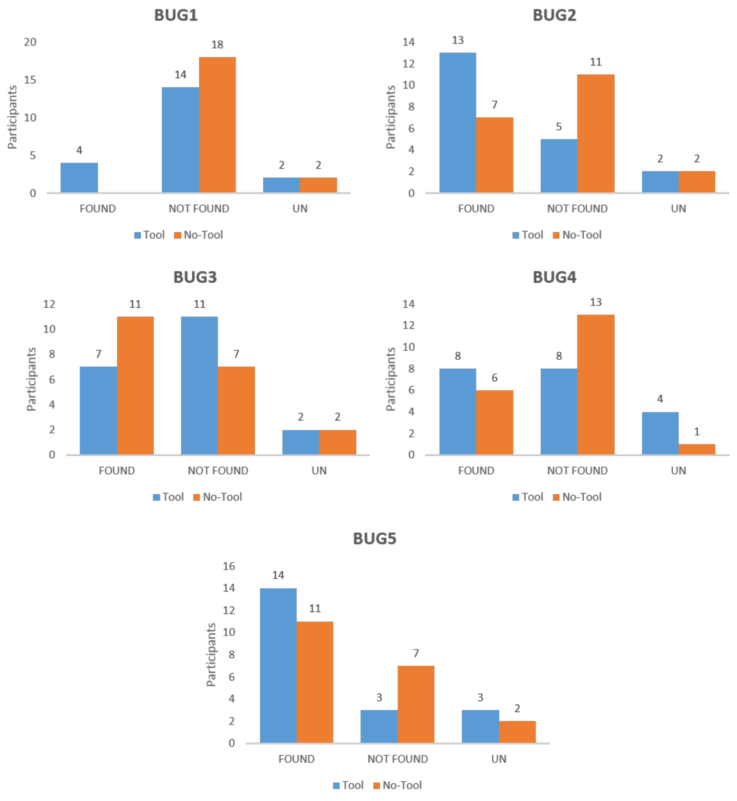


Fig. 2. Correctness of tasks by participants

As shown in Figure 2, for almost all bugs the participants in the Tool group had better performance than those in the No-Tool group, in terms of target methods found. In 4 of the 5 tasks the participants with the tool found more bugs than the ones who used default Eclipse IDE features. For bug1 no one in the No-Tool group found the target method. In average the Tool group found 2.3 bugs while the No-Tool group participants found 1.75. Bug 3 was an exceptional case where the No-Tool group outperformed the Tool group. We think some terms in the title or bug description may negatively influence the performance of the Tool group, since participants took keywords from there to write queries.

Secondly, we made an histogram for the number of target methods found by participants of each group. Figure 3 shows that subjects in the Tool group found at least one target method, while in the No-Tool group three subjects did not find any buggy method. Furthermore, the maximum number of target methods found by the No-Tool group was 3, while 4 and 5 target methods were found by some of the participants in the Tool group.

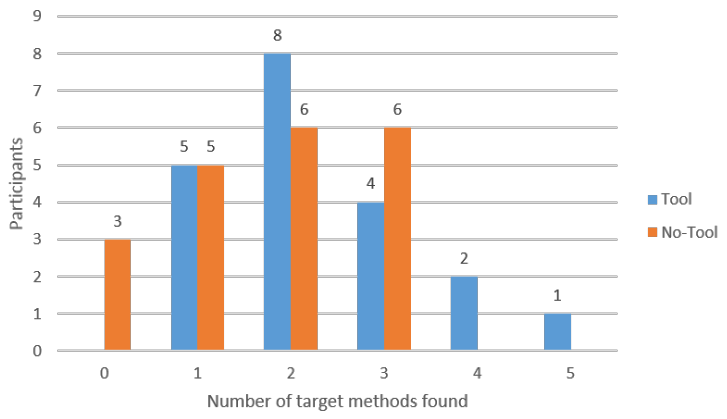


Fig. 3. Number of participants that found target methods by group

Before making hypothesis testing we performed several statistical normality tests to check whether there is normality in the results. We applied five tests available in ‘nortest’ package[21] of R[4]. For each numerical variable collected for each bug (time spent, marked methods, revised methods and queries formulated) we performed Anderson-Darling Cramer-von Mises, Lilliefors (Kolmogorov-Smirnov), Pearson chi-square, and Shapiro-Francia tests for normality and found that the $p-values$ of each test are lower than 0.05. Thus, none of variables have normal behavior.

[4] `https://www.r-project.org/`

Therefore, we applied Mann-Whitney test[22], a well known non-parametric test. We compared the results of the Tool and No-Tool groups who completed the tasks, i.e. the participants who found the target method of each bug. First, we perform the hypothesis tests that we stated in Table 1. Since for bug 1 none of the No-Tool participants completed the task, we concluded that in bug 1 the Tool group outperformed the No-Tool group. Table 3 shows that there are meaningful differences in highlighted performance variables for each bug, since $p-values$ are less than 0.05, and therefore, the null hypothesis must be rejected.

Bug	**Time Spent**	**Revised Methods**	**Marked Methods**	**Formulated Queries**
2	0.114600	0.008218	0.031100	0.000820
3	0.791400	0.525900	0.617500	0.049790
4	0.413600	0.365600	0.016680	0.516300
5	0.291500	0.323800	0.271800	0.009190

Table 3. $p - values$ of Mann-Whitney hypothesis tests (Tool vs. No-Tool)

We need to know what group has better performance, so we performed an hypothesis test where Ho : the mean of performance variables of No-Tool group is less or equal than the mean of performance variables of Tool group, and $H1$: the mean of the performance variables of No-Tool group is greater than the mean of the performance variables of Tool group. We only did this hypothesis test on performance variables where previous tests show meaningful differences.

Bug	**Time Spent**	**Revised Methods**	**Marked Methods**	**Formulated Queries**
2		0.004109	0.015550	0.000410
3				0.025400
4			0.994200	
5				0.004595

Table 4. $p - values$ of Mann-Whitney hypothesis tests (Tool vs. No-Tool)

Table 4 shows that highlighted $p - values$ are lower than 0.05, so that the null hypothesis should be rejected. It means that with exception of bug 4, for the number of formulated queries by bug, the Tool group outperformed the No-Tool group, i.e., the members of the Tool group had to formulate less queries than the participants of the No-Tool group to successfully complete the tasks. Regarding the number of revised and marked methods, only in bug 2 the subjects of the Tool group had to revise and mark less methods than the No-Tool group members to complete this task. With regards to time spent, we did not find statistical evidence of a significant difference between the Tool group and No-

Tool Group. On the other hand, the No-Tool group participants outperformed the Tool group in bug4, since they had to marked less methods for completing the task, that is to say, it was the only one performance measure where there is statistical evidence that the No-Tool group surpassed the Tool group.

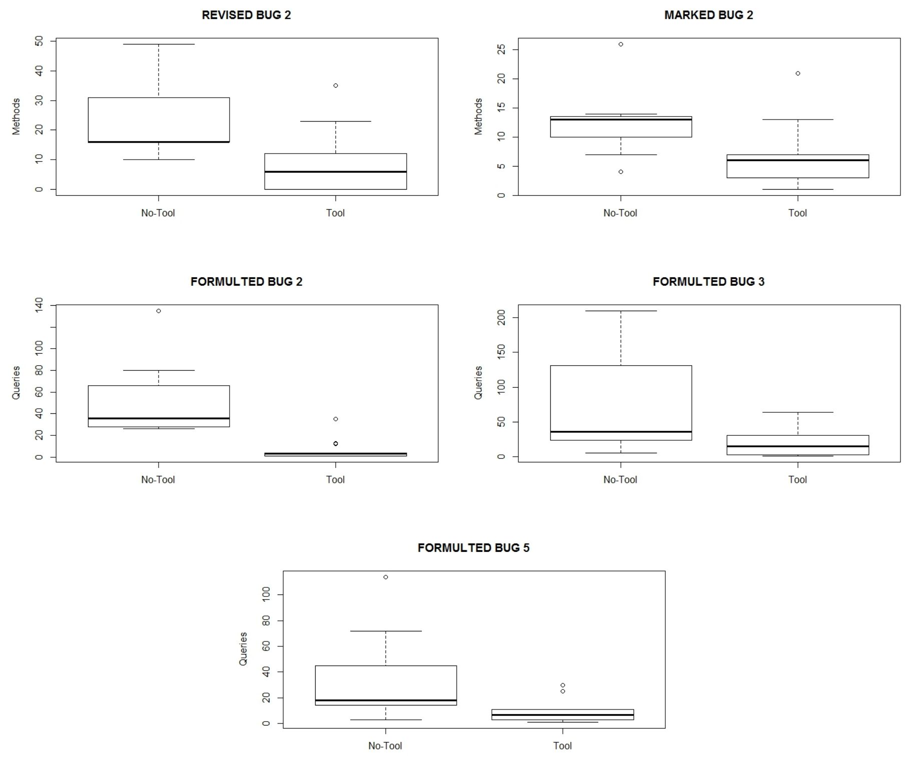


Fig. 4. Boxplots of performance measures where there was meaningful difference.

Figure 4 shows graphically the performance variables where hypothesis test showed meaningful difference and those where subjects of the Tool group outperformed the No-Tool group, when they successfully completed the tasks.

7 Threats to Validity

In this section we discuss the threats to the validity of the experiment results. The internal validity refers to unexpected factors that may compete with our independent variable and affect the values obtained for the dependent variables. First of all, to reduce the threat that the participants may not be competent to perform the tasks, we conduct a tutorial session where they learned how to use the Eclipse plug-in for concept location purposes. Moreover, we familiarized them with the main features and modules of the ADempiere Business Suite, and reviewed the main search features offered by the Eclipse IDE. Second, based on the answers to the demographic survey, we grouped the subjects such that both groups would have participants with fairly similar programming skills and experience with the Eclipse IDE. In this way we moderated the threat of an unbalanced distribution of the subjects' expertise across the two groups. Third, we conducted a pilot study with two students that allowed us to calibrate the tasks' difficulty and estimate the time required to carry out each of them. Based on this information, we selected the five bugs and decided to give them two hours to find the target methods. Fourth, the participants were not informed about the study objectives, and they did not know which group they were before performing the study. Lastly, we limited the learning effect by presenting the five tasks in random order.

The external validity refers to the degree of generalization of the experiment's results across individuals, settings, and times. As in all controlled experiments, the degree of generalization is low since in real situations there would be a lack of control over all the manipulated factors. First of all, we have to admit that the experiment results may had been different if the subjects had been experienced professional developers. Thus, for comparison purposes, a replication of this experiment with professional developers is more than desirable. Secondly, although the selected tasks are real tasks extracted from the bug tracker system of the ADempiere project, we restricted the experiment to include only bug fixing tasks where only one method needs to be changed. This fact reduces the representativeness of our tasks, and therefore, it makes difficult to extrapolate the results to other types of concept location tasks.

Construct validity refers to the validity of the measures used to quantify the dependent variables of the experiment. With respect to correctness, we identified from the ADempiere source code repositories the method that was modified in order to fix each bug so that the solution corresponds exactly to what was done by the real developers of ADempiere. Regarding the effort, the Qualtrics features to automatically measure and save the time spent by each participant were used. Finally, to measure the efficiency related variables, we implemented the required functionality to to keep track of that kind of information.

Conclusion validity refers to the ability to draw the correct conclusions based on the data collected. Before making hypothesis testing we performed several statistical normality tests to check whether there is normality in the results. Since we found that the distribution is non-normal, we applied Mann-Whitney test [22], a well known non-parametric test. Then, we compared the results of the Tool and No-Tool groups who complete the tasks, i.e. the participants who found the target method of each bug.

8 Conclusions and Future Work

We presented a controlled experiment for the empirical evaluation of an IR+RF approach for concept location. Moreover, we described an Eclipse plug-in that implements that approach and was a fundamental tool to achieve the goals of the experiment. In the experimental session, five bug fixing tasks were carried out by 40 students enrolled in a Software Engineering course. The subject system was ADempiere Business Suite, an industrial open-source software solution that combines ERP, CRM and SCM support for business process. For each bug fixing task we measured the time required by a developer to complete the task, its correctness, and the number of methods reviewed and marked, as well as the queries formulated by the developer when performing the task.

Results indicate that the use of an IR+RF based tool has a positive impact when developers are dealing with concept location tasks such as bug localization. As far as performance measures are concerned, in order to answer the RQ1, we proved that in almost all tasks the Tool group outperformed No-Tool group in at least one of these measures.

Moreover, we found that the tool increases the correctness of solutions when developers carry out fault localization tasks. For almost all tasks, more participants in the Tool group successfully completed them than those who used Eclipse IDE default features, considering the size and little knowledge of system.

Perhaps we did not have enough statistical evidence with other performance measures, so that we need to consider other experimental settings such as industry people, graduated students, open source developers, etc. in order to collect more evidence about the effect of the IR+RF approach.

As a future work we plan to evaluate the impact of our IR+RF based tool with other concept location tasks, as in this experiment we only used it for bugs' localization problems.

References

1. T. J. Biggerstaff, B. G. Mitbander, and D. Webster, "The concept assignment problem in program understanding," Proc. 1993 15th Int. Conf. Softw. Eng., 1993.
2. C. D. Manning, P. Raghavan, and H. Schtze, An Introduction to Information Retrieval. Cambridge University Press, 2009.
3. G. Kowalski, Information retrieval architecture and algorithms. Springer, 2011.
4. G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in IR-based concept location," IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 351360, 2009.
5. A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, "An information retrieval approach to concept location in source code," Reverse Eng. 2004. Proceedings. 11th Work. Conf., pp. 214223, 2004.
6. A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, "Static techniques for concept location in object-oriented code,"13th Int. Work. Progr. Compr., 2005.
7. D. Poshyvanyk and A. Marcus, "Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code," 15th IEEE Int. Conf. Progr. Compr. (ICPC 07), 2007.

8. G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," IEEE Int. Conf. Progr. Compr., pp. 110, 2011.
9. D. Poshyvanyk, M. Gethers, and A. Marcus, "Concept location using formal concept analysis and information retrieval," Trans. Softw. Eng. Methodol. (TOSEM), vol. 21, no. 4, 2012.
10. S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Automatic Query Performance Assessment During the Retrieval of Software Artifacts," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 9099.
11. S. Haiduc, G. De Rosa, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Query Quality Prediction and Reformulation for Source Code Search: The Refoqus Tool," in Proceedings of the 2013 International Conference on Software Engineering, 2013, pp. 13071310.
12. S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic Query Reformulations for Text Retrieval in Software Engineering," in Proceedings of the 2013 International Conference on Software Engineering, 2013, pp. 842851.
13. J. J. Rocchio, "Relevance Feedback in Information Retrieval," 1971.
14. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wessln, Experimentation in Software Engineering. Springer, 2012.
15. Q. Wang, C. Parnin, and A. Orso, "Evaluating the Usefulness of IR-based Fault Localization Techniques," in Proceedings of the 2015 International Symposium on Software Testing and Analysis, 2015, pp. 111.
16. B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. "Feature location in source code: a taxonomy and survey." J. Softw. Maint. Evol.: Res. Pract., 2011.
17. A. Marcus and S. Haiduc, "Text Retrieval Approaches for Concept Location in Source Code," in Software Engineering SE - 5, vol. 7171, A. De Lucia and F. Ferrucci, Eds. Springer Berlin Heidelberg, 2013, pp. 126158.
18. G. Scanniello, A. Marcus, and D. Pascale, "Link analysis algorithms for static concept location: an empirical assessment,"in Empirical Software Engineering,vol. 20, no. 6. 2014, pp. 1666-1720.
19. M. McCandless, E. Hatcher, Gospodnetic Otis, and D. Cutting, Lucene in action. Greenwich: Manning, 2010.
20. "Query Parser Syntax," Apache Lucene,`https://lucene.apache.org/core/3_0_3/queryparsersyntax.html`.
21. "Package 'nortest'," R project, `https://cran.r-project.org/web/packages/nortest/nortest.pdf`
22. M. Hollander and D. A. Wolfe, Nonparametric statistical methods, vol. 2. New York: John Wiley & Sons, pp 68-75, 1999.
23. E. Clayberg and D. Rubel, Eclipse Plug-ins. 2008.

B. Appendix: Qualtrics Questionnaires of Demographical, Controlled Experiment and Post-Experiment Surveys

In this appendix, we attach the questionnaires of the demographical, the controlled experiment and the post-experiment surveys from Qualtrics.

Default Question Block

¿Cuál es su nombre?

Por favor evalue su conocimiento del lenaguaje de programación Java.

- ○ Muy bueno
- ○ Bueno
- ○ Satisfactorio
- ○ Pobre
- ○ Muy pobre

Por favor indique cuántos años ha estado programado en Java?

- ○ Menos de 1 año.
- ○ De 1 a 3 años
- ○ De 3 a 5 años
- ○ Más de 5 años

¿Está actualmente trabajando como desarrollador en alguna empresa?

- ○ Si
- ○ No

¿TIene experiencia usando el IDE Eclipse?

- ○ SI

- NO

¿Cuánto tiempo ha usado el IDE Eclipse?

- Menos de 1 año.
- De 1 a 3 años.
- De 3 a 5 años
- Más de 5 años

¿Ha usado las funcionalidades de búsqueda de Eclipse?

- SI
- NO

Por favor evalue su destreza usando las funcionalidasdes de búsqueda de Eclipse

- Muy buena
- Buena
- Satisfactoria
- Pobre
- Muy pobre

Default Question Block

Instructions. [Please read carefully and clarify any doubt you may have]

In this experiment you will find the description of five bugs. For each bug, you have to find what method needs to be modified in order to fix the bug. That method is called the *Taget Method*. There is only one target method for each bug.

Be sure to choose the bug for which you are looking for the solution in the combobox.

In order to make the initial query, you can use terms from the bug title or description as keywords.

Once you have obtained the results of a search, you have to mark the methods you consider relevant, and then, click the *check* button in order to verify whether the target method has been found. You can mark as relevant maximum 30 methods in total.

If you have not found the target method (after using the check button), you have three options: i) use the feedback (if it is available), ii) review and select other methods, or iii) reformulate the query to start a new search.

When the target method is found or you have marked 30 methods you will be automatically notified.

At the end of the search process, please copy the result from the textbox (in Eclipse) and paste that text in the box below the description of the bug (in this survey). Then, continue to the next bug.

Thanks for participating.

Please introduce your participant ID

B1

Bug 1

Title:

Process Class Generator not get parameters type correctly

Despcription:

When a parameter is of type FilePathOrName, it s created like Object on getter method on Process abstract generated

These page timer metrics will not be displayed to the recipient.

First Click: *0 seconds*

Last Click: *0 seconds*

Page Submit: *0 seconds*

Click Count: *0 clicks*

B2

Bug 2

Title:

DRP not generate distribution order because business partner not link organization.

Despcription:

DRP not generate order , the issue that business partner are try find is based on source locator organization.**The system should use of organization from target locator**

These page timer metrics will not be displayed to the recipient.

First Click: *0 seconds*

Last Click: *0 seconds*

Page Submit: *0 seconds*
Click Count: *0 clicks*

B3

Bug 3

Title:

The tables mark like IsDocument is deleteable

Despcription:

Currently the tables mark like IsDocument are deleteables, it is bad for the document, all document that handle document number must be no deleteable

These page timer metrics will not be displayed to the recipient.
First Click: *0 seconds*
Last Click: *0 seconds*
Page Submit: *0 seconds*
Click Count: *0 clicks*

B4

Bug 4

Title:

Error when try merge product entity

Despcription:

When try merge from a duplicate product , the error throw exception because not can delete product because have a dependence with cost dimension.

These page timer metrics will not be displayed to the recipient.
First Click: *0 seconds*
Last Click: *0 seconds*
Page Submit: *0 seconds*
Click Count: *0 clicks*

B5

Bug 5

Title:

Get all employees for payroll process, should get all active or not active

These page timer metrics will not be displayed to the recipient.
First Click: *0 seconds*
Last Click: *0 seconds*
Page Submit: *0 seconds*
Click Count: *0 clicks*

PostExperiment

¿Qué tan complicado le resultó buscar los *target methods* o *buggy methods*?

Muy difícil

Difícil

Moderado

Fácil

Muy facil

¿Qué tan provechoso es usar la funcionalidad de feedback en el proceso de buscar los target methods o buggy methods?

Muy útil

Útil

Moderadamente útil

Poco útil

Inútil

Por favor describa cualquier dificultad que haya tenido en el desarrollo del ejercicio (por ejemplo, problemas con la conexión a internet)

Si tiene comentarios o sugerencias sobre este experimento, por favor repórtelos en este espacio

Report Abuse

References

[1] T. J. Biggerstaff, B. G. Mitbander, and D. Webster, “The concept assignment problem in program understanding,” Proc. 1993 15th Int. Conf. Softw. Eng., 1993.

[2] C. D. Manning, P. Raghavan, and H. Schütze, An Introduction to Information Retrieval. Cambridge University Press, 2009.

[3] G. Kowalski, Information retrieval architecture and algorithms. Springer, 2011.

[4] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, “On the use of relevance feedback in IR-based concept location,” IEEE Int. Conf. Softw. Maintenance, ICSM, pp. 351–360, 2009.

[5] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic, “An information retrieval approach to concept location in source code,” Reverse Eng. 2004. Proceedings. 11th Work. Conf., pp. 214–223, 2004.

[6] A. Marcus, V. Rajlich, J. Buchta, M. Petrenko, and A. Sergeyev, “Static techniques for concept location in object-oriented code,”13th Int. Work. Progr. Compr., 2005.

[7] D. Poshyvanyk and A. Marcus, “Combining Formal Concept Analysis with Information Retrieval for Concept Location in Source Code,” 15th IEEE Int. Conf. Progr. Compr. (ICPC ’07), 2007.

[8] G. Scanniello and A. Marcus, “Clustering support for static concept location in source code,” IEEE Int. Conf. Progr. Compr., pp. 1–10, 2011.

[9] D. Poshyvanyk, M. Gethers, and A. Marcus, “Concept location using formal concept analysis and information retrieval,” Trans. Softw. Eng. Methodol. (TOSEM), vol. 21, no. 4, 2012.

[10] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, “Automatic Query Performance Assessment During the Retrieval of Software Artifacts,” in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, 2012, pp. 90–99.

[11] S. Haiduc, G. De Rosa, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, “Query Quality Prediction and Reformulation for Source Code Search: The Refoqus Tool,” in Proceedings of the 2013 International Conference on Software Engineering, 2013, pp. 1307–1310.

[12] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, “Automatic Query Reformulations for Text Retrieval in Software Engineering,” in Proceedings of the 2013 International Conference on Software Engineering, 2013, pp. 842–851.

[13] J. J. Rocchio, “Relevance Feedback in Information Retrieval,” 1971.

[14] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, Experimentation in Software Engineering. Springer, 2012.

[15] Q. Wang, C. Parnin, and A. Orso, “Evaluating the Usefulness of IR-based Fault Localization Techniques,” in Proceedings of the 2015 International Symposium on Software Testing and Analysis, 2015, pp. 1–11.

[16] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. “Feature location in source code: a taxonomy and survey.” J. Softw. Maint. Evol.: Res. Pract., 2011.

[17] A. Marcus and S. Haiduc, “Text Retrieval Approaches for Concept Location in Source Code,” in Software Engineering SE - 5, vol. 7171, A. De Lucia and F. Ferrucci, Eds. Springer Berlin Heidelberg, 2013, pp. 126–158.

[18] G. Scanniello, A. Marcus, and D. Pascale, “Link analysis algorithms for static concept location: an empirical assessment,” in Empirical Software Engineering,vol. 20, no. 6. 2014, pp. 1666-1720.

[19] M. McCandless, E. Hatcher, Gospodnetić Otis, and D. Cutting, Lucene in action. Greenwich: Manning, 2010.

[20] “Query Parser Syntax,” Apache Lucene, Available: `https://lucene.apache.org/core/3_0_3/queryparsersyntax.html`.

[21] “Package ‘nortest’,” R project, Available: `https://cran.r-project.org/web/packages/nortest/nortest.pdf`

[22] M. Hollander and D. A. Wolfe, Nonparametric statistical methods, vol. 2. New York: John Wiley & Sons, pp 68-75, 1999.

[23] E. Clayberg and D. Rubel, Eclipse Plug-ins. 2008.

[24] C. Mills, G. Bavota, S. Haiduc, R. Oliveto, A. Marcus, and A. D. Lucia, “Predicting Query Quality for Applications of Text Retrieval to Software Engineering Tasks,” Transactions on Software Engineering and Methodology, vol. 26, no. 1, p. 3:1–3:45, May 2017.

[25] O.Chaparro , J. M. Florez, A. Marcus, “Using Observed Behavior to Reformulate Queries during Text Retrieval-based Bug Localization,” Proceedings of the 33rd IEEE International Conference on Software Maintenance and Evolution ICSME’17, Shanghai, China (to appear)

[26] B. Sisman and A. C. Kak, “Assisting code search with automatic query reformulation for bug localization,” in Proceedings of the Working Conference on Mining Software Repositories (MSR’13), 2013, pp. 309– 318.