Improving Recall in Code Search by Indexing Similar Codes under Proper Terms

The recall of a code search engine usually depends on the indexing mechanism and query formulation techniques. The reason is that proper indexing and query understanding help retrieving relevant code snippets that satisfy user needs [4]. Most of the code search engines employ information retrieval approach for indexing source code [6] i.e. terms are generated from keywords extracted from source code, and index is constructed based on these terms. If two code fragments perform the same task but have different keywords, these will be indexed against different terms. When a user query matches one of the code fragments, search engine should retrieve both code fragments as these codes perform similar functionality. However, traditional code search engines cannot do this because both code fragments contain different keywords. Hence, the recall is reduced for not retrieving such relevant code fragments and the performance of the code search engine is decreased [7].

To improve recall of a code search engine, similar code fragments should be indexed under the same terms. As a result, if a user query matches these terms, similar code fragments will be retrieved. However, it is challenging to automatically and efficiently determine that two code fragments are identical or similar [8]. Although identical code fragments can be detected through keywords matching [9], detecting feature wise similar code blocks, is a challenging task. Another challenge is to select proper terms that best represent similar code fragments. A code fragment may have keywords which may not express its intent (i.e. implemented feature) properly. Indexing based on the keywords, reduces matching probability between user query and these keywords. The reason is that user query defines functionality but the extracted keywords do not express the feature properly.

Researchers have proposed various techniques to improve performance of code search engines where recall is considered as one of the performance indicators. Initially, Keyword Based Code Search (KBCS) [10] [11] [12] was introduced where source code is considered as plaintext document [13]. Source codes are indexed based on the terms generated from the code and searching is performed on the index. As this approach does not consider similarity between source code having different keywords, it could not retrieve more relevant codes. In order to satisfy the need to find and reuse existing software components, Interface Driven Code Search (IDCS) [14] [15] was proposed. Using this approach, user can define his required component interface as query and relevant components are retrieved based on that. Usually, IDCS uses method name, parameters, class name etc. for searching relevant codes. However, two or more code fragments may have different interfaces but perform the same task. IDCS considers that these code fragments are different due to having different interfaces. Thus, the recall of the approach is reduced.

Reusing existing components often takes a significant amount of time to understand and adapt in the development context. This is because, components are developed for reusing under a particular development context which may not suit in another context or require modifications. To automatically find and adapt reusable components Test Driven Code Search (TDCS) [16] and Semantic Based Code Search (SBCS) [17] were proposed. TDCS is an effective technique in terms of precision which employs test cases on the retrieved codes [17]. In this approach, most test cases fail not only for functional requirements mismatch but also for syntactic mismatch of the interface definition [17]. Semantically relevant code fragments could not be retrieved for this reason and hence, the performance is degraded in terms of recall. A SBCS technique was proposed which uses test cases to obtain semantic information

[18]. It searches in the popular search engines like google code [24], krugle [25] etc. based on the user query and provides the semantically matched code fragments. It depends on the search results of the code search engines and these engines use KBCS [19] that has low recall as stated earlier.

Objectives

In order to improve the recall of a code search engine, similar codes should be identified and indexed under proper terms. Existing code search engines adopts KBCS which cannot retrieve code fragments that contain different keywords but implement similar feature. Thus, the recall of the code search engines is reduced for not being able to retrieve more relevant code fragments. However, if feature wise similar codes are detected and clustered with proper terms, the number of retrieved relevant codes will be increased. Ultimately the recall of the code search engines will be improved. So, this leads to the following research question.

How to increase recall by indexing similar codes under proper terms?

Two sub-questions are associated with this question, which are outlined as follows.

1. How to detect similar codes that implement the same feature?
    a. Building source code repository containing different projects
    b. Extracting projects and methods of each project to create a collection of methods
    c. Matching the signature and body of each method with other methods to identify similar methods
    d. Constructing method clusters where each cluster will contain feature-wise identical methods
2. How to select appropriate terms for similar codes during indexing?
    a. Gathering keywords from the body and signature of each method in a cluster obtained from 1 (d).
    b. Calculating term-document frequency (tf-idf) for each keyword where document frequency is the number of methods that contain the keyword, and term frequency is the number of occurrences of the keyword in whole method collection
    c. Selecting the top scored keyword(s) as term(s) to represent the similar codes

Research Methodology

The intent of the research is to improve the recall of code search engine so that more relevant code fragments will be retrieved against the user query. To attain this, the first task is to conduct literature survey. This will assist to understand the related work in this domain, as shown in the first activity of Table 1. Researches have been carried out on code search and several techniques have been proposed to improve the performance of code search engines like KBCS, TDCS, IDCS etc. However, literature survey is an ongoing process and it will be continued throughout the thesis.

The 2nd activity is to focus on the specific areas related to the research. Similar code fragments identification, cluster of similar codes construction, appropriate term(s) selection, these are the areas that have been identified initially for attaining the research goal. These areas should be searched and studied exhaustively.

To increase the recall of a code search engine, source codes should be indexed properly with appropriate terms that best match the user query. For this, feature wise similar code fragments identification would be the first step during index construction. Cluster creation based on the similarity among code fragments and proper terms selection for each cluster would be the next step for source code indexing.

After that, the proposed technique needs to be compared with existing approaches to evaluate recall in retrieving relevant codes. If such characteristics are found which make the proposed technique to produce less recall in contrast with the existing techniques then iterative refinement of the approach will be done.

Three types of publications will be produced within the research period. At the end of the each major step, a technical report summarizing the findings will be produced. Significant achievements will be published and presented in international conferences (for example ICSE, ASE - IEEE/ACM, SIGSOFT, SCAM, APSEC, ICSEA). A thesis for the fulfillment of the Master of Science in Software Engineering (MSSE) by the Institute of Information Technology (IIT) will be compiled at the end of the research.

The major steps of the research are summarized below in Table 1:

Table 1: Research Steps

Step No.	Title of Activity	Activity Description & Relation to Research Question
1.	**Literature Survey**	• Building the background of the research. This will be an ongoing task. • Writing research proposal. • Presenting initial ideas in front of the evaluation body and taking some feedbacks if required.
2.	**Understanding Code Search, Source Code Indexing Technique, Code Searching Mechanism**	• Intensifying the focus on more specific area. This task will cover the similar or related work to our area of interest exhaustively. • Reporting on survey.
3.	**Improving Recall in Code Search**	• Algorithm needs to be developed to increase recall in code search. • Evaluating the algorithm against existing techniques. • Producing a technical report and publishing a conference paper.
4.	**Technical Report Writing**	• At the end of each major step, one technical report will be published
5.	**Publications & Presentation (s)**	• Review reports and research papers are being prepared for international conferences like ICSE, ASE2016, SCAM, APSEC, ICSEA and IEEE Conferences on Software Engineering 2016 etc.
6.	**Thesis Compilation**	• For the fulfillment of the master's a thesis will be compiled at the end.