Query Quality Prediction and Reformulation for Source Code Search: The Refoqus Tool

Sonia Haiduc[1], Giuseppe De Rosa[2], Gabriele Bavota[2], Rocco Oliveto[3], Andrea de Lucia[2], Andrian Marcus[1]
[1]Department of Computer Science, Wayne State University, Detroit, MI 48202, USA
[2]School of Science, University of Salerno, 84084 Fisciano (SA), Italy
[3]Department of Bioscience and Territory, University of Molise, 86090 Pesche (IS), Italy

***Abstract*—Developers search source code frequently during their daily tasks, to find pieces of code to reuse, to find where to implement changes, etc. Code search based on text retrieval (TR) techniques has been widely used in the software engineering community during the past decade. The accuracy of the TR-based search results depends largely on the quality of the query used. We introduce *Refoqus*, an Eclipse plugin which is able to automatically detect the quality of a text retrieval query and to propose reformulations for it, when needed, in order to improve the results of TR-based code search. A video of Refoqus is found online at http://www.youtube.com/watch?v=UQlWGiauyk4.**

***Index Terms*— Source Code Search; Text Retrieval; Query Quality; Query Reformulation**

I. INTRODUCTION

Source code search is one of the most frequently performed actions by developers during software maintenance and evolution [8]. Over the last decade, text retrieval (TR) has emerged as one class of extremely effective techniques for this purpose and code search based on TR has been used in the context of numerous software engineering (SE) tasks (e.g., program comprehension [10] , code reuse [9], etc.).

One problem common to all TR-based approaches is that the text query used and its relationship to the text contained in the software artifacts greatly influences the search results. Writing good queries is not an easy task as it requires intimate knowledge of the source code vocabulary and its use, which is difficult to get even in small projects, let alone in large projects with millions of lines of code. A poor query not only fails to return relevant results, but it also leads to wasted time and effort on behalf of the developers, who need to investigate the irrelevant search results before realizing that the query was a poor choice. The challenge is to *provide immediate feedback about quality of the query.*

When the search results are not relevant to the task at hand (i.e., the query is poor), the query needs to be reformulated (i.e., words added or deleted from the query) such that better results are retrieved (i.e., relevant documents are retrieved closer to the top of the result list). Query reformulation is often very difficult for developers, given that they did not know how to write a good query in the first place. Researchers in the SE field have addressed this problem by proposing the use of two types of query reformulation approaches. The first is based on user relevance feedback [3, 4], which is an interactive approach that relies on the developer to analyze the list of results and mark the top documents as relevant or not relevant. The documents marked by the user are then used to reformulate the query. One disadvantage of this technique is that it requires significant developer effort for analyzing and marking documents that are not directly relevant for the task at hand. A second class of approaches is based on automatically adding terms to the query, which are similar to the terms found in the query (e.g., synonyms) [5, 11].

The main shortcoming of both categories of approaches is that they apply the same reformulation strategy to all queries, indiscriminately. The performance of a query depends on many factors and queries with different properties may need different reformulation strategies. The challenge is to *recommend the best reformulation technique for each query, based on its individual properties.*

We present *Refoqus* (**REF**ormulation **O**f **QU**erie**S**), a novel tool which addresses these challenges and: (1) automatically *predicts the quality of a query* and (2) selects *the best reformulation strategy* for low quality queries and *recommends a replacement query* to the developer based on this strategy.

Refoqus is meant to be used by developers during their daily tasks, whenever searching is needed. Fig. 1 shows the typical interaction between a developer and Refoqus and Fig. 2 shows the main window of Refoqus in Eclipse. At first sight, Refoqus works like any other TR-based tool for source code search, i.e., the developer writes and runs a text query and Refoqus returns a list of ranked methods relevant to the query. From this point on, however, Refoqus is unique. While retrieving the relevant results, Refoqus also analyzes the quality of the submitted query and classifies it as high or low quality. The developer can use this feedback as a guideline for determining if the results returned by the query are worth investigating or not. In the case when Refoqus indicates the query is of high quality, the developer is likely to find the relevant code among the top results and the search can end successfully. In case the quality of the query is low, the developer is most likely better off reformulating the query than analyzing the search results. In such cases, Refoqus automatically suggests a reformulation of the query.

The goal of Refoqus is to save time for the developer, who would spend less time investigating irrelevant results or trying to come up with query reformulations. In turn, this can reduce the time and the effort spent on daily tasks supported by source

978-1-4673-3076-3/13/$31.00 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Formal Demonstrations

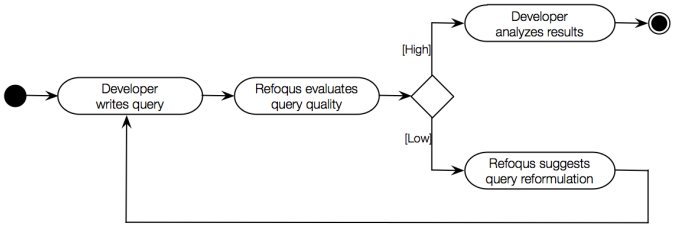

Fig. 1. How to search code in Refoqus

code search, such as, locating a bug or a feature in the code, finding reusable code implementing a particular concept, etc. Refoqus is implemented as an Eclipse plugin, such that it is accessible to developers directly in the IDE.

II. Query Quality Prediction in Refoqus

In order to predict the query quality, Refoqus uses a combination of techniques from the fields of natural language document retrieval and machine learning. We have previously proposed an approach for query quality prediction for SE tasks [7]. The approach uses 21 measures reflecting textual characteristics of the query and of the entire source code to determine if the written query is of high or low quality. The quality prediction is based on training the system using existing examples of good and poor queries. This approach does not require the execution of the query to provide an estimate of the query quality. An experiment conducted on five software systems indicated that the approach is able to correctly assess the performance of queries for the task of concept location with an accuracy of 79% [7].

While Refoqus builds on this approach, it includes seven additional measures, which convey information about the results returned by the query: Subquery Overlap, Robustness Score, First Rank Change, Clustering Tendency, Spatial Autocorrelation, Weighted Information Gain, and Normalized Query Commitment. They are defined in the field of natural language document retrieval [1] and their use in SE context is a first. Using these additional sources of information leads to better quality prediction in Refoqus, i.e., 86% correct classification when applied to the same datasets as in the previous work [7].

Query quality prediction in Refoqus consists of two main steps, i.e., training a classifier for predicting the quality of queries and using this classifier to predict the quality of incoming queries. Each of these steps is described below.

A. Training the Query Quality Classifier

Refoqus uses classification trees and 28 measures of query quality in order to categorize queries as having high or low quality. In order to classify incoming queries, the classifier first needs to be trained on a set of existing data in order to learn the properties of the queries that make them be of high or low quality. This training is performed for every software system individually and is done before using Refoqus on a new system. The training data and the predictor model is updated every time the developer provides feedback about the quality of a query through the Refoqus' user interface (see Fig. 2). Note

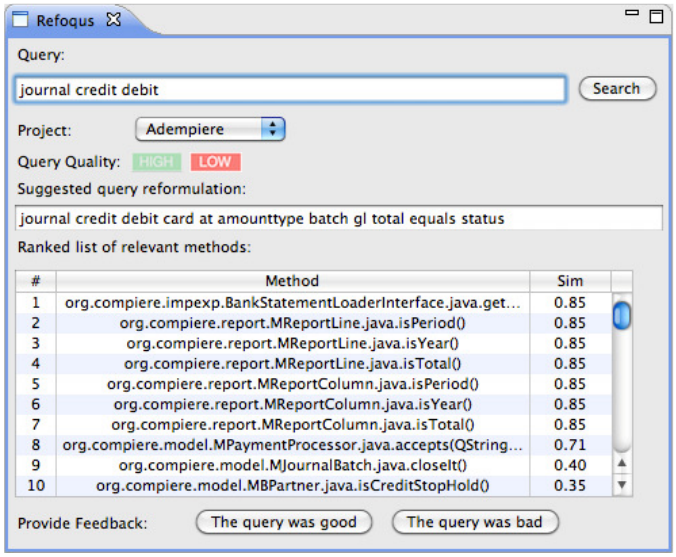

Fig. 2. The Refoqus Eclipse plugin

that we opted for individual training for each system in Refoqus over training on several systems as previous results indicate that it is the approach that leads to best results [7].

The training options of Refoqus can be set from the *Preferences* panel (see Fig. 3), which is integrated with the Eclipse Preferences window. First, the developer selects the project, which Refoqus is to be trained on. Once the project is selected, Refoqus will extract a text corpus from the source code of the software system. It will consider each method as a separate document and will extract the text found in the identifiers and comments in the method. The extracted text will be subject to typical processing techniques for TR applications in SE, such as, identifier splitting (CamelCase and under_score splitting), stemming (Porter stemmer), and stop words removal. Since stop words can be project-specific and can include items like the name of the project or copyright terms, the developer has the option of defining a project-specific list of stop words and selecting it in the Preferences panel. If no list is selected, the default list, composed of English stop words and programming keywords, will be used.

The next training option refers to the way the training data is to be provided to Refoqus. The data consists of a set of queries and their corresponding relevant methods (i.e., the methods that should be retrieved by Refoqus in response to the query). Refoqus offers two options to developers regarding training data. The first option can be used in the cases when the software system, which Refoqus is to be trained on, has a bug tracking system (e.g., BugZilla or Jira) and a versioning control system (e.g., SVN) in place. In this case, Refoqus can automatically extract its training data from these sources. The developer just needs to provide Refoqus with an http link to a list of bug reports or feature requests stored in the bug tracking system and with the address of the versioning control system. Next, Refoqus will download and parse the bug reports and feature requests from the online bug tracking system, extracting the bug/feature id, the title and description of the report. The

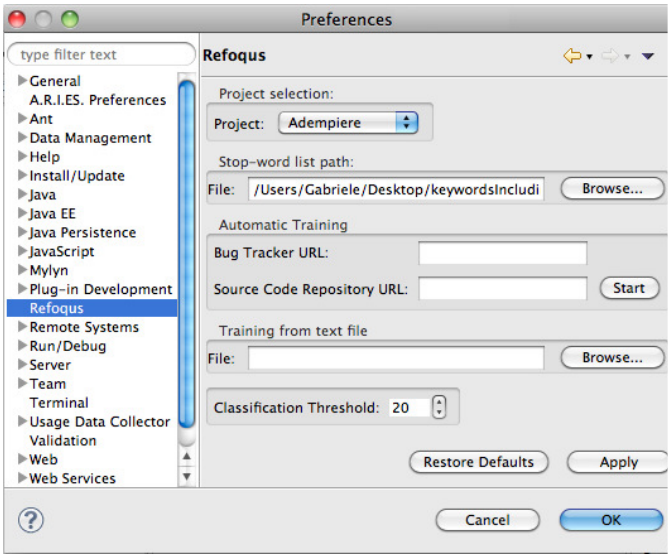

Fig. 3. The Refoqus Preferences Panel

title and description are each used as queries by Refoqus and are subject to the same text processing treatment as the corpus of the software system (i.e., identifier splitting, stop words removal, stemming). The bug/feature id will be used to search the history of the project in the version control system and extract the methods that were changed in order to fix the bug or implement the new feature.

In the case when a bug tracking system (or version control system) is not in place for the current software project, the developer can provide Refoqus with a text file containing the training data. The file will contain one entry per query, structured in the following way: the query appears on the first line, followed by the number of relevant methods to the query on the next line, followed by the qualified names of the relevant methods on the subsequent lines. A blank line divides separate entries. This file needs to be produced by the developer. In our future work we envision extending Refoqus to provide partial automation of this part of the training.

The last training option that can be configured by the developer is the classification threshold (default is 20). This threshold represents the maximum position in the ranked list of results on which a query needs to retrieve a relevant document to be considered a high quality query. For example, if the classification threshold is set to 20 and the first relevant document to the query is retrieved on position 21, the query will be considered of low quality. If, on the other hand, the first relevant method is situated on position 19 in the search result list, the query is considered of high quality. This threshold can vary depending on the needs of the task performed or on the personal preferences of a developer.

Once these options are applied, Refoqus will extract the training data from the selected source and will compute the 28 measures of query quality for each of the queries in the training set. Then, it will run each of the training queries using the IR engine Lucene[1] and categorize the queries as having high or low quality depending on where the first relevant method is found in the result list relative to the classification threshold. Using all these data, Refoqus will build a classification tree and automatically determine the classification rules that categorize training queries for that system with the highest accuracy.

[1] http://lucene.apache.org/

B. Predicting the Quality of New Queries

When the training is completed, the developer can start using Refoqus for code searching by writing a query in the designated text box and pressing the button "Search". Refoqus will then perform two operations simultaneously. First, it will run the query using Lucene and return a ranked list of results, similar to other tools using TR engines for searching code [9, 10]. The novelty for code searching tools is, however, the automatic prediction of the quality of the current query. Refoqus will also highlight the appropriate visual cue in the Refoqus window according to the query quality, i.e., green cue for high quality and red cue for low quality queries (see Fig. 2). The developer can use the cues to determine if investigating the list of results is worth the effort or seeking a reformulation of the query is more advisable. Refoqus can prevent the developer from wasting time investigating irrelevant results returned in response to a low quality query, while still allowing her to use a code search engine the same way she was used to.

III. Query Reformulation Recommendation in Refoqus

In the case of high-quality queries, the developer can proceed with confidence in analyzing the list of search results returned by Refoqus. When the query has low quality, a reformulation of the query is needed. This can be a very difficult task for the developer as there are no apparent clues to how the query should be reformulated. Refoqus makes this task easy by automatically recommending query reformulations for the queries, which are found to be of low quality. Specifically, Refoqus implements our approach to reformulate developer queries [6]. The approach uses four reformulation strategies, commonly used in natural language document retrieval [2] and selects the best one among them for each individual query. Three of these strategies work by adding relevant terms to the original query (i.e., RSV expansion, Dice expansion, and Rocchio expansion [2]), and one removes irrelevant terms from the original query (eliminates the terms that appear in more than 25% of the documents in the corpus, as they are considered non-discriminating [2]).

In order to be able to assess the best reformulation strategy for each query, Refoqus makes use of the 28 quality measures of a query and trains a classifier which is able to decide based solely on these measures the best reformulation approach for a new query. No other search tool to date selects a reformulation strategy for an individual query. Usually, the same reformulation strategy is applied to all queries in need for reformulation [3, 4]. Fig. 2 presents an example of a low-quality query and its reformulation proposed by Refoqus.

Query reformulation in Refoqus consists of two main steps: building a classifier and using the classifier to recommend the best reformulation technique for incoming queries. The following subsections detail each step.

A. Building the Query Reformulation Classifier

Refoqus uses classification trees also for learning the best reformulation strategy for a query. This type of classifiers is particularly useful for the type of training needed by Refoqus, involving a large number of independent variables (the 28 quality measures). Classification trees are able to perform feature selection across all these variables and automatically determine those that are truly relevant for the classification.

Refoqus uses the same input data for training its reformulation classifier as for predicting the quality of queries, i.e., the set of training queries and their relevant methods. It applies each of the four reformulation approaches at a time to each input query, and then uses Lucene to run all the reformulated queries and record the position of the relevant methods in the list of results. By comparing the results obtained by each of the reformulated queries determines which reformulation technique works best for each training query. It then uses the data from all training queries in order to build the classification tree by considering the best reformulation technique as the dependent variable and the values of the 28 query quality measures as the independent variables.

The two classifiers (i.e., for query quality prediction and for query reformulation) are built at the same time, after the developer sets the training parameters in the Preferences panel.

B. Selecting the Best Reformulation for Low Quality Queries

When a query has low quality in Refoqus, the 28 query quality measures of the query are computed and used to automatically determine the best reformulation strategy for the query. This strategy is then automatically applied to the query, resulting in the reformulation recommendation shown in the "Suggested query reformulation" text box (see Fig. 2). The developer can copy the reformulation in the query text box and run it as-is or make modifications to it.

IV. Related Work

Several source code search tools have been proposed in the last decade [9, 10] for supporting various SE tasks. However none of these tools addressed the problem of detecting the quality of the textual queries used with the tools. We have previously proposed and evaluated the first approach assessing and predicting the quality of text queries in the context of SE tasks [7], using pre-retrieval techniques from the field of natural language document retrieval. Refoqus is built based on an enhanced version of this approach, which includes information about the results returned in response to the query. It is the only tool in SE to assess the quality of queries.

Query reformulation for text queries has been addressed by researchers in the context of several SE tasks [3-5, 11], some supported by text retrieval techniques. However, Refoqus is the first tool in SE to automatically determine the best reformulation strategy for each individual query.

V. Conclusion and Work-in-Progress

We proposed Refoqus, an Eclipse plugin for code retrieval that automatically assesses the quality of a query and recommending reformulations for low quality queries.

Refoqus is currently a prototype tool. So far we have evaluated the query quality and reformulation components of Refoqus individually and on synthetic data (i.e., data extracted automatically from bug tracking systems, no users involved). We obtained an average of 84% accuracy in predicting the quality of the query and the proposed reformulations resulted in search results improvement or preservation in 85% of the cases [6]. In the future we plan to perform evaluation studies involving developers using Refoqus during their daily tasks. Our goal will be to determine if Refoqus helps developers save time and effort when searching source code.

As future work we plan to extend Refoqus by (i) adding the functionality to support developers when producing the textual training file and (ii) making Refoqus configurable for advanced user by allowing the customization of the corpus processing used by the TR engine (e.g., identifier splitting, stemming).

Acknowledgements

This work was supported in part by grants from the US National Science Foundation (CCF-1017263 and CCF-0845706).

References

[1] D. Carmel and E. Yom-Tov, Estimating the query difficulty for information retrieval: Morgan & Claypool, 2010.

[2] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," ACM Computing Surveys, vol. 44, 2012, pp. 1-56.

[3] A. De Lucia, R. Oliveto, and P. Sgueglia, "Incremental approach and user feedbacks: a silver bullet for traceability recovery," Proc. Intl. Conf. on Software Maintenance, 2006, pp. 299-309.

[4] G. Gay, S. Haiduc, A. Marcus, and T. Menzies, "On the use of relevance feedback in IR-based concept location," Proc. Intl. Conf. on Software Maintenance, 2009, pp. 351-360.

[5] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," Proc. Intl. Conf. On Automated Soft. Engineering, 2010, pp. 245-254.

[6] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies, "Automatic query reformulations for text retrieval in software engineering," Proc. Intl. Conf. on Software Engineering, 2013, p. Submitted.

[7] S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus, "Automatic query performance assessment during the retrieval of software artifacts," Proc. Intl. Conf. on Automated Software Engineering, 2012, pp.90-99.

[8] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," IEEE Trans. on Soft. Engineering, vol. 32, 2006, pp. 971-987.

[9] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: finding relevant functions and their usages," Proc. Intl. Conf. on Software Engineering, 2011, pp. 111-120.

[10] D. Poshyvanyk, M. Petrenko, A. Marcus, X. Xie, and D. Liu, "Source code exploration with Google," Proc. Intl. Conference on Software Maintenance, 2006.

[11] J. Yang and L. Tan, "Inferring semantically related words from software context," Proc. Working Conf. on Mining Software Repositories, 2012, pp. 161-170.