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]. Usually, terms are generated from keywords extracted from source code and index is constructed based on the 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 could not 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 reduced due to low recall [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 code blocks that implement similar feature but contain different keywords, 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 are different from the feature.

Researchers have proposed various techniques to improve performance of code search engines. 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 modification. 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 for not functional requirements mismatch but 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. Several techniques like parseweb [20], starthcona [21], xsnippet [22], and prospector [23] were proposed to find method invocation sequences to construct a specific object type from another object types. Here input-output type is given by user as query. However, only input-output type cannot represent the user need at all. For example, the user looks for a function that will convert a binary string into hexadecimal string and user query is defined as <string,string>. These techniques show many irrelevant results for this query because the query cannot properly express the intent of the conversion.

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

1. How to detect similar codes that implement same feature?
2. How to select appropriate terms for similar codes during indexing?