Predicting Query Quality for Applications of Text Retrieval to Software Engineering Tasks

CHRIS MILLS, Florida State University
GABRIELE BAVOTA, Università della Svizzera italiana
SONIA HAIDUC, Florida State University
ROCCO OLIVETO, University of Molise
ANDRIAN MARCUS, University of Texas at Dallas
ANDREA DE LUCIA, University of Salerno

Context: Since the mid-2000s, numerous recommendation systems based on text retrieval (TR) have been proposed to support software engineering (SE) tasks such as concept location, traceability link recovery, code reuse, impact analysis, and so on. The success of TR-based solutions highly depends on the query submitted, which is either formulated by the developer or automatically extracted from software artifacts.
Aim: We aim at predicting the quality of queries submitted to TR-based approaches in SE. This can lead to benefits for developers and for the quality of software systems alike. For example, knowing when a query is poorly formulated can save developers the time and frustration of analyzing irrelevant search results. Instead, they could focus on reformulating the query. Also, knowing if an artifact used as a query leads to irrelevant search results may uncover underlying problems in the query artifact itself.
Method: We introduce an automatic query quality prediction approach for software artifact retrieval by adapting NL-inspired solutions to their use on software data. We present two applications and evaluations of the approach in the context of concept location and traceability link recovery, where TR has been applied most often in SE. For concept location, we use the approach to determine if the list of retrieved code elements is likely to contain code relevant to a particular change request or not, in which case, the queries are good candidates for reformulation. For traceability link recovery, the queries represent software artifacts. In this case, we use the query quality prediction approach to identify artifacts that are hard to trace to other artifacts and may therefore have a low intrinsic quality for TR-based traceability link recovery.
Results: For concept location, the evaluation shows that our approach is able to correctly predict the quality of queries in 82% of the cases, on average, using very little training data. In the case of traceability recovery, the proposed approach is able to detect hard to trace artifacts in 74% of the cases, on average.
Conclusions: The results of our evaluation on applications for concept location and traceability link recovery indicate that our approach can be used to predict the results of a TR-based approach by assessing the quality of the text query. This can lead to saved effort and time, as well as the identification of software artifacts that may be difficult to trace using TR.

CCS Concepts: • **Software and its engineering → Software maintenance tools;**

Additional Key Words and Phrases: Text retrieval, concept location, artifact traceability

---

Andrian Marcus was supported in part by NSF grants CCF-1526118 and CCF-0845706. Sonia Haiduc was in part supported by NSF grant CCF-1526929.
Authors' addresses: C. Mills, Florida State University, 600 W. College Avenue, Tallahassee, FL 32306, USA; email: chris.mills@aderant.com; G. Bavota, Universita' della Svizzera italiana (USI), Via G. Buffi 13, 6904 Lugano, Switzerland; email: gabriele.bavota@usi.ch; S. Haiduc, Florida State University, 600 W. College Avenue, Tallahassee, FL 32306, USA; email: shaiduc@cs.fsu.edu; R. Oliveto, University of Molise, C.da Fonte Lappone, 86090 Pesche, IS, Italy; email: rocco.oliveto@unimol.it; A. Marcus, The University of Texas at Dallas, 800 W. Campbell Road; MS EC31, Richardson, TX 75080 USA; email: amarcus@utdallas.edu; A. De Lucia, University of Salerno, Via Giovanni Paolo II, 132, 84084 Fisciano (SA), Italy; email: adelucia@unisa.it.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2017 ACM 1049-331X/2017/05-ART3 $15.00
DOI: http://dx.doi.org/10.1145/3078841

RIGHTSLINK

ACM Reference Format:
Chris Mills, Gabriele Bavota, Sonia Haiduc, Rocco Oliveto, Andrian Marcus, Andrian Marcus, and Andrea De Lucia. 2017. Predicting query quality for applications of text retrieval to software engineering tasks. ACM Trans. Softw. Eng. Methodol. 26, 1, Article 3 (May 2017), 45 pages.
DOI: http://dx.doi.org/10.1145/3078841

1. INTRODUCTION

Since the mid-1990s, text retrieval (TR) approaches have been successfully applied to a multitude of software engineering tasks [Arnaoudova et al. 2015; Binkley and Lawrie 2010a, 2010b] such as concept/concern/feature/bug location [Dit et al. 2013; Marcus and Haiduc 2013], refactoring [Bavota et al. 2011], traceability link recovery [Antoniol et al. 2002; De Lucia et al. 2007; Hayes et al. 2006; Marcus and Maletic 2003], impact analysis [Canfora and Cerulo 2005], clone detection [Marcus and Maletic 2001], measuring the quality of code components [Marcus et al. 2008; Poshyvanyk and Marcus 2006], bug triage [Canfora and Cerulo 2006; Anvik and Murphy 2011], and so on.

As with any typical TR application, using TR for software engineering (SE) tasks usually requires a query as input, either formulated manually by developers or automatically extracted from software artifacts. As output, such approaches return a list of ranked software artifacts. A successful retrieval will lead to a result set of artifacts that are useful for solving the current SE task at hand. However, the success of TR approaches in SE depends strongly on the input query and its relationship to the text contained in the software artifacts. For example, it is extremely difficult (or nearly impossible) for TR-based techniques to produce relevant results if there is a strong deviation between the text used in the artifacts and the one used in the query (i.e., the *vocabulary mismatch problem* [Furnas et al. 1987]).

The *quality of a query* captures how well the query retrieves the desired documents when executed by a TR approach. A *high-quality query* retrieves the relevant documents on top of the results list. Conversely, a *low-quality query* either retrieves the desired documents in the bottom part of the list of the results does not retrieve them at all. When low-quality queries are executed, the software engineer will spend time and effort analyzing irrelevant search results and may prematurely lose faith in the usefulness of TR-based tools. Therefore, it would be useful to provide software engineers using TR tools with feedback related to the quality of the query being run, warning them when the query is likely to lead to poor results, or, on the contrary, indicating that the results are likely to contain useful information, when the query is of high quality. The software developer can then take the right action without frustration or wasted time. More than that, in the case where software artifacts are used as queries (e.g., traceability link recovery), predicting that an artifact will retrieve irrelevant results may reveal issues in the quality of the textual information contained in the artifact, making it hard to interpret by TR approaches.

We aim at automatically predicting the quality of TR queries in the context of an SE task and making this information available to the software engineer before his or her analysis of the results begins. To this end, we propose a solution to the problem of query quality prediction in SE by adapting solutions from the field of natural language (NL) document retrieval [Carmel and Yom-Tov 2010] to their use on software data. While similar, the problem of query quality prediction in SE has essential differences with respect to NL document retrieval, due to the properties of software artifacts, which contain different information than NL documents (e.g., the text extracted from the source code is not always correct English). Therefore, we carefully analyzed, selected, and adapted those NL techniques that are applicable to software data and also introduced new techniques. Our query quality predictor, called Q2P, uses machine-learning techniques and a carefully selected set of 28 pre-retrieval (collected before the query is

RIGHTSLINK

run by the TR engine) and post-retrieval (collected after the query is executed by the TR engine) measures of query properties to predict query quality for SE applications. Based on the 28 query measures and a training data set, the predictor learns the characteristics of high- and low-quality queries used in the context of SE tasks and is able to predict the quality of new queries.

We introduced a first approach addressing the problem of query quality prediction in SE [Haiduc et al. 2012], which used 21 pre-retrieval measures of query properties to learn and predict the quality of new queries. In this article, we extend and improve the previous approach by using seven post-retrieval query quality properties. Post-retrieval properties complement pre-retrieval ones by making use of the information captured in the list of results returned by a query when executed by the TR engine. The two combined sources of information lead to better-quality predictions. In addition, we perform a more extensive empirical evaluation by presenting two applications of the proposed approach in a SE context. The first application is for the task of concept location in source code, where we predict the quality of queries formulated by developers or extracted from change requests. Our predictor was able to correctly classify queries as having high or low quality in 82% of the cases. The second application is in the context of traceability link recovery, specifically between source code and documentation. Here, the predictor was used on queries automatically built from the contents of source code classes and was able to correctly predict the quality of these queries in 74% of cases, on average.

Applicability of our approach. Our approach can be applied in the context of SE tasks supported by TR techniques and can bring benefits to developers and to the quality of the software projects. For example, during TR-based concept location a software engineer can use the query quality information as a recommendation of either inspecting the returned results (in the case when the quality of the query is high) or reformulating the query (in the case where low quality is reported for the current query). She could also decide to use a different concept location approach than TR techniques if multiple queries she formulates are all marked as low-quality ones. When adapting our approach to traceability link recovery, the software engineer can use the query quality prediction information to detect artifacts that are hard-to-trace using TR approaches, which are revealed by the low-quality queries. More specifically, a low-quality query in the context of traceability link recovery means that it is unlikely that the list of results contains artifacts that have a true traceability link with the document represented by the query. Therefore, the document is hard to trace using TR techniques (as these do not retrieve the correct links), and the software engineer may consider manually retrieving links for such a hard-to-trace software artifact. In these cases, low-quality queries may reveal also deeper problems, such as a lexical disconnect between the query artifact and other types of artifacts and may require an investigation of the naming conventions and other text found in the query artifact. We present in this article a study on class-to-documentation traceability link recovery, but our approach is generic and can be used for detecting other types of hard-to-trace software artifacts, with little or no change.

The output of Q2P can also be exploited by automatic query reformulation tools (see e.g., Haiduc et al. [2013] and Roldan-Vega et al. [2013]), which can use such knowledge to avoid the (likely useless) reformulation of high-quality queries.

Structure of the article. The rest of the article is organized as follows. Section 2 discusses the related literature, while Section 3 presents the proposed approach. Sections 4 and 5 present the evaluation of the proposed approach in the context of concept location and traceability recovery, respectively. Finally, Section 7 concludes the article and outlines directions for future work after a discussion of the threats that could affect the validity of the results of our experimentations in Section 6.

RIGHTSLINK

2. RELATED WORK

This section first discusses related work on query quality prediction in the fields of Text Retrieval and Natural Language Processing. It then discusses work related to TR queries in SE, and, finally, it briefly describes previous work using TR approaches for the two addressed applications in SE: concept location and traceability link recovery.

Query Quality Prediction. Query quality was first studied in the Text Retrieval community and emerged from the need to explain the high variance in performance across different queries observed during the Text REtrieval Conference (TREC) competitions. A special track was created for this purpose at the TREC conference between 2003 and 2005 named the Robust track [Voorhees 2005]. The participants were first encouraged to decrease the variance of their TR engines across the range of queries, by focusing on improving the performance on queries known to be hard to answer (i.e., having low quality). This was followed in 2004 and 2005 by an additional challenge of predicting the performance of the TR engines on each individual query, that is, predicting the quality of each query. The query quality predictions were done based on different measures that captured various properties of the queries, document collections, and list of search results. The prediction power of each measure was determined by correlating its values with the average precision (AP) values achieved by the queries after execution. A high correlation would indicate that the measure is able to assess the performance of a query, in terms of AP. The correlations obtained were, however, very low and even negative in some cases. The outcome of the Robust track indicated that predicting the quality of queries is a challenging problem and inspired the research on this topic. Since then, numerous approaches for assessing the quality of TR queries have been proposed in the NL document retrieval field [Carmel and Yom-Tov 2010], but the main goal has remained the same: predicting the AP of a query based on measures that correlate with it.

Our goal differs from the work on query quality in the field of NL document retrieval. While establishing a correlation between the quality of a query and the AP may be interesting and useful from a research point of view, it has little practical application in SE. On the other hand, assigning a quality label to each query, for example, high or low, enables the software engineer to use this information as a recommendation of whether the results returned by a TR approach are worth investigating. If the query is estimated to be of low quality, then it is likely that the retrieved results are not satisfactory, and thus, investigating them could lead to time and effort wasted. If the query is of high quality, on the other hand, then the software engineer is likely to find useful information among the top retrieved results.

Query quality metrics like the ones used in the current study have been used in previous studies in the field of NL document retrieval to reformulate the query or re-rank the result set to improve retrieval performance [Kumaran and Carvalho 2009; Dang et al. 2010; Kim et al. 2011]. More closely related studies have also investigated the query quality prediction problem from the perspective of classifying incoming queries as easy to answer (high quality) or hard to answer (low quality) [Grivolla et al. 2005; Vercoustre et al. 2008; Yom-Tov et al. 2005]. In these studies on query quality classification, several approaches have been used, and in each case, decision trees were found to be the most suitable. While we take inspiration from this work in using classifiers, and in particular decision trees for predicting if a query is of low or high quality, our work fundamentally differs from previous work in NL document retrieval.

In particular, we use a unique combination of quality measures for training the classifiers that apply in a broader context than NL. Many of the measures used in NL retrieval rely on the fact that the query and the documents are written in natural language such as English and make use of the syntactic and semantic rules that govern

RIGHTSLINK

this language. On the other hand, software artifacts and the queries used to search them commonly contain terms and constructions that are not correct English and therefore do not adhere to those rules. We carefully selected a set of 24 pre-retrieval (collected before the query is executed) and post-retrieval (collected after the query is executed) query quality measures that do not rely on English rules and are suited to be applied to SE artifacts, while also introducing four new measures. In addition, our work presented here differs due to the context in which the query quality prediction is used. This is the first time query quality prediction has been addressed in the context of SE tasks, such as concept location in source code and traceability link recovery between software artifacts. While the applications are task specific, the construction of the predictor is generic enough that it can be adapted for many other SE tasks that can be formulated as TR problems.

TR Queries in SE. Within SE, the closest work deals with TR query formulation and refinement. Manual query reformulation using ontology fragments has been investigated in the context of concept location by Petrenko et al. [2008]. In this work, developers build and update ontology fragments that capture their knowledge of the system and then reformulate queries based on these fragments, leading to improved results. The quality of the queries is assessed manually and explicitly by the developer. Lemos et al. [2015] compared keyword-based code search with interface-driven code search, which makes use of interface elements such as return and parameter types. Through a user study, they showed cases where each of the querying approaches works best. Bajracharya and Lopes [2012] analyzed the search logs of Koders, a code search engine, and identified the topics developers are searching for and how they formulate their queries. They also unveiled the fact that more than 64% of the users gave up searching due to their queries returning irrelevant results. This underlines the need to let developers know when queries are poorly formulated to prevent them from losing faith in the capabilities of a search engine. The findings of a field study of developer search strategies by Damevski et al. [2016] highlight the difficulties developers face with code search, citing a need for strategies to help developers formulate better queries, as well as better tool integration to assist with poor tool adoption. Furthermore, Ge et al. [2014] determined through a longitudinal study that up to 42% of manual queries return no relevant results and also showed that over 32% of all queries in their sample were adopted from automated reformulation, which shows the practical importance of automation in query reformulation.

A semi-automated approach for reformulating TR queries, which requires the intervention of a developer, is based on using the feedback of a user about the relevance of the top returned results to automatically reformulate the query. This approach has been used to improve TR-based traceability link recovery between various types of software artifacts [De Lucia et al. 2006; Hayes et al. 2006] and concept location in code [Gay et al. 2009].

Another type of semi-automatic query reformulation approach focuses on automatically determining and suggesting reformulation terms to the developer, who has the final word in choosing the terms. The built-in assumption of these approaches is that the queries need improvement (i.e., they have low quality). Hill et al. [2009] capture the context of the query words' usage in the source code and display it as phrases to the developer. The developer can use this information to select reformulation terms. Shepherd et al. [2007] built a code search tool that suggests terms for expanding a query learned from verb-direct object pairs. Jiang et al. [2015] take a different approach entirely, taking in a query and suggesting a list of candidate questions whose answers will fulfill the developer's information need, utilizing an approach similar to, yet more general than, Ferret [De Alwis and Murphy 2008], a tool for mapping conceptual queries to concrete queries.

RIGHTSLINK

A few articles have investigated automated query reformulations, mostly based on reformulating the query using words that are either similar or related in some way to the query terms. Marcus et al. [2004] used Latent Semantic Indexing to determine the most similar terms to the query from the source code and include them in the query. Yang and Tan [2012] use the context in which query words are found in the source code to extract synonyms, antonyms, abbreviations, and related words to include them in the reformulated query. Other approaches make use of external sources of information to determine the related words that should be included in the query. For example, web mining is used [Gibiec et al. 2010; Cleland-Huang et al. 2010] to identify web documents relevant to the query from which to extract domain terms to replace the original query. Other automatic reformulation approaches include *Conquer*, the tool proposed by Roldan-Vega et al. [2013] and Hill et al. [2014] aimed at supporting the reformulation of queries, as well as the thesaurus-based approach presented by Lemos et al. [2014] to automatically expand queries. Kimmig et al. [2011] and Sisman and Kak [2013] use pseudo-relevance feedback and positional proximity to automatically reformulate the query by adding terms from the top ranked artifacts retrieved in response to the query. Finally, Lv et al. [2015] introduced an automatic expansion technique that leverages Application Programming Interface (API) documentation to add the name of relevant APIs into the original user query.

In our previous work [Haiduc et al. 2013], we also proposed an approach, called Refoqus, which recommends an automated query reformulation strategy (e.g., query reduction, Dice expansion, etc.) meant to improve the performance of a particular query. While this approach also exploits query properties to recommend a reformulation, Refoqus always reformulates a query provided as input, independent of its quality. The Q2P approach presented in this article could be used in tandem with Refoqus by first employing Q2P to assess the quality of a query and then applying Refoqus to reformulate only those queries determined by Q2P to be of low quality.

Some studies have also investigated the results of formulating different queries for the same information need [Liu et al. 2007; Starke et al. 2009]. The massive differences between the results returned by the different queries highlight the strong dependence of the retrieval performance on the query and motivate our work. The need for improving the retrieval performances of queries, especially when run on a corpus composed of source code artifacts, has also been highlighted in other works (see e.g., Eddy and Kraft [2014]). In fact, a recent study by Chaparro and Marcus [2016] shows the significant increase in retrieval performance for verbose queries when eliminating even a few ancillary search terms.

This article extends our previous work on query quality prediction for SE tasks [Haiduc et al. 2012] by extending the set of query properties used in building the predictor model, which leads to a higher prediction accuracy. At the same time, this article extends the evaluation previously performed on concept location and presents a new application of the query quality prediction for determining hard-to-trace artifacts in the context of TR-based traceability link recovery. The next section briefly describes previous work on TR support for these two SE tasks.

TR Approaches for Concept Location and Traceability Link Recovery. Since the mid-1990s, hundreds of articles using TR approaches to address the tasks of concept/feature/bug location and traceability link recovery have been published in SE venues. The first works to use TR techniques for these tasks were those by Antoniol et al. [2000] for traceability link recovery and Marcus et al. [2004] for concept location. From these first efforts, TR-based approaches for concept location and traceability link recovery have evolved drastically over time, in two main directions. First, generic techniques adopted as-is from the field of Information Retrieval have been gradually replaced by approaches that are tailored to SE data and configured specifically for

RIGHTSLINK

each software system they are applied on [Kuhn et al. 2007; Biggers et al. 2012; Lohar et al. 2013; Panichella et al. 2013]. Second, single TR techniques have been replaced by complex systems that combine several techniques and use heuristics [Rao and Kak 2011; Thomas et al. 2013; Gethers et al. 2011; Zou et al. 2015], genetic algorithms [Wang et al. 2014; Panichella et al. 2016], or machine-learning approaches [Ye et al. 2014; Binkley and Lawrie 2014; Moreno et al. 2015] to determine the combination that performs the best on a particular dataset. While a detailed account of these approaches is beyond the scope of this article, we refer the interested reader to several surveys for a more in-depth overview of TR-based approaches for concept location [Dit et al. 2013; Marcus and Haiduc 2013] and traceability link recovery [De Lucia et al. 2011; Borg et al. 2014; Nair et al. 2013].

Despite the large amount of existing work using TR to address SE tasks, our work is the first to predict the quality of queries in the context of any SE task, including concept location and traceability link recovery.

3. AN APPROACH FOR QUERY QUALITY PREDICTION (Q2P)

We present an approach for predicting the quality of text retrieval queries in a SE context. The term *query quality* refers here to the ability of the query to retrieve the relevant software artifacts to the task at hand in such a way that they are easily accessible by developers (i.e., they are in the top list of results retrieved by the query). A query that achieves this is considered a *high-quality query*, as opposed to a *low-quality query*, which either fails to retrieve the relevant documents altogether or it places them at or near the bottom of the list of results, making them hard to reach by developers. This definition of high- and low-quality queries can be fine tuned considering the needs of a developer in the context of a particular SE task. For example, in some applications, a high-quality query may be considered one that retrieves *one* of the relevant artifacts in the top-10 list of results. In other applications, however, a high-quality query could be one that retrieves *all* the relevant documents in the top 25 results, and so on. In all cases, our goal is to give developers a clear indication of whether a query is worth pursuing or should be reformulated, our choice of a binary classification (high or low quality). While having more, finer-grained categories would give a more detailed view on the quality of a query, our goal here was pragmatism and clarity to make the decision on the developer's side on whether to reformulate as easy and straightforward as possible. While in this article we adopt specific thresholds and criteria for deeming a query high or low quality based on the tasks we address, we want to make clear that these can be adjusted based on the needs of the developer for the specific task at hand.

Predicting the quality of queries in SE bares obvious resemblance to the analogous problem in the field of TR, which is concerned with retrieving natural language documents. However, the techniques used for NL documents do not always apply to software artifacts. For example, many approaches to determine the quality of queries in TR rely on the rules governing the English language. These rules often do not apply to software artifacts, which contain much more than just natural language. For example, Sridhara et al. [2008] showed that the semantic relationships between words (i.e., synonymy, hyponymy, etc.) differ in source code than in natural language documents. Therefore, one must carefully consider the existing query quality prediction approaches in the context of SE and select only those query properties that are applicable to software artifacts. As mentioned before, for our approach, we performed an analysis of all the techniques existing for NL and we eliminated those that rely on English semantic and syntactic rules. At the same time, since we want to generate and present information about the quality of the query in real time to the developer who is searching the source code, we did not consider approaches that required a long processing time (i.e., higher than the order of seconds). We identified a set of 28 measures of query properties that

RIGHTSLINK

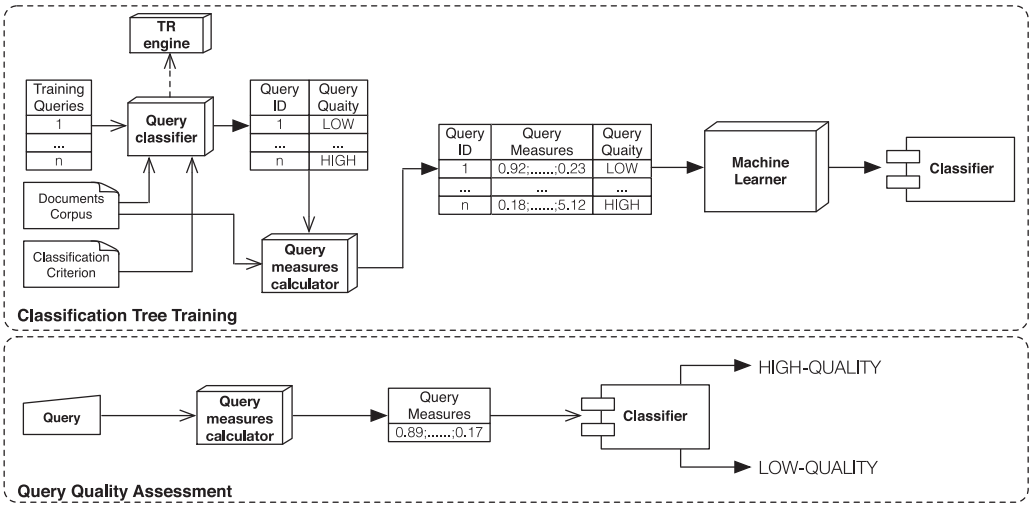

Fig. 1. The proposed approach Q2P.

meet these criteria. In the field of TR, these measures have been mostly used in isolation as predictors of query quality, with very few approaches making use of two or more measures. We propose an approach that initially considers all the 28 query properties we identified and uses a machine-learning approach (i.e., a classifier) to identify a series of rules that distinguish high-quality queries from low-quality ones. Based on these rules, Q2P is able to predict the quality of new queries. Therefore, our approach offers a clear and pragmatic indication to developers if a query is worth pursuing (i.e., high-quality queries) or requires reformulation (i.e., low-quality queries).

Figure 1 depicts our approach. It is composed of two main steps: the training of the classifier (upper part of Figure 1) and the prediction of the quality of queries (bottom part of Figure 1). In the next subsections, we first present the measures of query properties used by the classifier and then we detail the two steps described above.

3.1. Query Properties

Existing query quality properties are categorized into pre-retrieval and post-retrieval [Carmel and Yom-Tov 2010], depending on the moment when they are employed and the type of information about the query they capture. Pre-retrieval properties are computed before the query is run, that is, before the results to the query are retrieved. They capture various linguistic and statistical properties of the query and of the document collection. In contrast, post-retrieval properties make use of the list of results returned by the query and are employed after the query is run and the results are retrieved. The two types of properties capture complementary aspects of the query and our approach makes use of both pre-retrieval and post-retrieval properties.

We want to note that the properties we use (defined in the following subsection) analyze the query in different ways. Many of the properties are applied to each query term individually (and capture some aspects about the query this way), while some focus on relationships between pairs of words in the query, and others consider the query as a whole. These diverse approaches assure that different aspects of the query are captured.

The following subsections explain in detail the pre- and post-retrieval properties we use. The complete formulas formally defining these properties are available in Appendix I.

RIGHTSLINK

Table I. A Bug Report in Filezilla 3.0.0

Title: No confirm for delete in folder view
Reported by: trellmor **Priority:** normal **Component:** FileZilla client
Description: If you try to delete a folder by right-click -> delete in the remote folder window (e.g., the upper right) it won't ask for confirmation. If you do the same in the normal windows (e.g., lower right) FileZilla asks if you really want to delete the file/folder.

Some of the query properties we use may capture similar information. For this reason, we investigate the relationships existing between the 28 query properties in our empirical studies via correlation analysis.

3.1.1. Pre-Retrieval Properties. Our approach makes use of a set of 21 pre-retrieval properties, which assess four different aspects of query quality: *specificity*, *similarity*, *coherency*, and *term relatedness*. The properties were selected among all those proposed in the field of TR such that they can be applied to any type of software artifacts. We present the 21 properties used by the approach below, categorized according to the query quality aspect they capture.

Specificity

Specificity refers to the ability of the query to represent the current information need and discriminate it from others. A query composed of terms commonly used in the collection of documents is considered as having low specificity, as it is hard to differentiate the relevant documents from non-relevant ones based on its terms. For example, consider the bug report in Table I, reported in FileZilla 3.0.0. When searching for the bug in the source code (i.e., concept location), the query "folder view" would have low specificity, since the terms "folder" and "view" both appear many times in the system and in many methods (i.e., documents). On the other hand, the query "remote delete" would have a high specificity, since these terms are specific to only a few documents in the corpus. Specificity properties are usually based on the query terms' distribution over the collection of documents, but the way this information is captured differs from property to property. We use eight properties of query specificity in our approach.

Properties based on the Inverse Document Frequency of a term. The Inverse Document Frequency (IDF) of a term is the inverse of the number of documents in the collection in which a term appears and is a measure of the term's importance for any particular document it appears in. If a term's inverse document frequency is low, then it means the term appears in many documents in the collection, so it is not specific for any document in particular. If, on the other hand, the term appears in few documents, its IDF will be high, and the term is specific and representative for those documents it appears in. A query term with a high IDF makes it easier to retrieve only relevant documents to the query, and thus query terms should have high IDF.

We use three properties based on IDF for capturing the specificity of a query. The Average Inverse Document Frequency (AvgIDF) captures the average value of IDF among all query terms, and a high-quality query should have a high AvgIDF. The Maximum Inverse Document Frequency (MaxIDF), which represents the maximum IDF value across all query terms, is a popular variation of the average and is also expected to assume high values in the case of high-quality queries. The Standard Deviation of the Inverse Document Frequency (DevIDF) captures how much the values of IDF vary among all the query terms. The assumption is that a low variance reflects the lack of dominant, discriminative terms in the query, which may prevent the retrieval of relevant documents.

RIGHTSLINK

Properties based on the Inverse Collection Term Frequency. The Inverse Collection Term Frequency (ICTF) is another way to capture the specificity of a term. ICTF is the inverse of the number of occurrences of a term in the entire document collection. A specific term has a low ICTF, and the assumption is similar to that used in the case of IDF: The more a term is used in the documents in the collection, the less specific it is, leading to a difficulty in discriminating the relevant documents based on it.

Three properties are based on ICTF: the Average Inverse Collection Term Frequency (AvgICTF), the Maximum Inverse Collection Term Frequency (MaxICTF), and the Standard Deviation of Inverse Collection Term Frequency (DevICTF), which represent the average, maximum, and standard deviation of the ICTF values among all the terms in the query. Highly specific terms have high ICTF values, and a highly specific query should have a high AvgICTF, MaxICTF, and DevICTF.

Query Scope (QS) is another specificity property, independent of IDF and ICTF. It measures the percentage of documents in the collection containing at least one of the query terms. A high QS value indicates that there are many candidates for retrieval, and thus separating relevant documents from irrelevant ones might be difficult. Therefore, a query should aim at having a low QS.

The last specificity property we considered is *Simplified Clarity Score (SCS)*, which measures the divergence of the query language model from the collection language model, as an indicator of query specificity. More specifically, the property considers that a query is not specific if the language used in it (i.e., terms and their frequency) is similar to the language used in the entire collection of documents, which indicates a large number of documents that could potentially be retrieved, including many irrelevant ones. A high SCS, indicating a significant divergence of the two language models, is thus desirable.

In addition to the properties existing in the field of TR, we introduced four new properties based on using information entropy to identify specific, high-quality queries. In a preliminary study [Haiduc et al. 2012], we have shown that entropy is a better indicator of query specificity for SE tasks than the leading specificity measures from text retrieval. The entropy of a given term t is computed as

$$entropy(t) = \sum_{d \in D_t} \frac{tf(t,d)}{tf(t,D)} \cdot \log_{|D|} \frac{tf(t,d)}{tf(t,D)}, \tag{1}$$

where D_t is the set documents in the corpus containing the term t and $\frac{tf(t,d)}{tf(t,D)}$ represents the probability that the random variable (term) t is in the state (document) d. Such a probability is computed as the ratio between the number of occurrences of the term t in the document d over the total number of occurrences of the term t in all the documents in the corpus. The entropy has a value in the interval of $[0, 1]$. The higher the value, the lower the discriminating power of the term. We defined four query specificity properties using entropy, AvgEntropy, which is the average entropy value among the query terms, MedEntropy and MaxEntropy, which represent the median and the maximum entropy values across the terms in the query, and DevEntropy, which is the standard deviation of the entropy across all query terms. As low entropy indicates high information content, the desirable values for a high-quality query are low for the first three entropy-based measures. For DevEntropy, high values are desirable.

Similarity

The *Similarity* between the query and the entire document collection is considered as being another indicator of query quality. The argument behind this type of property is that it is easier to retrieve relevant documents for a query that is similar to the collection, as high similarity potentially indicates the existence of many relevant documents

RIGHTSLINK

to retrieve from. For the bug report in Table I, a query "remove directory right click" would have low similarity if its terms are barely found in the system (for example, other terms may be used in the system to name the same concepts: "delete" instead of "remove" or "folder" instead of "directory") or not found at all (for example, "right click" may not be used in any of the methods in the source code). Similarity approaches to query quality make use of a metric called Collection Query Similarity (SCQ). This is computed for each query term and is a linear combination of the collection frequency of a term (CTF) and its IDF in the corpus. We use three query quality properties based on SCQ, namely SumSCQ, which is the sum of the SCQ values over all query terms, AvgSCQ, which is the average SCQ across all query terms, and MaxSCQ, which represents the maximum of the query terms' SCQ values. In the case of every SCQ-based property, a high value is expected for high-quality queries.

Coherency

Another quality indicator for queries is their *coherency*, which measures how focused a query is on a particular topic. The coherency of a query is usually measured as the level of inter-similarity between the documents in the collection containing at least one of the query terms. The more similar the documents are, the more coherent the query is. For example, for the bug report in Table I, the term "window" is contained in many documents that do not have much in common besides the fact that they relate to a window. Since there are many windows, serving different purposes in the system, the documents containing this term will not be very similar to each other, and therefore the query "window" would be incoherent. On the other hand, the term "remote," found in a few documents that all deal with a remote connection and have many terms in common, would be a coherent query. The coherence score (CS) of a term is one of the properties used for this quality aspect and it reflects the average pairwise similarity between all pairs of documents in the collection that contain that particular term. The CS of the query is then computed as the average CS over all its query terms, and it is expected to be high in the case of high-quality queries.

A second approach for measuring the query coherency is based on measuring the variance (VAR) of the query term weights over the documents containing the terms in the collection. The weight of a term in a document indicates the importance, or relevance, of the term for that document and it can be computed in various ways. One of the most frequent ways to compute it, which we also adopt in our implementation, is TF-IDF, that is, a combination between the frequency of a term in the document (TF) and the term's IDF value over the document collection. The intuition behind measuring the variance of the query term weights is that if the variance is low, then the retrieval system will be less able to differentiate between highly relevant documents and less relevant ones, making the query harder to answer. In this case, the query "remote" would be a coherent query based on VAR if its weight would have a high variance, that is, it would be considerably higher in some documents than in others. The documents in that its weight is considerably higher would be considered highly relevant to the query. We use three properties based on VAR, that is, SumVAR, which is the sum of the variances for all query terms, AvgVAR, computed as the average VAR value across all query terms, and MaxVAR, which is the maximum VAR value among the query terms. As in the case of CS, high values are expected for high-quality queries.

Term relatedness

Term relatedness properties make use of term co-occurrence statistics to assess the quality of a query. The terms in a query are assumed to be related to the same topic and are expected to occur together frequently in the document collection. For example, for locating the bug in Table I in the source code, the query "remote transfer" would

RIGHTSLINK

be a high-quality query if the terms "remote" and "transfer" frequently occur in the same methods in the corpus. We use two measures of term relatedness, both using the Pointwise Mutual Information (PMI) metric, which is based on the probability of two terms appearing together in the corpus. The two PMI-based properties are AvgPMI and MaxPMI, which compute the average and the maximum PMI values across all query terms. High average and maximum PMI values indicate a query with strongly related terms.

3.1.2. Post-Retrieval Properties. Post-retrieval query quality properties analyze the list of results retrieved in response to the query and make a prediction based on the language used in the top documents. The list of results provides a different type of information about the query than the pre-retrieval measures. For example, the coherence of the search results, that is, how focused they are on aspects related to the query, is not captured by the query text and is hard to assess without an analysis of the results list. Post-retrieval properties are categorized into three main paradigms, based on the properties of the query and of the result list they capture. These are described below.

Robustness

Robustness-based methods evaluate how stable the list of search results is to perturbations in the query and the documents in the result list. The more robust the result list is to perturbations, the higher the quality of the query. There are properties based on query perturbation, which assess the robustness of the result list to small modifications of the query. When small changes in the query translate to large changes in the search results, the confidence in the capacity of the query to capture the essential information diminishes. Document perturbation properties, on the other hand, rely on injecting the top documents in the result list with noise and re-ranking them, measuring the difference in their ranks before and after the perturbation. In the case of high performing queries, small perturbations of the documents in the result list should not result in significant changes in their ranking. We use five robustness properties: Subquery Overlap, Robustness Score, First Rank Change, Clustering Tendency, and Spatial Autocorrelation.

Subquery Overlap perturbs the query and captures the extent (i.e., the standard deviation) of the overlap between the result set retrieved by the entire query and the result sets retrieved by individual query terms. This is based on the observation made in the field of TR that some query terms have little or no influence on the retrieved documents, especially in the case of low-quality queries. A low standard deviation of the overlap values indicates that the list of results is robust to modifications of the query, indicating a high-quality query. For example, considering the bug report in Table I, the query "remote transfer" would be of high quality based on this metric if the individual terms "remote" and "transfer," used as separate queries, would return very similar lists of results.

To measure the *Robustness Score*, a document perturbation property, the terms' weights in the top relevant documents are slightly perturbed and the resulting documents are re-ranked. The correlation between the initial rank and that after modification is considered. The higher the robustness score, the higher the quality of the query.

First Rank Change captures the probability of a document found on the first position in the list of results to still remain on the first position after a perturbation is applied to it. A high-quality query will have a high first rank change, corresponding to a high probability that the first ranked document will remain the same across perturbations.

Clustering Tendency is another document perturbation property, capturing the cohesion of the top retrieved documents as the textual similarity between them. The higher the clustering tendency, the better the query.

RIGHTSLINK

The *Spatial Autocorrelation* property replaces the retrieval-scores of each top relevant document with the average of the scores of its most similar documents. The linear correlation between the new scores and the original ones is the spatial autocorrelation of the query. The higher the spatial autocorrelation, the higher the quality of the query.

Score distribution

Score distribution-based peoperties analyze the strength of the similarity between the query and the results. For example, the highest retrieval score and the mean of top scores indicate query quality, since, in general, low scores of the top-ranked documents indicate some difficulty in retrieval. We use two score distributions properties, described below.

Weighted Information Gain (WIG) measures the divergence between the mean retrieval score of top-ranked documents and that of the entire corpus. The hypothesis is that for a high-quality query, the top-ranked documents will be much more similar to the query than a random document in the rest of the corpus. The higher the weighted information gain, the better the query.

Normalized Query Commitment (NCQ), on the other hand, measures the standard deviation of retrieval scores in the top k documents returned in response to query, normalized by the score of the whole collection. The higher NCQ, the higher the quality of the query.

3.2. Training the Classifier

To predict the quality of queries, Q2P uses a classifier, which requires training on a set of queries to discriminate among *low-* and *high-quality* queries. To this aim, two things are needed:

(1) a *training data set* consisting of queries and their associated relevant documents.
(2) a *classification criterion* defined by the user (i.e., the developer) on how to discriminate between *low-* and *high-quality* queries.

These two points strongly depend on the SE task addressed. For example, if the task to perform is concept location, the training queries could be automatically extracted from bug tracking systems (i.e., BugZilla) from the text present in bug reports associated with fixed bugs. Using fixed bugs, it is easy to identify the documents relevant to the queries by looking at the source code components (i.e., methods, classes, etc.) changed to solve the bugs. Concerning the criterion defined by the developer to discriminate between *low-* and *high-quality* queries, a practical one for the concept location task is

$$queryQuality_{cl} = \begin{cases} low\text{-}quality & \text{if } FRD_{rank} > 20 \\ high\text{-}quality & otherwise \end{cases},$$

where FRD_{rank} is the rank of the First Relevant Document retrieved by a TR engine using a given query. In other words, if the query is able to retrieve the first relevant document in the top 20 positions in the ranked list of results, than it is considered *high quality*, otherwise *low quality*. Clearly, different developers could set different classification criterion based on their "personal view" of high- and low-quality queries. For example, in the context of a concept location task, some developers might be willing to analyze the first 20 documents in the ranked list before giving up. Others might be less patient, only focusing on the very top part of the ranked list (e.g., top 5 documents). Q2P can be made to work with any of these classification criteria.

For this criterion in our concept location study, we based our choice of using the top 20 ranked documents on two aspects. First, 20 has been considered in previous concept location studies as a reasonable approximation of the maximum number of results a developer would be willing to look at before abandoning the search [Thomas et al.

RIGHTSLINK

2013; Nguyen et al. 2011]. Second, a developer's query for relevant methods is a search for information, which can be considered analogous to a web-based query. Empirical studies on web-based search behavior have shown that users place more weight on the first pages of results. For example, two studies [Jansen et al. 2000; Silverstein et al. 1999] found, respectively, that 77% and 92.7% of users only analyzed results returned on the first two pages, where a page is usually considered as 10 results [Spink et al. 2001].

As a further example, if the task at hand is traceability link recovery between documentation and source code, the training data would consist of existing validated traceability links, where parts of the documentation could be the queries and the relevant documents would be the code components they link to (or the other way around). A possible criterion to discriminate between *low-* and *high-quality* queries in case of traceability recovery could be

$$queryQuality_t = \begin{cases} low\ quality & \text{if (when } recall > 60\% \rightarrow precision < 20\%) \\ high\ quality & otherwise \end{cases},$$

where *recall* and *precision* [Baeza-Yates and Ribeiro-Neto 1999] are two well-known TR metrics widely used to assess the performance of traceability recovery techniques and are defined as follows:

$$recall = \frac{|correct \cap retrieved|}{|correct|}\%$$

$$precision = \frac{|correct \cap retrieved|}{|retrieved|}\%.$$

In particular, recall measures the percentage of links correctly retrieved by an automated method, while precision measures the percentage of retrieved links that are correct.

In terms of the thresholds for precision and recall for the traceability link recovery study, we adopted the classification threshold from previous work [Hayes et al. 2006], which considered a value of 60% for recall and 20% for precision as acceptable for traceability link recovery results.[1]

Given the training data and the chosen classification criterion, Q2P uses the TR engine to classify the queries as *low* or *high quality* (see Figure 1). In particular, each query is executed by the TR engine and, analyzing the ranked list of retrieved documents, is classified as *low* or *high quality* according to the chosen classification criterion. Then, the values of the 28 properties of each query is computed. Finally, the classifier is trained using the collected training data. Each data point in the final training data used by the classifier corresponds to a query and has 28 values corresponding each to one of the query properties and one additional value corresponding to the query classification, that is, *low* or *high quality*.

Our approach uses the *Weka* [Witten and Frank 2011] implementation of the Random Forest machine-learning algorithm [Breiman 2001]. The Random Forest algorithm builds a collection of decision trees with the aim of solving classification-type problems, where the goal is to predict values of a categorical variable from one or more continuous and/or categorical predictor variables. The categorical dependent variable is represented by the quality of a particular query (i.e., *low* or *high quality*), while the independent variables are the 28 query properties described in Section 3.1. The classifier uses the training data to automatically select the independent variables and

[1]For a detailed example-based explanation of why the following evaluation criterion could be suitable for the traceability recovery task, please see our online appendix [Bavota et al. 2016].

RIGHTSLINK

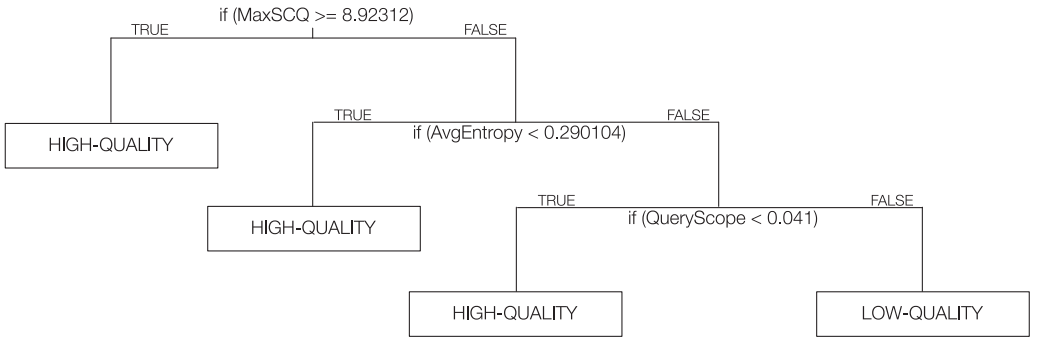

Fig. 2. An example of decision tree generated by the Random Forest.

their interactions that are most important in determining the dependent variable to be explained.

We have chosen Random Forest after experimenting with different machine-learning algorithms, in particular RandomTree, J48, Bayesian Network, Logistic Regression, and Bagging classifiers (details of the comparison are in the online appendix [Bavota et al. 2016]).

Note that the Random Forest, as most of the classification algorithms, performs implicit feature selection. Thus, it could accept as input all 28 properties of a query and will efficiently determine automatically those properties that are relevant for the classification. However, in our evaluations reported in Sections 4 and 5 we also study the correlation of the 28 properties to identify those properties capturing the same signals in the data. This would allow us to avoid the computation of some properties, thus saving computational resources and time.

There are two possible approaches to train the classifier, namely *within-project* and *cross-project* training. In the former approach, the classifier is trained independently for each software system, using training data only from one single system. Conversely, *cross-project* training may use any available training data, from any system. In our previous work [Haiduc et al. 2012] we showed the definitive superiority of the *within-project* training. For this reason, we adopt this kind of training in our approach.

The output of the training stage is a Random Forest classifier, represented by a collection of decision trees composed by yes/no questions that split the training sample into gradually smaller partitions that group together cohesive sets of data, that is, those having the same value for the dependent variable. An example of a partial decision tree is reported in Figure 2.

3.3. Using the Classifier to Assess the Quality of New Queries

Once the classifier is built, it can be used to automatically assess the quality of a given query (see bottom part of Figure 1). When a new query is issued (manually or automatically) to the TR engine, Q2P computes the 28 properties of the new query. Based on the classification tree and the 28 properties, Q2P automatically classifies the quality of the given query as being *low* or *high*. Based on the SE task to be performed, this information can be useful in different ways. Referring back to our previous examples, if the task at hand is concept location and our technique classifies a query written by the developer as a *low-quality* one, then he or she could reformulate the query without spending time in analyzing the likely useless documents retrieved by the TR engine or simply decide to not rely on TR-based techniques for the concept location task. As for traceability recovery, where the query is represented by a software artifact (e.g., an use case), a *low-quality* query could indicate "hard-to-trace artifacts," that is, artifacts

RIGHTSLINK

difficult to trace using a TR technique. So the developer will know that he or she will have to invalidate many candidate links before finding the correct ones and thus decide to manually retrieve the links for such artifacts.

One important point to discuss is the overhead paid in terms of execution time when using our approach. On the one hand, the execution time might be considered not important when applying Q2P to SE tasks like traceability recovery. Indeed, in this case, the developer could simply run our approach and wait for the identification of hard-to-trace artifacts before starting the traceability recovery task. On the other hand, Q2P is also meant to be used in real time by developers when performing TR-based SE tasks like concept location. In such a context, the developer would write a query and Q2P would provide immediate feedback about its quality (i.e., would indicate whether a reformulation is needed). The time needed to compute the quality of a query strongly depends on (i) the size of the document corpus on which the query is run, (ii) the size of the document corpus vocabulary (i.e., how many different terms appear in the document corpus), and (iii) the number of terms composing the query. Evidence in the literature shows that typical queries written by developers are composed by up to six terms [Damevski et al. 2016]. For such queries, the time needed to predict their quality would be ∼4s for document corpuses up to 5K documents and 10K terms in the vocabulary, and ∼30s when the document corpus scales up to 30K documents and 20K terms in the vocabulary. This execution time includes the computation of the query properties for the query as well as its evaluation with the Random Forest model.[2] However, it does not include the time needed to index the document corpus. It is important to note that corpus indexing (i) must be executed only once, therefore having no direct impact on real-time performance, and (ii) is not specific to Q2P, as it is something needed to adopt any TR-based approach. The process of indexing in our case took ∼45s for the largest corpus, Eclipse.

4. ASSESSING QUERY QUALITY DURING CONCEPT LOCATION

The first study evaluating our approach is in the context of concept location in source code. Concept location is an activity performed during software change, concerned with identifying a point in the source code (e.g., a class or a method) where a change needs to be made to implement a given change request [Marcus et al. 2004]. TR approaches are used to retrieve the part of the code for the point of change using a query formulated based on the change request [Marcus et al. 2004]. This section describes the evaluation on concept location in detail.

4.1. Study Design

The goal of this study is to determine how well our approach performs when predicting the quality of TR queries in the context of concept location. There are several aspects we want to evaluate. First, we want to investigate the relationships existing between the 28 query properties exploited by our approach. Such an analysis will provide us with useful insights to understand which properties provide the same indications about the quality of a query and can then be ignored in our classifier since they are providing redundant information.

[2]Note that these execution times have been observed with our Java implementation of the query properties computation run on a machine having a 3.6GHz quad core processor and 16GB of RAM. A lower execution time can be obtained by improving performance optimization (e.g., massive usage of multi-threading).

RIGHTSLINK

Therefore, we first formulated the following research question:

—**RQ**$_1$: Are there query properties capturing the same information about the quality of queries?

To answer RQ$_1$, we compute the Kendall rank correlation coefficient (i.e., Kendall's τ) [Daniel 1978] between all possible 378 pairs of measures to determine whether there are pairs exhibiting a strong correlation. We adopted the Kendall's τ, since it does not assume the data to be normally distributed nor the existence of a straight linear relationship between the analyzed pairs of measures. Cohen [Cohen 1988] provided a set of guidelines for the interpretation of the correlation coefficient. It is assumed that there is no correlation when $0 \leq \tau < 0.1$, small correlation when $0.1 \leq \tau < 0.3$, medium correlation when $0.3 \leq \tau < 0.6$, and strong correlation when $0.6 \leq \tau \leq 1$. Similar intervals also apply for negative correlations. Besides investigating the correlation between the 28 properties exploited by Q2P, this analysis will be also used to perform feature selection for our Random Forest classifier. In particular, for each pair of properties exhibiting a strong correlation (i.e., with a Kendall's $|\tau| \geq 0.6$), one has been randomly excluded when building the classifier. This allows us to reduce the number of measures exploited by our approach, lowering computation time and risk of overfitting the model.

The second research question we formulated aims at assessing the accuracy of our approach as compared to a set of baseline techniques:

—**RQ**$_2$: How accurate is Q2P in predicting the quality of queries for TR-based concept location and how does its accuracy compare to baseline classifiers?

To answer RQ$_2$, we require a dataset of classified queries (i.e., queries labeled as high- or low-quality). The process we used to obtain such a dataset is detailed in Section 4.2.

We performed a 10-fold cross validation over the dataset, computing the overall average accuracy of Q2P when only relying on pre- and post-retrieval properties in isolation as well as when combining the two categories of properties (i.e., when considering all properties together).

Since our dataset is unbalanced, as we will show later (Table III), we balance the training set at each iteration (i.e., for each of the 10 folds) by exploiting the Synthetic Minority Oversampling TEchnique (SMOTE) [Chawla et al. 2002]. SMOTE rebalances the training set by creating artificial instances obtained by joining nearest neighbors of the minority class instances. While we did balance the training set to build the classifier, the test set was never modified to avoid any bias.

We assess the overall performance of Q2P with its average accuracy and by reporting the number of Type I and Type II errors. A Type I error occurs when the model wrongly classifies a high-quality query as low-quality, while a Type II error is when the model wrongly classifies a low-quality query as high-quality.

We compared our approach with three basic/baseline classifiers: a random classifier and two variants of a constant classifier (pessimistic and optimistic). The random classifier randomly selects a prediction from the possible values, that is, high or low quality. The two values have the same probability to be selected. The constant classifier always predicts a specific value disregarding the instance. In particular, the pessimistic constant classifier always classifies a query as low quality, while the optimistic constant classifier works by always classifying a query as high quality.

We performed this comparison to ensure that Q2P is smarter than just a random or constant classifier. This kind of comparisons are performed in the field of machine learning to detect possible anomalies or biases of new classifiers [Matjaz and Kononenk 2002].

RIGHTSLINK

Table II. The Systems Used in the Concept Location Study and Their Properties

System	Version	Language	KLOC	#Methods	Avg. Commit Size*	St. Dev. Commit Size**	#Queries
Adempiere	3.1.0	Java	330	28,355	1.18	0.52	51
Nutch	2.1	Java	29	2,113	4.71	7.74	54
ATunes	1.10.0	Java	80	3,481	1.76	1.18	51
BookKeeper	4.1.0	Java	38	3,381	10.59	14.88	81
Derby	10.9.1.0	Java	633	41,038	33.24	123.00	99
Eclipse	2.0	Java	2,500	76,335	1.71	1.29	51
FileZilla	3.0.0	C++	240	3,240	1.97	2.00	87
JEdit	4.2	Java	250	5,532	2.00	2.07	54
Mahout	0.4	Java	110	15,338	3.72	3.49	54
Mahout	0.8	Java	123	9,479	5.09	6.34	66
Tika	1.3	Java	46	3,401	3.30	4.24	60
WinMerge	2.12.2	C++	410	8,012	1.96	1.79	69
Total	—	—	**4,789**	**199,705**	**5.94**		**777**

*Average number of methods changed per commit for each system.
**Standard deviation of the number of methods changed per commit for each system.

4.2. Context

To collect the queries needed for the study, we used an approach frequently adopted in concept location empirical studies, based on change reenactment and user simulation [Jensen and Scacchi 2006]. We collected queries for 12 open source Object-Oriented (OO) systems from different problem domains, implemented in Java and C++, which are summarized in Table II.

We reenacted the concept location that occurred while fixing bugs in these systems by following a few steps. First, the online bug tracking systems of each of the 12 systems were consulted and a set of closed bug fix requests was identified. The selected bug reports correspond to bugs that are present in the version of the software system used in our study, but fixed in a later version.

While informally inspecting the bug reports, we noticed that they seem to be written by a mix of users and developers and that the two types of authors tend to use different vocabularies, one more technical than the other. End-users often describe the problem in terms of the UI elements they see and interact with on the screen, and the questions they ask as well as the answers they provide in comments often reveal the lack of technical expertise. For example, for Adempiere bug #514556, the reporter asks questions in the comments such as "What can I do so you can see the error?," "It is a problem with the database?," "Do I need to reinstall all?," and "What do I do?" Developers writing bug reports, on the other hand, sometimes refer (in comments or in the bug description) to previous similar problems encountered, they may try to estimate the general region of the code where the issue is located, they may offer suggestions for potential solutions, or they may fix the bug themselves. For example, for Eclipse bug #21062, which is part of our dataset, the author of the bug report is also the one that submitted the patch to fix it. While we did not explicitly study the impact of the author type on the results of our approach, our future work will investigate this aspect in detail.

We also determined the set of methods that were modified to fix each bug, based on the patches attached to the bug reports in the online bug tracking systems. This set of methods represents the oracle for concept location. We will refer to these methods as the *target methods*. So, during the original concept location, it is likely that the developer started with the bug request and ended up deciding to change one of the target methods (and later the other ones).

Since we do not know what queries developers formulated, for each change request, we created three queries. The first query was obtained from the title of the bug report

RIGHTSLINK

(i.e., the short description); the second query represented the description of the bug (i.e., the long description); finally, the third one was composed by the concatenation of the bug title and description. These three types of queries are often used in concept location studies [Rao et al. 2015; Scanniello et al. 2015; Biggers et al. 2012; Thomas et al. 2013]. As usually done for concept location evaluations, trace information or log files contained in these descriptions were eliminated prior to the extraction. We then normalized the text by using simple identifier splitting (we also kept the original identifiers), stop words removal (i.e., we removed common English words and programming keywords), and stemming (we used the Porter stemmer [Porter 1980]). Table II reports the number of queries for each system.

For example, from Bug #1605980 of Adempiere, we obtained the following three queries after extraction and normalization (in parenthesis is the original text extracted from the bug reports before the normalization):

(1) From bug title: *invoic process draft select*
(Original title: Print Invoices process - draft and selection)
(2) From bug description: *us garden world select date rang in todai all invoic select regardless document statu client bad print post custom us email option draft potenti cancel invoic sent*
(Original description: Using Garden World, if you select a date range from somewhere in 2001 to today then ALL invoices are selected regardless of document status OR client!!! Not so bad if you are printing them and posting them to customers but if you use the email option then drafted (and potentially cancelled) invoices are sent too!)
(3) From bug title + bug description: *invoic process draft select us garden world select date rang in todai all invoic select regardless document statu client bad print post custom us email option draft potenti cancel invoic sent*

While fixing this bug, the method changed by the developers (i.e., the target method) was `doIt()`, found in the `process` package, `InvoicePrint.java` file, and `InvoicePrint` class. The document corresponding to this method is the one that the queries are supposed to retrieve during TR-based concept location.

For each system, a source code corpus used by the TR engine needs to be built. We considered every method in the system as a separate document. For each method, we extracted the terms found in its source code identifiers and comments and normalized this text using the same techniques previously applied on the queries: identifier splitting, stop words removal and stemming. The chosen TR technique was Lucene,[3] which is a popular and improved implementation of the Vector Space Model. Lucene was used to index the source code corpus and to search the source code using the defined queries. Note that Q2P can be used with any TR technique.

During concept location, it is important that developers find their target method (i.e., the method where they have to start the change) as fast as possible. Other methods that change are usually identified during impact analysis. When reenacting concept location, the success criterion is translated into the rankings of the target methods (as opposed to many other TR applications where recall and precision are considered). In other words, if any of the target methods ranks in among the top retrieved results, we consider it a successful retrieval.

A rule often used in concept location applications is that finding a target method among the top 20 ranked results is considered a good result, based on the observation that most developers would rarely look at more than 20 methods before reformulating their query. Hence we define a query as *high quality* if any of the target methods is

[3]https://lucene.apache.org/core/.

RIGHTSLINK

Table III. The Quality of the Queries Used in the Concept Location Study

System	Total Number of Queries	High-Quality Queries	Low-Quality Queries
Adempiere	51	23	28
Nutch	54	27	27
ATunes	51	20	31
BookKeeper	81	44	37
Derby	99	26	73
Eclipse	51	15	36
FileZilla	87	19	68
JEdit	54	24	30
Mahout 0.4	54	25	29
Mahout 0.8	66	34	32
Tika	60	23	37
WinMerge	69	23	46
Total	**777**	**303**	**474**

retrieved in the top 20 results. Otherwise, we consider the query as having *low quality*. We classify in this way all the queries used in our evaluation. In the above example, if a query returns the target method `doIt()` in top 20, then it is considered of high quality. Note that this threshold can be changed easily, as needed.

Knowing the target methods beforehand (from the submitted patches) allowed us to categorize all the queries used in the study following the same procedure. Table III shows the actual number of high- and low-quality queries for each system used in our study. Note that this reflects the actual quality of the queries, not the predicted one.

One interesting observation is that the number of low-quality queries is always higher (for some systems 2 to 4 times higher) than the number of high-quality queries. This means that, when TR techniques are applied for concept location in these systems, it is likely that developers will have to investigate more than 20 irrelevant artifacts before finding a relevant one. This underlines the need for query quality prediction, which would prevent developers from investigating the results of such low-quality queries and would prompt them to reformulate the query instead.

Having such a difference in the number of high- and low-quality queries can also represent a challenge for our approach, as our approach would have significantly less examples to learn from for determining the properties that high-quality queries share. As previously explained, we exploit the SMOTE technique [Chawla et al. 2002] to rebalance the training sets in our evaluation.

One other point to discuss is the size of the gold set for a query (i.e., the number of target methods). While it may seem obvious that the more methods there are in the target set, the easier it would be to retrieve one of them (and therefore have a high-quality query), our observations indicate that this is not necessarily the case. We investigated the 10 largest commits in the dataset and found that their queries have mixed quality. These commits range in size from 20 to 702 methods and come from four different systems: Derby, BookKeeper, Nutch, and Mahout 0.8. Of the 30 queries formulated for the 10 largest commits, only 12 were high-quality queries, whereas 18 were low quality and therefore not able to retrieve any of the relevant methods in the top 20 results. This makes us believe that, while it may be the case that the size of the commit can help certain queries, this does not hold universally. To provide statistical evidence for this claim, we computed the Kendall's τ correlation coefficient between commit size and the number of high-quality queries for all bugs in our dataset. The calculated coefficient of 0.154 indicates a low positive correlation (p-value $= 0.003$). Therefore, we are confident that any bias introduced by bugs associated with large commit sizes is minimal.

RIGHTSLINK

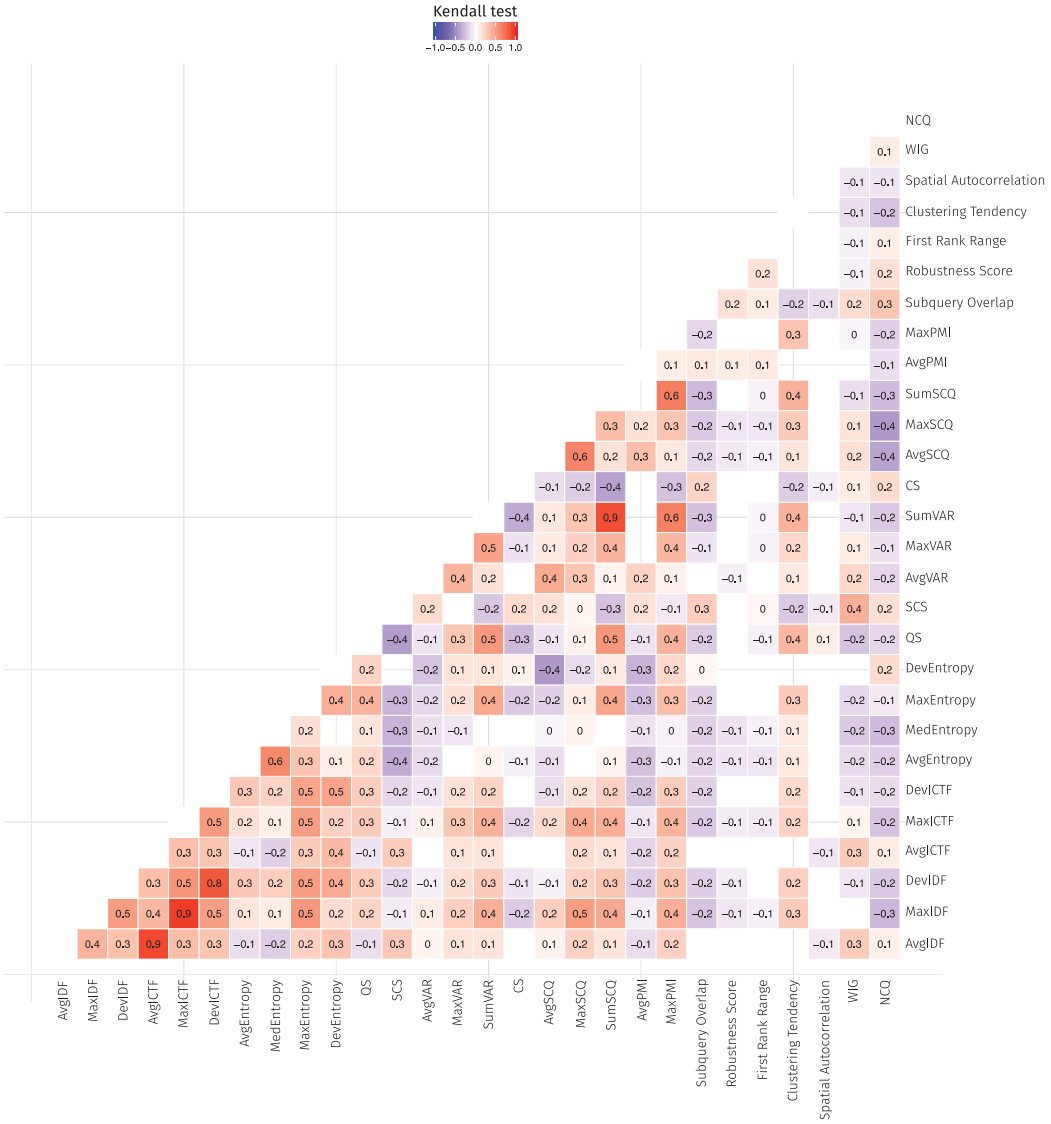

Fig. 3. Kendall τ between the 378 pairs of query properties.

4.3. Results and Discussion

In this section, we provide answers to our research questions.

4.3.1. Are There Query Properties Capturing the Same Information about the Quality of Queries? Figure 3 depicts a heatmap reporting the Kendall's τ between all 378 pairs of 28 query properties considered in our approach. Note that we compute such data by considering the query properties collected for all 12 systems (i.e., we considered all 777 queries as a single dataset for this analysis). Note that we only report statistically significant correlations (i.e., those having a p-value < 0.05), showing a white square in correspondence to non-significant correlations (e.g., the one between DevEntropy and MedEntropy).

RIGHTSLINK

The first noticeable thing is the very strong correlation ($\tau \geq 0.8$) existing between properties based on IDF and those based on ICTF. This result is not entirely surprising given that the IDF is the inverse of the number of documents in the collection in which a term appears while the ICTF is the inverse of the number of occurrences of a term in the entire document collection. Both of them assess the specificity of a query term by capturing its spread in the document corpus, the primary difference being that ICTF accounts also for the term's frequency in documents rather than just its presence. These measures have been found to correlate also on some natural language corpora [Kwok 1995].

A very strong correlation ($\tau = 0.9$) can also be observed between SumSCQ and SumVAR. Such a correlation was less expected, considering the fact that SumSCQ and SumVAR are supposed to capture different aspects of query quality (i.e., similarity and coherence, respectively). However, when analyzing the formulas of these measures in detail some mathematical similarities become apparent. Note that both SCQ and VAR are scaled by the inverse document frequency (idf). Moreover, SCQ relies on the term frequency (tf) of a term in all documents in the corpus, while VAR considers the frequency of a term in each document individually. Given that summing all term frequencies over documents containing a term is equivalent to the collection term frequency utilized in SCQ, it makes sense that while the average and maximum analogs of SQC and VAR seem to capture different properties of the query, SumVAR and SumSQC are in some ways related.

Other strong correlations are observed between MedEntropy and AvgEntropy ($\tau = 0.6$), MaxSCQ and AvgSCQ ($\tau = 0.6$), and MaxPMI and both SumSCQ/SumVAR ($\tau = 0.6$). The first two correlations are quite intuitive to explain. Indeed, the pairs of correlated properties are built on top of the same measure (entropy and SCQ) captured for each term in the query. The only difference is the way such a measure is aggregated in the property (e.g., median vs. average). More interesting is the third correlation involving MaxPMI and SumSCQ/SumVAR (which, as previously shown, correlate as well). The PMI measure is computed for each pair of terms in the query and represents the probability that they appear together in the corpus. In general, correlation between MaxPMI and SumSCQ/SumVar can occur when the queries either (i) contain a pair of terms that have a high probability to co-occur in the corpus and also the query terms appear many times in the corpus or (ii) do not contain pairs of terms that are likely to co-occur, and also the query terms generally appear scarcely in the corpus. This highlights the dependence of these correlations on the nature of both the query and corpus terms.

Finally, it is worth highlighting that almost no correlation exists between pre- and post-retrieval properties, therefore reinforcing the idea that they capture different signals in the data.

Summary for RQ_1. In our concept location dataset, ICTF- and IDF-based properties strongly correlate, as well as SumSCQ and SumVAR. Other strong correlations involved pairs of properties computed based on the same primary measure (e.g., MedEntropy and AvgEntropy). Also, we observed a correlation between MaxPMI and both SumSCQ/SumVAR. Given the results of this research question, we excluded from our prediction model for concept location the following metrics: AvgICTF, MaxICTF, DevICTF, AvgEntropy, MaxSCQ, SumSCQ, and MaxPMI. Thus, the model used in RQ_2 is built on top of the 21 query measures left.

4.3.2. How Accurate Is Q2P in Predicting the Quality of Queries for TR-Based Concept Location and How Does Its Accuracy Compare to Baseline Classifiers? Table IV shows the accuracy for

RIGHTSLINK

Table IV. Q2P When Using Pre-Retrieval Query Properties Only, Post-Retrieval Only, and All Query Properties: Corr = Correct, T I = Type I errors, T II = Type II errors

	Q2P - pre-retrieval			Q2P - post-retrieval			Q2P - all properties		
System	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II
Adempiere	0.82	0.04	0.14	0.36	0.32	0.32	0.88	0.04	0.08
Nutch	0.78	0.08	0.14	0.74	0.10	0.16	0.84	0.06	0.10
ATunes	0.76	0.12	0.12	0.74	0.16	0.10	0.80	0.10	0.10
BookKeeper	0.72	0.14	0.14	0.61	0.24	0.15	0.78	0.14	0.09
Derby	0.78	0.11	0.11	0.74	0.12	0.13	0.82	0.11	0.07
Eclipse	0.84	0.08	0.08	0.78	0.12	0.10	0.90	0.08	0.02
FileZilla	0.76	0.15	0.09	0.69	0.15	0.16	0.81	0.14	0.05
JEdit	0.78	0.12	0.10	0.68	0.20	0.12	0.84	0.12	0.04
Mahout 0.4	0.74	0.14	0.12	0.68	0.16	0.16	0.78	0.10	0.12
Mahout 0.8	0.80	0.10	0.10	0.68	0.13	0.18	0.80	0.12	0.08
Tika	0.72	0.12	0.17	0.58	0.22	0.20	0.77	0.12	0.12
WinMerge	0.72	0.15	0.13	0.67	0.15	0.18	0.77	0.15	0.08
Average	**0.77**	**0.11**	**0.12**	**0.66**	**0.17**	**0.16**	**0.82**	**0.11**	**0.07**

the 12 software systems (i.e., percentage of correctly classified queries) achieved by Q2P when only exploiting pre-retrieval properties, when only exploiting post-retrieval properties[4] and when considering all properties together. Table IV also reports the number of Type I and Type II misclassifications on each system. When using only pre-retrieval properties, the accuracy of Q2P is quite good, with an average of 0.77 (min = 0.72, max = 0.84).

Performances are much worse when only relying on post-retrieval properties (see Table IV). In this case, the average accuracy of Q2P drops to 0.66. Such a result might be due to the higher number of pre-retrieval properties taken into consideration by our approach with respect to the post-retrieval ones. In other words, having more properties to exploit in the "only pre-retrieval model" could help Q2P in discerning more effective classification rules.

When exploiting both pre- and post-retrieval properties, the accuracy of Q2P obtains a boost in performance of +5% in accuracy with respect to the model using only pre-retrieval properties. The average accuracy is in this case 0.82 (min = 0.77, max = 0.90). The all-inclusive model achieves better results than the pre-retrieval one on 11 of the 12 subject systems (i.e., all but Mahout 0.8, where both models achieve 0.80 as accuracy). Thus, exploiting both pre- and post-retrieval properties leads *almost* always to a better prediction accuracy. Table V compares the accuracy of Q2P when using both pre- and post-retrieval properties against the three considered baselines.

Compared to these baselines, Q2P is the most accurate predictor on each of the 12 individual systems and overall when considering the average across all systems. As mentioned before, Q2P obtains a correct classification rate of 0.82 on average. In comparison, the optimistic constant model obtains a correct classification rate of only 0.40, the pessimistic constant model has a correct classification rate of 0.60, and the random model correctly classifies the queries in only 52% of the cases. The total percentage of errors obtained by Q2P is 18% (11% Type I + 7% Type II). This is much lower compared to 60% for the optimistic constant classifier, 40% of the pessimistic constant, and 48% of the random classifier. Among the baseline classifiers, the pessimistic classifier performs the best on average (0.60 accuracy), as well as for most of the systems (10 of 12). This is explained by the fact that there is a considerably higher number of low-quality

[4]Note that also when building models only exploiting pre- and post-retrieval models, highly correlating properties were discarded.

RIGHTSLINK

Table V. Q2P Compared with the Baselines: Corr = Correct, T I = Type I Errors, T II = Type II Errors

	Q2P			Optimistic Const.			Pessimistic Const.			Random		
System	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II
Adempiere	0.88	0.04	0.08	0.45	0.00	0.55	0.55	0.45	0.00	0.41	0.33	0.26
Nutch	0.84	0.06	0.10	0.50	0.00	0.50	0.50	0.50	0.00	0.46	0.28	0.26
ATunes	0.80	0.10	0.10	0.39	0.00	0.61	0.61	0.39	0.00	0.59	0.31	0.10
BookKeeper	0.78	0.14	0.09	0.54	0.00	0.46	0.46	0.54	0.00	0.58	0.23	0.19
Derby	0.82	0.11	0.07	0.26	0.00	0.74	0.74	0.26	0.00	0.48	0.15	0.37
Eclipse	0.90	0.08	0.02	0.29	0.00	0.71	0.71	0.29	0.00	0.61	0.25	0.14
FileZilla	0.81	0.14	0.05	0.22	0.00	0.78	0.78	0.22	0.00	0.57	0.07	0.36
JEdit	0.84	0.12	0.04	0.44	0.00	0.56	0.56	0.44	0.00	0.50	0.43	0.07
Mahout 0.4	0.78	0.10	0.12	0.46	0.00	0.54	0.54	0.46	0.00	0.46	0.06	0.48
Mahout 0.8	0.80	0.12	0.08	0.52	0.00	0.48	0.48	0.52	0.00	0.54	0.26	0.20
Tika	0.77	0.12	0.12	0.38	0.00	0.62	0.62	0.38	0.00	0.58	0.15	0.27
WinMerge	0.77	0.15	0.08	0.33	0.00	0.67	0.67	0.33	0.00	0.49	0.20	0.31
Average	**0.82**	**0.11**	**0.07**	**0.40**	**0.00**	**0.60**	**0.60**	**0.40**	**0.00**	**0.52**	**0.23**	**0.25**

Table VI. Percentage of Trees Generated by the Random Forest for the Concept Location Study in Which Each Query Property Is Used

Query Property	%Trees
CS	62%
SCS	48%
AvgSCQ	46%
SumVAR	45%
MaxVAR	43%
AvgIDF	42%
AvgVAR	41%
DevIDF	40%
Subquery Overlap	39%
Robusteness Score	39%
MedEntropy	35%
First Rank Range	34%
AvgPMI	33%
QS	33%
WIG	26%
DevEntropy	26%
NCQ	25%
Spatial Autocorrelation	25%
MaxEntropy	25%
Clustering Tendency	24%
MaxIDF	12%

queries compared to high-quality queries in the data sets used in the study. All these low-quality queries are therefore classified correctly by the pessimistic classifier, which leads to its higher accuracy compared to the rest of the baseline approaches.

Finally, to understand which features are used most by the Random Forest when classifying queries, Table VI sorts the considered query properties on the basis of the percentage of trees built by the Random Forest classifier used in Q2P (column %Trees). The total number of trees built in this study is 36,000 (300 trees in each Random Forest model $\times$ 10 folds $\times$ 12 subject systems). The first interesting observation is that all the 21 properties are kept in the prediction model after the correlation analysis

RIGHTSLINK

have been exploited. However, it is clear from the distribution in Table VI that some query properties are used more than others by the Random Forest, indicating that they might be the most useful in discriminating between high- and low-quality queries. Interestingly, the most used query properties are the pre-retrieval ones that monopolize the top eight positions in Table VI, that is, from CS (62% of the trees) down to DevIDF (40%). This is somewhat expected given the previously discussed results, highlighting that pre-retrieval properties ensure better classification performances, compared to the post-retrieval ones. The least used property is MaxIDF, yet it is still used in a non-negligible percentage of trees (12%).

Summary for RQ_2. Q2P achieves its best accuracy when exploiting both pre- and post-retrieval properties (0.82, on average). In its best configuration it outperforms all baselines, being much better than a random or a constant classifier.

5. IDENTIFYING HARD TO TRACE SOFTWARE ARTIFACTS

This section describes the application and evaluation of Q2P in the context of artifact traceability recovery. We first briefly describe a generic TR-based traceability link recovery process.

TR-based traceability link recovery aims at applying TR techniques to compare a set of source artifacts (e.g., classes from source code) used as queries against another set of artifacts (e.g., use cases) and rank the similarity of all possible pairs of artifacts. Pairs having high textual similarity are candidates to be linked (a.k.a., candidate links). The ranked list of candidate links is then analyzed by a software engineer, who can accept the tool suggestion by tracing a candidate link or classify the link as a false positive. Empirical studies have indicated that the list of candidate links contains a higher density of correct traceability links in the upper part of the list and a much lower density of such links in the bottom part of the list [De Lucia et al. 2007; Hayes et al. 2003; Cleland-Huang et al. 2005]. This means that in the lower part of the ranked list the effort required to discard false positives becomes much higher than the effort to validate correct links.

Generally, TR-based techniques work well (i.e., retrieve correct links on top of the ranked list) when the text used in the document corpus is consistent with that used in the query artifacts. However, in practice, there is often deviation between the text used in artifacts at various abstraction levels. Additionally, some artifacts are by nature less verbose (e.g., an interface or an abstract class) and most TR techniques would return a very low similarity value between them even if the two artifacts are related. As a result, such correct links will appear in the lower part of the list of candidate links, requiring high effort on the part of the software engineer performing traceability recovery. Our conjecture is that by using our approach we can identify artifacts that are potentially *hard-to-trace* when performing a TR-based traceability recovery process. This information can be useful to avoid wasting time in investigating irrelevant suggestions produced by a TR-engine. When such artifacts are identified, developers can use a different approach to recover links for these artifacts (e.g., manual identification of the links).

5.1. Study Design

The *goal* of this study is to assess the ability of Q2P to identify *hard-to-trace* artifacts when using a TR-based traceability recovery approach. Thus, the following research questions were formulated:

RIGHTSLINK

Table VII. Classification of the Quality of Candidate Link Lists Produced by Automated Methods

Classification	Recall	Precision
non-acceptable	<60%	<20%
acceptable	60%–69%	20%–29%
good	70%–79%	30%–49%
excellent	80%–100%	50%–100%

—$\mathbf{RQ}_3$: Are there query properties capturing the same information about the traceability level of software artifacts (e.g., hard-to-trace*vs* not hard-to-trace)?
—$\mathbf{RQ}_4$: How accurate is Q2P in identifying hard-to-trace artifacts for TR-based traceability link recovery and how does its accuracy compare to baseline classifiers?

To answer our research questions, we first need to define a criterion to discriminate *hard-to-trace* artifacts from the rest. To this aim, we adopt the evaluation criteria proposed by Hayes et al. [2006] to assess the quality of candidate link lists produced by automated methods. In particular, the authors define thresholds for recall and precision [Baeza-Yates and Ribeiro-Neto 1999], two widely used Information Retrieval (IR) measures, to classify the quality of the produced candidate link lists as *non-acceptable*, *acceptable*, *good*, and *excellent*.

Table VII reports the defined quality levels, reflecting the effort required by a developer performing the traceability recovery task to trace correct links [Hayes et al. 2006]. In particular, candidate link lists having *excellent* recall and precision values require low effort from the developer performing traceability link recovery. On the other hand, low recall and precision values (e.g., recall <60% and precision <20%) result in a high effort for the developer performing the traceability recovery task. In this situation, one might prefer to execute the task manually, without adopting any TR-based tool.

Since our goal is to verify if Q2P is able to identify *hard-to-trace* artifacts, we adopt the following classification criteria: If, when using the artifact as query it is possible to achieve recall of 60% together with a precision of at least 20%, then we consider the artifact traceability *acceptable*, otherwise it is considered *non-acceptable*, thus the artifact is *hard-to-trace*. Note that we consider only two values for the quality of the candidate link lists, that is, *hard-to-trace* and *not hard-to-trace*, since our goal is to give the developer a clear yes/no answer to the question of whether the list of candidate links is worth investigating. We are planning to investigate ranges to assess the quality of link lists in our future work.

To answer $\mathbf{RQ}_3$ we follow the same process adopted in our concept location study. In particular, we compute the Kendall rank correlation coefficient (i.e., Kendall's τ) [Daniel 1978] between all possible 378 pairs of measures to determine whether there are pairs exhibiting a strong correlation. Also in this case, for each pair of properties exhibiting a strong correlation (i.e., with a Kendall's $|\tau| \geq 0.6$), one has been randomly excluded from the classifier building in $\mathbf{RQ}_4$.

Concerning $\mathbf{RQ}_4$, we performed a 10-fold cross validation over the dataset, computing the accuracy of Q2P and the AUC of the built model when only relying on pre- and post-retrieval properties in isolation as well as when combining the two categories of properties. Also, we compare the accuracy of our approach with those achieved by the three baseline classifiers seen before: optimistic constant (always classify an artifact as *not hard-to-trace*), pessimistic constant (always classify the traceability of the artifacts as *hard-to-trace*), and the random classifier, which randomly assigns a classification (*not hard-to-trace* or *hard-to-trace*) to an artifact.

The misclassification rate of the experimented models has been evaluated in terms of Type I and Type II classification errors. In the traceability link recovery context, a Type I error occurs when the model wrongly classifies a *hard-to-trace* artifact as a

RIGHTSLINK

Table VIII. The Systems Used in the Traceability Study and Their Properties

System	Version	Language	#Classes	#Use Cases	#Links
eAnci	1.1	Java	57	139	558
eTour	1.5	Java	119	60	402
SMOS	1.0	Java	100	67	1,048
Total	—	—	276	266	2,008

not hard-to-trace one, while a Type II error is when the model wrongly classifies a *not hard-to-trace* artifact as a *hard-to-trace* one.

Finally, also in this case, we balance the training set at each iteration (i.e., for each of the 10 folds) by exploiting the SMOTE technique [Chawla et al. 2002]. Again, we never modify the test set.

5.2. Context

Table VIII shows the object systems used in our study. Each of these systems has been developed by a team of Master's students at the University of Salerno during the software engineering course in the context of an industrial internship. eAnci is a software system to manage municipalities, while eTour is an electronic touristic guide. SMOS is a software developed for high schools, which offers a set of functionalities aimed at simplifying the communications between the school and the students' parents.

For all the object systems, we have available the source code (i.e., Java classes) and use case artifacts, as well as the traceability matrices provided by the original developers establishing relationships among these two artifact categories. Information about the number of artifacts in each system and links existing among them is reported in Table VIII. In our study, we focus on the task of retrieving traceability links between source code classes (representing our queries) and use case artifacts (representing the document corpus). In particular, the following process has been adopted to classify each query artifact as *hard-to-trace* or as *not hard-to-trace*:

(1) The source code and the use cases have been subjected to a modified version of the text normalization process described in the study on concept location (see Section 4.2):
   (a) For use cases, any formatting or labeling template that was present in all use cases in identical arrangement was removed. (e.g., "Use case name" at the beginning of each eTour use case document, etc).
   (b) For splitting, the original identifiers were not kept as it is unlikely that specific identifier names will appear in use case documentation.
   (c) For stop words removal, a combination list of English and Italian stop words, as well as standard Java keywords, was utilized, as the text for these systems is primarily in Italian, but Java keywords and some English words are also present.
   (d) For stemming, the Snowball[5] Italian stemmer was used via a Weka wrapper implemented in Java,[6] since Italian was the language used in use cases and source code identifiers and comments.
(2) Each query artifact Q (i.e., each class) has been run on the document corpus (i.e., all use cases) using Lucene to retrieve the ranked list of candidate links for Q.
(3) Knowing the correct links for each artifact Q, we automatically evaluated it as *hard-to-trace* or as *not hard-to-trace* by analyzing the produced ranked list. In particular, we started from the top document in the ranked list and then traversed

[5]http://snowballstem.org/.
[6]https://weka.wikispaces.com/Stemmers.

RIGHTSLINK

Table IX. The Quality of the Query Artifacts Used in the Traceability Study

System	Total Number of Query Artifacts	Artifacts *not hard-to-trace*	Artifacts *hard-to-trace*
eAnci	55	21	34
eTour	95	60	35
SMOS	68	21	47
Total	218	102	116

```
1  package smos.application.userManagement;
2
3  import javax.servlet.http.HttpServlet;
4  import javax.servlet.http.HttpServletRequest;
5  import javax.servlet.http.HttpServletResponse;
6
7  /**Invalidate the session.
8   * @author Napolitano Vincenzo.*/
9  public class ServletLogout extends HttpServlet {
10
11         /* doPost method
12          * @param pRequest
13          * @param pResponse */
14         protected void doPost(HttpServletRequest pRequest,
15         HttpServletResponse pResponse)  {
16                 pRequest.getSession().invalidate();
17                 pResponse.sendRedirect(./index.html);
18         }
19 }
```

Listing 1. Example of a *hard-to-trace* class.

the list until 60% of recall was reached. At this point, we measured the precision. If the precision was higher than 20%, then we classified the artifact as *not hard-to-trace*; otherwise, we classified it as *hard-to-trace*. Note that this reflects the actual classification of the query artifacts, not the predicted one.

Source code classes not having any traceability link (as reported in the traceability matrix) to the target use case artifacts were excluded from our study, since it is not possible to evaluate in any way their traceability level. The final set of query (class) artifacts considered in our study is reported, together with their classification as "*not hard-to-trace* artifacts" and "*hard-to-trace* artifacts" in Table IX.

An example of a *hard-to-trace* class is the `ServletLogout` class from the SMOS system shown in Listing 1. `ServletLogout` is in charge of performing the logout of a user previously logged into the system. This operation is performed in lines 16 and 17 by (i) invalidating the user session and (ii) redirecting the user to the login page (i.e., `index.html`). Given the simplicity of this class, developers did not invest too much effort in explaining the functionality of this class through comments. In fact, the only comment containing meaningful information is *"Invalidate the session"* reported in line 7. Using this artifact as a query does not retrieve the use cases related to it (e.g., UserLogout), as it contains only a few, very generic, terms.

On the other hand, an example of a *not hard-to-trace* artifact is the `User` class from the SMOS system reported in Listing 2. The class is a Java bean modeling a user of the system, characterized by login, password, firstName, and lastName. Given the high frequency occurrence of meaningful terms representing the aim of this class (e.g., user, login, first name, last name, password), it is quite simple to trace all target artifacts (i.e., use cases) describing features concerning the management of users in the system,

RIGHTSLINK

```
package smos.bean;
/** Class used to model an user */
public class User{
  private String login, password, firstName, lastName;
  /** @return the user login*/
  public String getLogin() {
    return this.login;
  }
  /** Sets the user login
   * @param pLogin: the login to set*/
  public void setLogin(String pLogin) {
    this.login = pLogin;
  }
  /** @return the complete name of the user. */
  public String getName() {
    return this.lastName + " " + this.firstName;
  }
  /** @return the first name of the user */
  public String getFirstName() {
    return this.firstName;
  }
  /** Sets the first name of the user
   * @param pFirstName: the first name to set*/
  public void setFirstName(String pFirstName) {
    this.firstName = pFirstName;
  }
  /** @return the password of the user */
  public String getPassword() {
    return this.password;
  }
  /** Sets the password of the user
   * @param pPassword: the password to set*/
  public void setPassword(String pPassword) {
    this.password = pPassword;
  }
  /** @return the last name of the user*/
  public String getLastName() {
    return this.lastName;
  }
  /** Sets the last name of the user
   * @param pLastName: the last name to set*/
  public void setLastName(String pLastName) {
    this.lastName = pLastName;
  }
}
```

Listing 2. Example of a *not hard-to-trace* class.

like the *Registration* use case containing, in particular, the fields of the registration form a user needs to fill in (i.e., login, first name, last name, password).

5.3. Results

In this section, we provide answers to our research questions.

5.3.1. Are There Query Properties Capturing the Same Information about the Traceability Level of Software Artifacts (e.g., Hard-to-Tracevs Difficult-to-Trace)? Figure 4 depicts a heatmap reporting the Kendall's τ between all 378 pairs of 28 query properties considered in our approach. Remember that in the context of traceability recovery, the "queries" on which the properties have been computed are represented by the classes of the three subject systems. As previously done, we only report statistically significant correlations (i.e., those having a p-value < 0.05), showing a white square in correspondence of non-significant correlations (e.g., the one between MaxICTF and DevIDF).

RIGHTSLINK

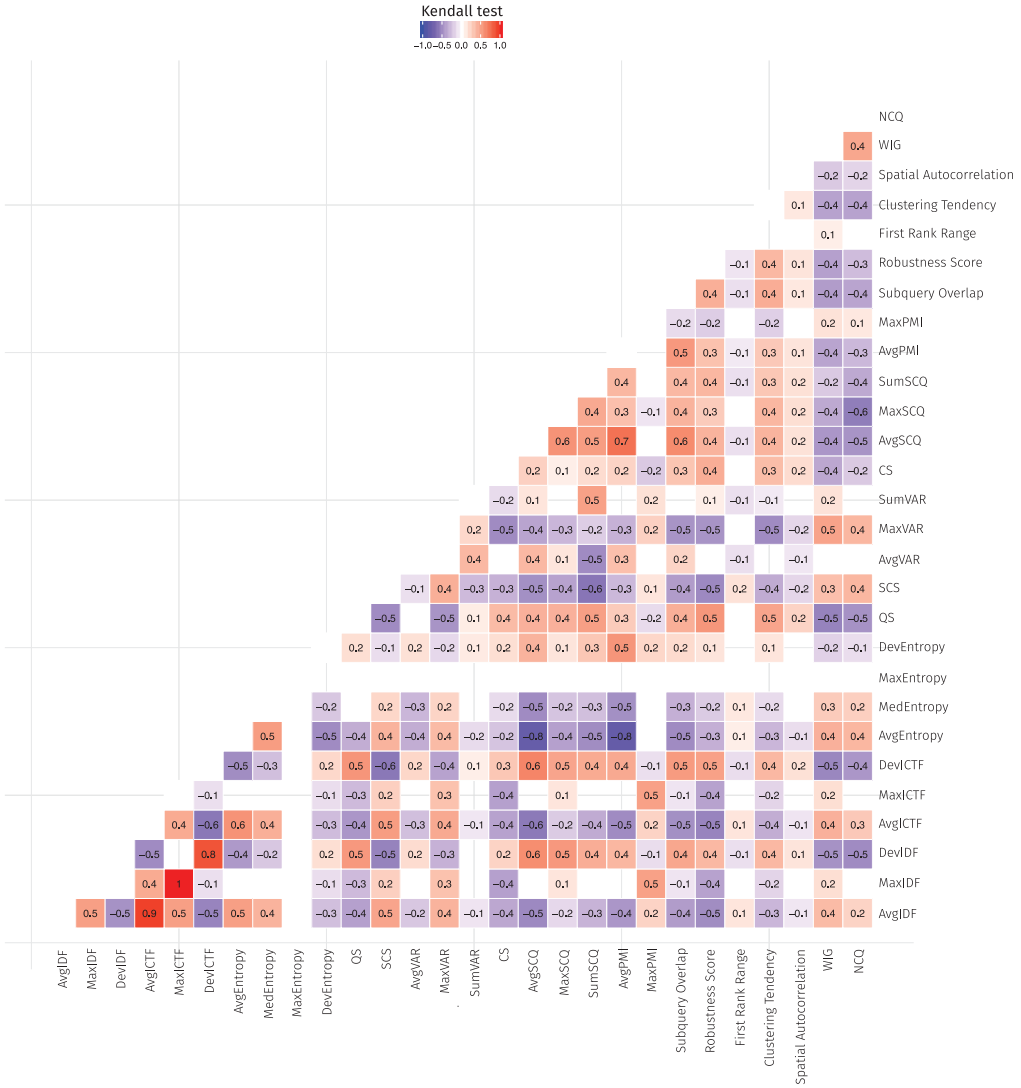

Fig. 4. Kendall τ between the 378 pairs of query properties.

The very strong correlation between the properties based on IDF and those based on ICTF is confirmed also in our traceability dataset. MaxICTF and MaxIDF exhibit a perfect positive correlation ($\tau = 1.0$), and the other two pairs of properties in this set (i.e., AvgICTF-AvgIDF and DevICTF-DevIDF) have a strong positive correlation ($\tau \geq 0.8$). From this point of view, we observed no difference with what found in our concept location dataset.

AvgPMI and AvgSCQ are strongly correlated ($\tau = 0.7$) in our traceability study. Also, AvgEntropy exhibits on this dataset several strong correlations (see the corresponding row in Figure 4). The strongest ones are with AvgSCQ and AvgPMI ($\tau = -0.8$). First, the negative sign of these correlations is expected, since hard-to-trace artifacts should be characterized by high values of AvgEntropy and low values for AvgSCQ and AvgPMI. Second, the specific queries (source code classes) and document corpus (use cases) in

RIGHTSLINK

which these three properties have been measured could have played a role in their correlation. Indeed, the strength of these correlations differed in our concept location study, suggesting that the dataset and the type of queries influence the metrics and their correlations.

The last strong correlation observed on this dataset is the one between MaxSCQ and NQC ($\tau = -0.6$). Such a correlation is the most interesting as well as difficult to explain. Note that:

(1) It is the only correlation we observed between a pre- (MaxSCQ) and a post-retrieval (NQC) property and, as can be seen in our Appendix, their definition and computation differ considerably.
(2) While for both properties a higher value should indicate a higher query quality, they are negatively correlated (i.e., when one increases the other decreases).

As stated before, our data suggest that the nature of queries and of the corpus documents can influence the correlations seen between metrics for a given dataset. In this case, we see a strong negative correlation in the traceability study, while the same metrics did not have a significant correlation in the concept location study. One possible explanation of the differences in correlations seen between our two studies is that while the queries in our concept location study are based on natural language text (bug report titles and descriptions), the queries in the traceability study are extracted from source code artifacts, which may affect the metrics, which were initially introduced for natural language queries [Carmel and Yom-Tov 2010]. Further dedicated research is required to fully characterize and understand these potential differences and their effect on metric correlations.

Finally, it is worth noting that for MaxEntropy it was not possible to compute any correlation index on this dataset. The reason is that MaxEntropy always equals 1 for each of the considered query artifacts (i.e., there is also at least a term in each class having entropy equals one). While such a result might look surprising, this is a result of the specific type of queries and documents in the considered corpus. In our implementation, the entropy for a query term t that is not present in the corpus is set to one by default, indicating an essentially negligible amount of information. This choice is dictated by the intuition that t is not discriminating, since it does not help to find relevant documents in the corpus. When using code classes as queries on a document corpus composed of use cases, this vocabulary mismatch is not unexpected, since some implementation-specific terms will not be found in high-level artifacts such as use cases.

Summary for RQ_3. In our traceability dataset, ICTF- and IDF-based properties strongly correlate, as already observed in the concept location dataset. Other strong correlations were observed among AvgEntropy, AvgSCQ, and AvgPMI. We also observed a correlation between MaxSCQ and NCQ. Given the results of this research question, we excluded from our prediction model on these systems the following metrics: AvgICTF, MaxICTF, DevICTF, MaxEntropy, AvgSCQ, AvgPMI, and NCQ. Thus, the model used in RQ_4 is built on top of the left 21 query properties.

5.3.2. How Accurate Is Our Approach in Identifying Hard-to-Trace Artifacts for TR-Based Traceability Recovery When Using within- and Cross-Project Training? Table X shows the accuracy of Q2P in classifying *hard-to-trace* and *not hard-to-trace* artifacts when exploiting (i) pre-retrieval properties only, (ii) post-retrieval properties only, and (iii) a combination of both pre- and post-retrieval mea sures.

RIGHTSLINK

Table X. Q2P When Using Pre-Retrieval Query Properties only, Post-Retrieval only, and All Query Properties: Corr = Correct, T I = Type I Errors, T II = Type II Errors

	Q2P - pre-retrieval			Q2P - post-retrieval			Q2P - all properties		
System	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II
eAnci	0.68	0.18	0.14	0.44	0.26	0.30	0.72	0.20	0.08
eTour	0.66	0.17	0.18	0.57	0.22	0.21	0.71	0.17	0.12
SMOS	0.77	0.15	0.08	0.57	0.23	0.20	0.80	0.12	0.08
Average	**0.70**	**0.17**	**0.13**	**0.53**	**0.24**	**0.24**	**0.74**	**0.16**	**0.10**

Table XI. Q2P Compared with the Baselines: Corr = Correct, T I = Type I Errors, T II = Type II Errors

	Q2P			Optimistic Const.			Pessimistic Const.			Random		
System	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II	Corr	T I	T II
eAnci	0.72	0.20	0.08	0.38	0.00	0.62	0.62	0.38	0.00	0.65	0.13	0.22
eTour	0.71	0.17	0.12	0.63	0.00	0.37	0.37	0.63	0.00	0.42	0.38	0.20
SMOS	0.80	0.12	0.08	0.30	0.00	0.70	0.70	0.30	0.00	0.41	0.21	0.38
Average	**0.74**	**0.16**	**0.10**	**0.44**	**0.00**	**0.56**	**0.56**	**0.44**	**0.00**	**0.49**	**0.24**	**0.27**

The trend observed in Table X mirrors the one previously discussed for the concept location study: Q2P achieves good performances when using pre-retrieval predictors only (average accuracy=0.70, min=0.66, max=0.77), while its accuracy strongly decreases when only relying on post-retrieval predictors (average accuracy=0.53, min=0.44, max=0.57). Note that in this latter case the performances of Q2P are very low and comparable to the ones of a random classifier. This confirms that, while using pre-retrieval properties only might be an option (e.g., for the application of Q2P to specific tasks in which it is important to reduce as much as possible the computational cost), post-retrieval properties alone are simply not an option.

When considering both pre- and post-retrieval measures, the accuracy of Q2P increases for all subject systems. In particular, we observe a +4% on eAnci, +5% on eTour, and +3% on SMOS (+4% on average). The average accuracy provided by Q2P when exploiting pre- and post-retrieval predictors is 0.74, with 16% of Type I and 10% of Type II errors.

Table XI compares the results achieved by Q2P to those of the baseline classifiers. As it can be noticed, the accuracy of Q2P is better on all object systems compared to all baseline classifiers. On average, the accuracy of our approach is 30% higher than that of the optimistic constant classifier, 18% higher than the pessimistic constant classifier, and 25% higher than the random classifier. Also, when comparing Q2P with the best of the baselines classifiers on each system, the differences are always rather large:

—eAnci: +7% with respect to the random classifier (0.72 vs. 0.65);
—eTour: +8% with respect to the optimistic constant (0.71 vs. 0.63);
—SMOS: +10% with respect to the pessimistic constant (0.80 vs. 0.70).

These results highlight the superiority of Q2P in correctly classifying query artifacts as *hard-to-trace* or *not hard-to-trace*, compared to the baselines.

In terms of misclassifications, as said before, Q2P misclassifies in 26% of cases (16% Type I + 10% Type II) vs. the 56% of the optimistic constant (all of Type II), the 44% of the pessimistic constant (all of Type I), and the 51% of the random classifier (24% Type I + 27% Type II).

Finally, as in the concept location study, Table XII sorts the considered query properties based on the percentage of trees built by the Random Forest classifier used in Q2P (column %Trees). The total number of trees built in this study is 9,000 (300 trees in each Random Forest model × 10 folds × 3 subject systems). The achieved results are consistent with those observed in the context of the concept location study. Specifically:

RIGHTSLINK

Table XII. Percentage of Trees Generated by the Random Forest for the Traceability Study in Which Each Query Property Is Used

Query Property	%Trees
CS	82%
SCS	66%
AvgIDF	58%
SumSCQ	58%
DevIDF	55%
SumVAR	52%
AvgEntropy	51%
AvgVAR	49%
WIG	44%
Robusteness Score	43%
First Rank Range	42%
DevEntropy	41%
MaxVAR	40%
Subquery Overlap	39%
MaxSCQ	38%
Clustering Tendency	36%
Spatial Autocorrelation	35%
QS	30%
MaxPMI	14%
MedEntropy	10%
MaxIDF	5%

—The pre-retrieval properties, again, monopolize the top eight positions, confirming their central role in Q2P.
—The top-two ranked properties (i.e., CS and SCS) are the same as in the concept location study.
—The MaxIDF confirms its minor impact in the built trees, as in concept location.

Summary for RQ$_4$. Confirming the results of our concept location study, Q2P performs the best (average accuracy = 0.74) when combining both pre- and post-retrieval properties. Also, it has a good average accuracy (0.70) when only relying on pre-retrieval properties. Q2P's accuracy in discriminating between *hard-to-trace* and *not hard-to-trace* artifacts is better than that achieved by all the considered baselines.

6. THREATS TO VALIDITY

In this section, we discuss the main threats [Yin 2003] that could affect the validity of our results.

Construct validity threats concern the relationship between theory and observation and are mainly related to imprecisions made when building the oracles used in both our studies. For the concept location study (Section 4), we considered as relevant methods for queries extracted from a bug report b_r, the set of methods that were modified to fix b_r, based on the patches attached to it in the online bug tracking systems. Concerning the traceability study (Section 5), we relied on the traceability matrices provided by the original developers of the three subject systems. Thus, in both cases we are confident about the correctness of the used oracle.

RIGHTSLINK

Also related to construct validity, there are possible threats derived from the experimenter bias. We paid attention to follow standard procedures (e.g., 10-fold validation) when assessing the accuracy of the proposed approach. Also, while the data gathering and analysis have been performed by one of the authors, two other authors double checked the complete procedure.

With respect to the *internal validity*, in our experimentation for concept location we automatically extracted the set of queries from the online bug tracking system of the object systems. Specifically, we extracted three different queries from the bug reports, one derived from the title of the bug report, one from the description of the bug, and one by combining the bug title and description. Such queries are approximations of actual user queries, but they are commonly used in concept location studies.

The *external validity* refers to the generalization of our findings. To address this threat, we selected a set of 12 software systems of different sizes, from diverse domains, implemented in two programming languages for our concept location study and three systems for the traceability link recovery study. A larger set of queries and more systems would clearly strengthen the results from this perspective. One threat to the external validity of our results is the fact that we used the results of only one TR engine to classify the queries as high quality or low quality. More precisely, we used Lucene, which is an implementation of the Vector Space Model (VSM) technique and is often used for concept location and traceability link recovery applications. Since several other TR methods have been previously used to support concept location and traceability link recovery, further experimentation is needed to analyze whether the proposed approach works well also with other TR methods. We have, however, no reasons to expect a significantly different performance from Q2P using a different TR engine.

In the context of the traceability recovery study, we used three systems developed by a team of Master's students at the University of Salerno during a Software Engineering course in the context of an industrial internship. While such projects may not be fully representative of industrial software systems, they have been previously used in studies performed in the context of traceability recovery [Oliveto et al. 2010; Gethers et al. 2011; Bavota et al. 2013b; Panichella et al. 2013; Diaz et al. 2013; De Lucia et al. 2013; Capobianco et al. 2013], as well as in studies evaluating other types of software engineering recommenders [Panichella et al. 2013; Bavota et al. 2013a, 2014; Moreno et al. 2014].

The last threat to external validity is related to the fact that we only evaluated the proposed approach for the tasks of TR-based concept location and TR-based traceability link recovery. Thus, we cannot (and do not) generalize the results to other SE tasks.

Finally, *conclusion validity* refers to the degree to which conclusions reached about relationships between variables are justified. In our case study, we only draw conclusions referring to the use of different classifiers, which we support with evidence in the form of classification correctness and Type I and II errors. These measures are widely used in software engineering to evaluate predictor models [Agresti 2002]. In addition, we analyze and compare the overall classification accuracy of the proposed approach taking into account the number of queries correctly and wrongly classified and also perform a qualitative analysis of the errors.

7. CONCLUSION AND FUTURE WORK

Q2P is a novel approach for predicting the quality of queries in the context of TR-supported software engineering tasks. The proposed approach is based on using supervised learning (i.e., classification trees) and 28 query properties. Our empirical evaluation in the context of concept location showed that the proposed approach was able to correctly classify 82% of queries on average, strongly outperforming several baseline approaches. Also, the evaluation on traceability link recovery showed that

RIGHTSLINK

Q2P was able to correctly classify artifacts as hard-to-trace or not in 74% of cases, outperforming any of the considered baselines.

We conjecture that Q2P can save the developer time and effort in concept location, as he or she is notified early when a query is unlikely to lead to satisfactory results and would likely need reformulation. As for traceability link recovery, Q2P will indicate the developer when it may be better to perform the task manually and also point out artifacts that may need redocumentation. Overall, the performances of Q2P were satisfactory and provided us with a set of lessons learned.

Lesson 1. *Pre- and post-retrieval properties capture different signals in the data*. This has been shown in both studies during our correlation analysis. Indeed, while we observed several strong correlations among pre-retrieval properties, we only observed one strong correlation between a pre- (MaxSCQ) and a post-retrieval (NCQ) property. Also, such a correlation has only been observed on the queries belonging to the traceability dataset (i.e., query represented by source code classes) and not confirmed in our concept-location study. Besides correlation analysis, improvement in Q2P's accuracy obtained when combining pre- and post-retrieval properties also indicates their ability to describe different aspects of the query quality.

Lesson 2. *Pre-retrieval properties alone ensure a good classification accuracy, whereas post-retrieval properties do not*. Both our studies indicate that Q2P provides good performance when using only pre-retrieval properties (average accuracy is 0.77 and 0.70 in the concept location and in the traceability study, respectively). Such a "lighter version" of Q2P can be adopted in applications where Q2P's response time should be reduced as much as possible. Indeed, the post-retrieval properties are more expensive to compute than pre-retrieval properties. Moreover, pre-retrieval properties can be computed without running the query at all, which allows for real-time prediction prior to the submission of the query. Differently, the Q2P accuracy is very low when only relying on post-retrieval properties. However, combining both families of properties ensures the best classification accuracy.

Lesson 3. *The Q2P performances and the behavior of the query quality properties substantially change when working with natural language queries (see concept location study) and with code-based queries (see traceability study)*. Q2P provided a higher classification accuracy in the concept location study (0.82) than in the traceability study (0.74). Also, the correlation analysis performed on the two datasets showed some differences in terms of the relationships between the query quality properties (e.g., strong correlations observed on the traceability dataset were not present in the concept location dataset). These differences might be related to the specific and different "nature" of the queries employed in the two studies (natural-language-based vs. code-based). However, further investigation is needed from the research community on this point to better understand how to maximize the query quality prediction performances for TR-based SE tasks.

Our work is the first to present a technique to predict the quality of queries in the context of TR-based SE tasks. Still, we feel that Q2P represents only a first step in tackling this problem. More research in this direction is needed for a full understanding of the query quality phenomenon. Our future research agenda includes the application of Q2P to support other TR-based SE tasks and a deep study of the query properties and their characteristics on a large amount of data. Also, we plan to experiment with predicting the quality of a query at a finer granularity level than just high or low quality (e.g., quality on a scale from 1 to 10) and to investigate the impact that the type of bug report author (e.g., end user or developer) can have on the results. Finally, it is worth noting that Q2P is semantics agnostic (i.e., it does not consider the semantics of the query). Including such information in Q2P (e.g., by trying to understand what the developer is looking for) could help in further improving its accuracy, and it is a research direction we plan to pursue in future work.

RIGHTSLINK

8. APPENDIX

Table XIII. The 21 Pre-Retrieval Query Measures

Measure	Description	Formula
	Specificity	
AvgIDF	Average of the Inverse Document Frequency (idf)[7] values over all query terms	$\frac{1}{\|Q\|}\sum_{q\in Q} idf(q)$
MaxIDF	Maximum of the Inverse Document Frequency (idf) values over all query terms	$\max_{q\in Q}(idf(q))$
DevIDF	The standard deviation of the Inverse Document Frequency (idf) values over all query terms	$\sqrt{\frac{1}{\|Q\|}\sum_{q\in Q}(idf(q)-avgIDF)}$
AvgICTF	Average Inverse Collection Term Frequency (ictf)[8] values over all query terms	$\frac{1}{\|Q\|}\sum_{q\in Q} ictf(q)$
MaxICTF	Maximum Inverse Collection Term Frequency (ictf) values over all query terms	$\max_{q\in Q}(ictf(q))$
DevICTF	The standard deviation of the Inverse Collection Term Frequency (ictf) values over all query terms	$\sqrt{\frac{1}{\|Q\|}\sum_{q\in Q}(ictf(q)-avgICTF)}$
AvgEntropy	Average entropy[9] values over all query terms	$\frac{1}{\|Q\|}\sum_{q\in Q} entropy(q)$
MedEntropy	Median entropy values over all query terms	$\underset{q\in Q}{median}(entropy(q))$
MaxEntropy	Maximum entropy values over all query terms	$\max_{q\in Q}(entropy(q))$
DevEntropy	The standard deviation of the entropy values over all query terms	$\sqrt{\frac{1}{\|Q\|}\sum_{q\in Q}(entropy(q)-avgEntropy)}$
Query Scope (QS)	The percentage of documents in the collection containing at least one of the query terms	$\frac{\|\bigcup_{q\in Q} D_q\|}{\|D\|}$
Simplified Clarity Score (SCS)	The Kullback-Leiber divergence of the query language model from the collection language model[10]	$\sum_{q\in Q} p_q(Q)\log\left(\frac{p_q(Q)}{p_q(D)}\right)$
	Coherency	
AvgVAR	Average of the variances of the query term weights over the documents containing the query term (VAR),[11] over all query terms	$\frac{1}{\|Q\|}\sum_{q\in Q} VAR(q)$

(Continued)

[7] $idf(t) = log(\frac{|D|}{|D_t|})$.

[8] $ictf(t) = log(\frac{|D|}{tf(t,D)})$.

[9] $entropy(t) = \sum_{d\in D_t} \frac{tf(t,d)}{tf(t,D)} \cdot \log_{|D|} \frac{tf(t,d)}{tf(t,D)}$.

[10] $p_t(X) = \frac{tf(t,X)}{|X|}$.

[11] $VAR(t) = \sqrt{\frac{\sum_{d\in D_t}(w(t,d)-\bar{w}_t)^2}{df(t)}}$, where $w(t,d) = \frac{1}{|d|} log(1+tf(t,d)) \cdot idf(t)$, and $\bar{w}_t = \frac{1}{|D_t|}\sum_{d\in D_t} w(t,d)$.

RIGHTSLINK

Table XIII. Continued

MaxVAR	Maximum of the variances of the query term weights over the documents containing the query term (VAR), over all query terms	$\max_{q \in Q}(VAR(q))$
SumVAR	Sum of the variances of the query term weights over the documents containing the query term (VAR), over all query terms	$\sum_{q \in Q} VAR(q)$
Coherence Score (CS)	The average of the pairwise similarity between all pairs of documents containing one of the query terms among all documents in the corpus	$\frac{1}{\lvert Q \rvert} \sum_{q \in Q} \left(\frac{\sum_{(d_i, d_j) \in D_q} sim(d_i, d_j)}{\lvert D_q \rvert \cdot (\lvert D_q \rvert - 1)} \right)$
	Similarity	
AvgSCQ	The average of the collection-query similarity (SCQ)[12] over all query terms	$\frac{1}{\lvert Q \rvert} \sum_{q \in Q} SCQ(q)$
MaxSCQ	The maximum of the collection-query similarity (SCQ) over all query terms	$\max_{q \in Q}(SCQ(q))$
SumSCQ	The sum of the collection-query similarity (SCQ) over all query terms	$\sum_{q \in Q} SCQ(q)$
	Term Relatedness	
AvgPMI	Average Pointwise Mutual Information (PMI)[13] over all pairs of terms in the query	$\frac{2(\lvert Q \rvert - 1)!}{(\lvert Q \rvert)!} \sum_{q_1, q_2 \in Q} PMI(q_1, q_2)$
MaxPMI	Maximum Pointwise Mutual Information (PMI) over all pairs of terms in the query	$\max_{q_1, q_2 \in Q} (PMI(q_1, q_2))$

Q, the set of query terms; q, a term in the query; D, the set of documents in the collection;

D_t, the set of documents containing term t; d, a document in the document collection D; $tf(t, D)$, the frequency of term t in all docs;

$tf(t, d)$, the frequency of term t in d; $tf(t, Q)$, the frequency of term t in the query; $sim(d_i, d_j)$, the cosine similarity between the vector-space representations of d_i and d_j

[12] $SCQ(t) = (1 + log(tf(t, D))) \cdot idf(t)$.

[13] $PMI(t_1, t_2) = log \frac{p_{t_1, t_2}(D)}{p_{t_1}(D) \cdot p_{t_2}(D)}$, where $p_{t_1, t_2}(D) = |D_{t_1} \bigcap D_{t_2}|/|D|$, and $p_t(D) = |D_t|/|D|$.

RIGHTSLINK

Table XIV. The Seven Post-Retrieval Query Measures

Measure	Description	Formula or Algorithm
	Robustness	
Subquery Overlap	Captures the extent of the overlap between the result set retrieved by the entire query and the result sets retrieved by individual query terms. The lower the standard deviation, the better the query.	(1) Run the original query q, and obtain the result list R (2) Run each individual query term q_t in the original query as a separate query and obtain the result list R_t. (3) For each individual query term q_t, compute the overlap between the first k ($k = 10$) documents in R and the first k documents in R_t (i.e., number of documents found in both result lists) (4) The overall score of the query is the standard deviation of the values of the overlap considered for each term in the query
Robustness Score	The terms' weights in the top relevant documents are slightly perturbed and the resulting documents are re-ranked. The correlation between the initial rank and that after modification is considered. The higher the robustness score, the better the query.	(1) Run the original query q, and obtain the result list R (2) Take the top 50 documents in R and consider them as ranked list L (3) For each document d in L, get a perturbed document d' from d in the following way: (a) All terms t from the corpus that do not appear in document d, will not be included in d neither (b) All terms t from the corpus that appear in document d with frequency f, but do not appear in the query will appear in document d' with the same frequency f (c) Each term t that appears in d with frequency f and appears also in the query q will appear also in d, but with a frequency f', which is a random number obtained from a Poisson distribution $P(\lambda)$ with $\lambda = f$ (4) The new 50 documents obtained are ranked according to the query q, resulting in a second ranked list L', where each document corresponds to a document in L (5) Compute the Spearman rank correlation between the positions of the 50 documents in L and the positions of their corresponding perturbed documents in L' and record the correlation obtained (6) Repeat steps 8 to 8 100 times, and the final robustness score is the average Spearman correlation between the 100 runs

(Continued)

RIGHTSLINK

Table XIV. Continued

First Rank Change	Captures the probability of a document found on the first position in the list of results to still remain on the first position after a perturbation is applied to it. The higher the score, the better the query.	(1) Run the original query q, and obtain the result list R (2) Take the top 50 documents in R and consider them as ranked list L (3) For each document d in L, get a perturbed document d' from d in the following way: (a) All terms t from the corpus that do not appear in document d, will not be included in d neither (b) All terms t from the corpus that appear in document d with frequency f, but do not appear in the query will appear in document d' with the same frequency f. (c) Each term t that appears in d with frequency f and appears also in the query q will appear also in d, but with a frequency f'', which is a random number obtained from a Poisson distribution $P(\lambda)$ with $\lambda = f$ (4) The new 50 documents obtained are ranked according to the query q, resulting in a second ranked list L', where each document corresponds to a document in L (5) Record a 1 if the top ranked document in L is also the top ranked document (after perturbation) in L, and record 0 otherwise (6) Repeat steps 8 to 8 100 times, and the final score is the sum of the values (0 or 1) obtained in step 8 for all the 100 runs
Clustering Tendency	Measures the cohesion of the top retrieved documents as the textual similarity between them. The higher the clustering tendency the better the query	$CT = Mean\left(\frac{Sim_{query}(d_{mp}, d_{nn}\|q)}{Sim_{query}(p_{sp}, d_{mp}\|q)}\right) * \frac{1}{T}\sum_{i=1}^{T}(x_i - y_i)$, where: q, the query p_{sp}, the sampled point. that is, a randomly chose document from the corpus, which does not appear in the top 100 documents d_{mp}, the marked point, that is, the document, inside the top 100 documents in the ranked list, with largest similarity with the sampled point d_{nn}, the nearest neighbor of the marked point within the top 100 documents in the ranked list x_i, the maximum weight for a term i across the top 100 retrieved documents y_i, the minimum weight for a term i across the top 100 retrieved documents The mean in the CT formula is computed for 100 randomly sampled points (i.e., the similarity formulas are computed 100 times, each time with a different randomly sampled point). $Sim_{query}(d_i, d_j\|q) = \frac{\sum_{k=1}^{T} d_{ik}d_{jk}}{\sqrt{\sum_{k=1}^{T} d_{ik}^2 \sum_{k=1}^{T} d_{jk}^2}} * \frac{\sum_{k=1}^{T} c_k q_k}{\sqrt{\sum_{k=1}^{T} c_k^2 \sum_{k=1}^{T} q_k^2}}$, where: d_i and d_j, the two documents T, the number of unique terms in the collection q, the query (with weight q_k for term k), the weight of a term in the query or a document is its $tf - idf$ c, the vector of terms common to both d_i and d_j with weights ck being the average of d_{ik} and d_{jk}

(Continued)

RIGHTSLINK

Table XIV. Continued

Spatial Autocorrelation	Changes the retrieval-scores of each top relevant document as the average of the scores of its most similar documents. Then, the linear correlation of the new scores with original ones is used. The higher the spatial autocorrelation the better the query.	(1) Run the original query q, and obtain the result list R (2) Take the top 50 documents in R and consider them as ranked list L (3) For each document d in L, compute the cosine similarity between d and the rest of the documents in L, using $tf - idf$ as the weight of the terms in the document vectors (4) Among the documents in L, select the 5 documents that are most similar to d according to the cosine similarity (5) Let s be the score of document d in L. Assign a new score to d, which is the average score of the 5 most similar documents to it according to the cosine similarity (6) Perform the above steps for each document d in L (7) The Pearson correlation between the original scores of the documents in L and the derived scores of those documents represents the index of spatial autocorrelation
Score distribution		
Weighted Information Gain (WIG)	Measures the divergence between the mean retrieval score of top-ranked documents and that of the entire corpus. The hypothesis is that the more similar these documents are to the query, with respect to the query similarity exhibited by a general non-relevant document (i.e., the corpus), the more effective the retrieval. The higher the weighted information gain, the better the query.	$$WIG(q) = \frac{1}{k} \sum_{d \in D_q^k} \sum_{t \in q} \lambda(t) log \frac{Pr(t\|d)}{Pr(t\|D)},$$ where: q, query t, a term in the query q D, set of all documents in corpus D_q, the set of documents in the result set to query q D_{qk}, the top k documents in the result list k, the number of top documents to consider $\|q\|$, number of terms in the query q $\lambda(t) = 1/\sqrt{\|q\|}$
Normalized Query Commitment (NQC)	Measures the standard deviation of retrieval scores in D_q^k, normalized by the score of the whole collection. The higher NCQ, the better the query.	$$NCQ = \frac{\sqrt{\frac{1}{k} \sum_{d \in D_q^k} (Score(d) - \mu)^2}}{Score(D_q)},$$ where: k, the number of top documents to consider. Best performance was obtained with k = 100 D_q^k, The top k documents from the result list returned in response to query q $Score(d)$, the score obtained by document d in D_q^k D_q, the set of all results returned in response to query q $Score(D_q)$, the sum of the scores of all the documents in the result list returned by the IR technique $\mu = \frac{1}{k} \sum_{d \in D_q^k} Score(d)$ NCQ, represents the normalized standard deviation of the retrieval scores in D_q^k

RIGHTSLINK

REFERENCES

A. Agresti. 2002. *Categorical Data Analysis*. Wiley-Interscience.

G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. 2000. Information retrieval models for recovering traceability links between code and documentation. In *Proceedings of 16th IEEE International Conference on Software Maintenance*. IEEE CS Press, San Jose, CA, 40–51.

G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. 2002. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.* 28, 10 (2002), 970–983.

J. Anvik and G. Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. Softw. Eng. Methodol.* 20, 3 (Aug. 2011), Article 10, 35 pages. DOI:http://dx.doi.org/10.1145/2000791.2000794

V. Arnaoudova, S. Haiduc, A. Marcus, and G. Antoniol. 2015. The use of text retrieval and natural language processing in software engineering. In *Proceedings of the 37th IEEE/ACM International Conference on Software Engineering (ICSE'15), Vol. 2*. 949–950.

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

S. Bajracharya and C. Lopes. 2012. Analyzing and mining a code search engine usage log. *Emp. Softw. Eng.* 17, 4–5 (Aug. 2012), 424–466.

G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto. 2013a. Using structural and semantic measures to improve software modularization. *Emp. Softw. Eng.* 18, 5 (2013).

G. Bavota, A. De Lucia, and R. Oliveto. 2011. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *J. Syst. Softw.* 84 (Mar. 2011), 397–414. Issue 3.

G. Bavota, A. De Lucia, R. Oliveto, A. Panichella, F. Ricci, and G. Tortora. 2013b. The role of artefact corpus in LSI-based traceability recovery. In *Proceedings'13 of the 2013 International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*. 83–89.

G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia. 2014. Improving software modularization via automated analysis of latent topics and dependencies. *ACM Trans. Softw. Eng. Methodol.* 23, 1 (Feb. 2014), 4:1–4:33.

G. Bavota, S. Haiduc, R. Oliveto, A. Macus, and A. De Lucia. 2016. (2016). Retreived from http://www.cs.fsu.edu/serene/queryquality.

L. Biggers, C. Bocovich, R. Capshaw, B. Eddy, L. Etzkorn, and N. Kraft. 2012. Configuring latent dirichlet allocation based feature location. *Emp. Softw. Eng.* 19, 3 (Aug. 2012), 465–500.

D. Binkley and D. Lawrie. 2010a. Development: Information retrieval applications. *Encyclopedia of Software Engineering*, P. A. Laplante (Ed.). Taylor & Francis, 231–242.

D. Binkley and D. Lawrie. 2010b. Maintenance and evolution: Information retrieval applications. *Encyclopedia of Software Engineering* (2010), 454–463.

D. Binkley and D. Lawrie. 2014. Learning to rank improves IR in SE. In *Proceedings of the 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*. 441–445.

M. Borg, P. Runeson, and A. Ard. 2014. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Emp. Softw. Eng.* 19 (2014), 1565–1616. Issue 6.

L. Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

G. Canfora and L. Cerulo. 2005. Impact analysis by mining software and change request repositories. In *Proceedings of 11th IEEE International Symposium on Software Metrics*. IEEE CS Press, 20–29.

G. Canfora and L. Cerulo. 2006. Supporting change request assignment in open source development. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC'06)*. ACM, New York, NY, 1767–1772. DOI:http://dx.doi.org/10.1145/1141277.1141693

G. Capobianco, A. De Lucia, R. Oliveto, A. Panichella, and S. Panichella. 2013. Improving IR-based traceability recovery via noun-based indexing of software artifacts. *J. Softw.: Evol. Process* 25, 7 (2013), 743–762.

D. Carmel and E. Yom-Tov. 2010. *Estimating the Query Difficulty for Information Retrieval*. Morgan and Claypool Publishers.

O. Chaparro and A. Marcus. 2016. On the reduction of verbose queries in text retrieval based software maintenance. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE'16)*. ACM, New York, NY, 716–718. DOI:http://dx.doi.org/10.1145/2889160.2892647

N. Chawla, K. Bowyer, L. Hall, and P. Kegelmeyer. 2002. SMOTE: Synthetic minority over-sampling technique. *J. Artif. Intell. Res.* 16, 1 (June 2002), 321–357.

J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. 2010. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the International Conference on Software Engineering*. 155–164.

RIGHTSLINK

J. Cleland-Huang, R. Settimi, C. Duan, and X. Zou. 2005. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proceedings of 13th IEEE International Requirements Engineering Conference*. IEEE CS Press, 135–144.

J. Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Lawrence Earlbaum Associates.

K. Damevski, D. Shepherd, and L. Pollock. 2016. A field study of how developers locate features in source code. *Emp. Softw. Eng.* 21, 2 (Apr. 2016), 724–747. DOI: http://dx.doi.org/10.1007/s10664-015-9373-9

V. Dang, M. Bendersky, and B. Croft. 2010. Learning to rank query reformulations. In *Proceedings of the Special Interest Group of Information Retrieval (SIGIR'10)*. 807–808.

W. W. Daniel. 1978. *Applied Nonparametric Statistics*. Houghton Mifflin.

B. De Alwis and G. Murphy. 2008. Answering conceptual queries with ferret. In *Proceedings of the 2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 21–30.

A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora. 2007. Recovering traceability links in software artefact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.* 16, 4 (2007).

A. De Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. 2011. Information retrieval methods for automated traceability recovery. *Softw. Syst. Traceabil.* (2011), 71–98.

A. De Lucia, R. Oliveto, and P. Sgueglia. 2006. Incremental approach and user feedbacks: A silver bullet for traceability recovery. In *Proceedings of the International Conference on Software Maintenance*. 299–309.

A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella. 2013. Applying a smoothing filter to improve IR-based traceability recovery processes: An empirical investigation. *Inf. Softw. Technol.* 55, 4 (2013), 741–754.

D. Diaz, G. Bavota, A. Marcus, R. Oliveto, S. Takahashi, and A. De Lucia. 2013. Using code ownership to improve IR-based traceability link recovery. In *Proceedings of the 2013 IEEE 21st International Conference on Program Comprehension (ICPC'13)*. 123–132.

B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. 2013. Feature location in source code: A taxonomy and survey. *J. Softw.: Evol. Process* 25, 1 (2013), 53–95.

B. Eddy and N. Kraft. 2014. Using structured queries for source code search. In *Proceedings of the 30th IEEE International Conference on Software Maintenance and Evolution*. 431–435.

G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. 1987. The vocabulary problem in human-system communication: An analysis and a solution. *Commun. ACM* 30, 11 (1987), 964–971.

G. Gay, S. Haiduc, A. Marcus, and T. Menzies. 2009. On the use of relevance feedback in IR-based concept location. In *Proceedings of the International Conference on Software Maintenance*. 351–360.

X. Ge, D. Shepherd, K. Damevski, and E. Murphy-Hill. 2014. How developers use multi-recommendation system in local code search. In *Proceedings of the Conference on Visual Language and Human-Centric Computing*. Retrieved from http://people.engr.ncsu.edu/ermurph3/papers/vlhcc14.pdf.

M. Gethers, R. Oliveto, D. Poshyvanyk, and A. De Lucia. 2011. On integrating orthogonal information retrieval methods to improve traceability recovery. In *Proc. of International Conference on Software Maintenance (ICSM'11)*. 133–142.

M. Gibiec, A. Czauderna, and J. Cleland-Huang. 2010. Towards mining replacement queries for hard-to-retrieve traces. In *Proceedings of the International Conference on Automated Software Engineering*. 245–254.

J. Grivolla, P. Jourlin, and R. De Mori. 2005. Automatic classification of queries by expected retrieval performance. In *Proceedings of the ACM Special Interest Group on Information Retrieval*.

S. Haiduc, G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto. 2012. Evaluating the specificity of text retrieval queries to support software engineering tasks. In *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering, NIER Track*. 1273–1276.

S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. De Lucia, and T. Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *Proceedings of the 35th International Conference on Software Engineering (ICSE'13)*. 842–851.

S. Haiduc, G. Bavota, R. Oliveto, A. De Lucia, and A. Marcus. 2012. Automatic query performance assessment during the retrieval of software artifacts. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE'12)*. 90–99.

J. H. Hayes, A. Dekhtyar, and J. Osborne. 2003. Improving requirements tracing via information retrieval. In *Proceedings of 11th IEEE International Requirements Engineering Conference*. IEEE CS Press, 138–147.

J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. 2006. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Softw. Eng.* 32, 1 (2006), 4–19.

RIGHTSLINK

E. Hill, L. Pollock, and K. Vijay-Shanker. 2009. Automatically capturing source code context of NL-queries for software maintenance and reuse. In *Proceedings of the International Conference on Software Engineering*.

E. Hill, M. Roldan-vega, J. Fails, and G. Mallet. 2014. NL-based query refinement and contextualized code search results: A user study. In *Proceedings of the IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE'14)*. 3443.

B. J. Jansen, A. Spink, and T. Saracevic. 2000. Real life, real users, and real needs: a study and analysis of user queries on the web. *Inf. Process. Manage.* 36, 2 (2000), 207–227.

C. Jensen and W. Scacchi. 2006. Discovering, modeling, and reenacting open source software development processes. *New Trends Softw. Process Model.* 18 (2006), 1–20.

S. Jiang, L. Shen, X. Peng, Z. Lv, and W. Zhao. 2015. Understanding developers' natural language queries with interactive clarification. In *Proceedings of the 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER'15)*. IEEE, 13–22.

Y. Kim, J. Seo, and B. Croft. 2011. Automatic boolean query suggestion for professional search. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'11)*. ACM, New York, NY, 825–834. DOI:http://dx.doi.org/10.1145/2009916.2010026

M. Kimmig, M. Monperrus, and M. Mezini. 2011. Querying source code with natural language. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. 376379.

A. Kuhn, S. Ducasse, and T. Gîrba. 2007. Semantic clustering: Identifying topics in source code. *Inf. Softw. Technol.* 49, 3 (2007), 230–243.

Giridhar Kumaran and Vitor R. Carvalho. 2009. Reducing long queries using query quality predictors. In *Proceedings of the Special Interest Group of Information Retrieval (SIGIR'09)*. 564–571.

K. L. Kwok. 1995. A network approach to probabilistic information retrieval. *ACM Trans. Inf. Syst.* 13, 3 (July 1995), 324–353. DOI:http://dx.doi.org/10.1145/203052.203067

O. Lemos, A. de Paula, H. Sajnani, and C. Lopes. 2015. Can the use of types and query expansion help improve large-scale code search?. In *Proceedings of Source Code Analysis and Manipulation (SCAM'15)*. 41–50.

O. Lemos, A. de Paula, F. Zanichelli, and C. Lopes. 2014. Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th Working Conference on Mining Software Repositories (MSR'14)*. 212–221.

D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich. 2007. Feature location via information retrieval based filtering of a single scenario execution trace. In *Proceedings of the International Conference on Automated Software Engineering*. 234–243.

S. Lohar, S. Amornborvornwong, A. Zisman, and J. Cleland-Huang. 2013. Improving trace accuracy through data-driven configuration and composition of tracing features. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 378–388.

F. Lv, H. Zhang, J. Lou, S. Wang, D. Zhang, and J. Zhao. 2015. CodeHow: Effective code search based on API understanding and extended boolean model (E). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE'15)*. IEEE, 260–270.

A. Marcus and S. Haiduc. 2013. Text retrieval approaches for concept location in source code. *Lecture Notes in Computer Science*, A. De Lucia and F. Ferrucci (Eds.). Vol. 7171 (2013), Springer, 126–158. Issue Software Engineering.

A. Marcus and J. I. Maletic. 2001. Identification of high-level concept clones in source code. In *Proceedings of 16th IEEE International Conference on Automated Software Engineering*. IEEE CS Press, 107–114.

A. Marcus and J. I. Maletic. 2003. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of 25th International Conference on Software Engineering*. IEEE CS Press, 125–135.

A. Marcus, D. Poshyvanyk, and R. Ferenc. 2008. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Trans. Softw. Eng.* 34, 2 (2008), 287–300.

A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. 2004. An information retrieval approach to concept location in source code. In *Proceedings of the Working Conference on Reverse Engineering*. 214–223.

K. Matjaz and I. Kononenk. 2002. Reliable classifications with machine learning. In *Proceedings of the 13th European Conference on Machine Learning*. 219–231.

L. Moreno, G. Bavota, S. Haiduc, M. Di Penta, R. Oliveto, B. Russo, and A. Marcus. 2015. Query-based configuration of text retrieval solutions for software engineering tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, 567–578. DOI:http://dx.doi.org/10.1145/2786805.2786859

RIGHTSLINK

L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora. 2014. Automatic generation of release notes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE-22'14)*. 484–495.

S. Nair, J. L. de la Vara, and S. Sen. 2013. A review of traceability research at the requirements engineering conference RE@21. In *Proceedings of the 21st IEEE International Requirements Engineering Conference (RE'13)*. 222–229. DOI: http://dx.doi.org/10.1109/RE.2013.6636722

A. Nguyen, T. Nguyen, J. Al-Kofahi, H. Nguyen, and T. Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE'11)*. IEEE, 263–272.

R. Oliveto, M. Gethers, D. Poshyvanyk, and A. De Lucia. 2010. On the equivalence of information retrieval methods for automated traceability link recovery. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension (ICPC'10)*. 68–71.

A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. 2013. How to effectively use topic models for software engineering tasks? An approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*. 522–531.

A. Panichella, B. Dit, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia. 2016. Parameterizing and assembling IR-based solutions for SE tasks using genetic algorithms. In *Proceedings of the 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER'16)*, Vol. 1. IEEE, 314–325.

A. Panichella, C. McMillan, E. Moritz, D. Palmieri, R. Oliveto, D. Poshyvanyk, and A. De Lucia. 2013. When and how using structural information to improve IR-based traceability recovery. In *Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering (CSMR'13)*. 199–208.

M. Petrenko, V. Rajlich, and R. Vanciu. 2008. Partial domain comprehension in software evolution and maintenance. In *Proceedings of the International Conference on Program Comprehension*. 13–22.

M. F. Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

D. Poshyvanyk and A. Marcus. 2006. The conceptual coupling metrics for object-oriented systems. In *Proceedings of 22nd IEEE International Conference on Software Maintenance*. IEEE CS Press, 469–478.

S. Rao and A. Kak. 2011. Retrieval from software libraries for bug localization: A comparative study of generic and composite text models. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. 43–52.

S. Rao, H. Medeiros, and A. Kak. 2015. Comparing incremental latent semantic analysis algorithms for efficient retrieval from software libraries for bug localization. *SIGSOFT Softw. Eng. Not.* 40, 1 (Feb. 2015), 1–8.

M. Roldan-Vega, G. Mallet, E. Hill, and J. Fails. 2013. CONQUER: A tool for NL-based query refinement and contextualizing code search results. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*. 512–515.

G. Scanniello, A. Marcus, and D. Pascale. 2015. Link analysis algorithms for static concept location: An empirical assessment. *Emp. Softw. Eng.* 20, 6 (2015), 1666–1720.

D. Shepherd, Z. Fry, E. Gibson, L. Pollock, and K. Vijay-Shanker. 2007. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of 6th International Conference on Aspect Oriented Software Development*. ACM Press, 212–224.

C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. 1999. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, Vol. 33. ACM, 6–12.

B. Sisman and A. Kak. 2013. Assisting code search with automatic query reformulation for bug localization. In *Proceedings of the IEEE International Working Conference on Mining Software Repositories (MSR'13)*. 309318.

A. Spink, D. Wolfram, M. B. J. Jansen, and T. Saracevic. 2001. Searching the web: The public and their queries. *J. Am. Soc. Inf. Sci. Technol.* 52, 3 (2001), 226–234.

G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker. 2008. Identifying word relations in software: A comparative study of semantic similarity tools. In *Proceedings of the International Conference on Program Comprehension*. 123–132.

J. Starke, C. Luce, and J. Sillito. 2009. Searching and skimming: An exploratory study. In *Proceedings of the International Conference on Software Maintenance*. 157–166.

S. W. Thomas, M. Nagappan, D. Blostein, and A. E. Hassan. 2013. The impact of classifier configuration and classifier combination on bug localization. *IEEE Trans. Softw. Eng.* 39, 10 (Oct. 2013), 1427–1443.

A. Vercoustre, J. Pehcevski, and V. Naumovski. 2008. Topic diffculty prediction in entity ranking. In *Proceedings of the 7th International Workshop of the Initiative for the Evaluation of XML Retrieval*. 280–291.

E. Voorhees. 2005. The TREC robust retrieval track. *ACM SIGIR Forum* 39 (2005), 11–20. Issue 1.

RIGHTSLINK

S. Wang, D. Lo, and J. Lawall. 2014. Compositional vector space models for improved bug localization. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME'14)*. 171–180.

I. Witten and E. Frank. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc.

J. Yang and L. Tan. 2012. Inferring semantically related words from software context. In *Proceedings of 9th Working Conference on Mining Software Repositories*. 161–170.

X. Ye, R. Bunescu, and C. Liu. 2014. Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, New York, NY, 689–699.

R. K. Yin. 2003. *Case Study Research: Design and Methods* (3rd ed.). SAGE Publications.

E. Yom-Tov, S. Fine, and D. C. A. Darlow. 2005. Learning to estimate query difficulty. In *Proceedings of the ACM Special Interest Group on Information Retrieval*. 512–519.

Y. Zou, T. Ye, Y. Lu, J. Mylopoulos, and L. Zhang. 2015. Learning to rank for question-oriented software text retrieval (T). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1–11.

Received October 2015; revised February 2017; accepted March 2017

RIGHTSLINK