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

The recall of a code search engine that is, number of retrieved relevant codes 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 [1]. Most of the code search engines employ information retrieval approach for indexing source code [2]. The approach generates terms from keywords extracted from source code, and constructs index based on these terms. Following this approach, if two code fragments perform the same task but have different keywords, these will be indexed against different terms. For this reason, when a user query matches one of the code fragments, traditional code search engines cannot retrieve both code fragments. Thus, the recall of existing code search engines is reduced for not retrieving such relevant code fragments and the performance of these engines is then decreased [3].

To improve recall of a code search engine, similar code fragments should be indexed under the same terms. However, it is challenging to automatically and efficiently determine that two code fragments are identical or similar [4]. Although identical code fragments can be detected through keywords matching [5], detecting feature wise similar code blocks, is difficult. The reason is that automatically perceiving the intent of a code block is still a research challenge [6]. Another challenge is to select proper terms that best represent similar code fragments. For example, two methods have name “x” and “sort” respectively and perform bubble sort. Here, two keywords (“x” and “sort”) are available where “sort” is more relevant than “x”. However, automatically deciding that “sort” is the better keyword to represent these methods is a challenging task. Again, a code fragment may have keywords which may not express its intent (that is, implemented feature) properly. Indexing based on these keywords, reduces matching probability between user query and these keywords. This is because user query defines functionality but the extracted keywords do not express the feature properly. So, instead of using these keywords, more meaningful terms need to be selected that best match the user query.

Researchers have proposed various techniques to improve performance of code search engines where recall is considered as one of the performance indicators. These techniques can be broadly classified into four types like Keyword Based Code Search (KBCS), Interface Driven Code Search (IDCS), Test Driven Code Search (TDCS), and Semantic Based Code Search (SBCS). Initially, KBCS [2] [7] [8] was introduced where source code is considered as plaintext document [2]. 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, IDCS [9] [10] [11] [12] was proposed. Using this approach, user can define the required component interface as query and relevant components are retrieved based on that. Usually, for searching relevant codes, IDCS uses method name, parameters, class name etc. 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 may require modifications. To automatically find and adapt reusable components, TDCS [13] and SBCS [14] were proposed. TDCS is an effective technique in terms of precision which employs test cases on the retrieved codes [15]. In this approach, most test cases fail not only for functional requirements mismatch but also for syntactic mismatch of the interface definition [15]. For this reason, semantically relevant code fragments could not be retrieved and hence, the performance is degraded in terms of recall. A SBCS technique was proposed which uses test cases to obtain semantic information [14]. For a given user query, it searches in the popular search engines like google code [16], krugle [17] etc. and provides the semantically matched code fragments. SBCS depends on the search results of the code search engines and these engines use KBCS [14] that has low recall like all the approaches mentioned above.

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. As a result, 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?

   To answer this sub-question, following steps may be adopted:

   a. A source code repository containing different projects needs to be built on which searching will be performed
   b. Projects and methods of each project are required to be extracted to provide method level searching facility

c. The signature and body of each method with other methods need to be matched to identify similar methods
d. It is required to construct method clusters based on the similarity among methods to retrieve more relevant code fragments

2. How to select appropriate terms for similar codes during indexing?

The steps outlined below may be followed to deal with this sub-question.

a. Keywords from the body and signature of each method in a cluster need to be gathered to identify candidate terms
b. Term frequency- inverse document frequency (tf-idf) needs to be calculated for each keyword to select proper terms. Here, document frequency is the number of methods that contain the keyword, inverse document frequency is the logarithmic ratio of document frequency and total number of documents, and term frequency is the number of occurrences of the keyword in whole method collection
c. The top scored keyword(s) may be selected as term(s) to represent the similar codes

Research Methodology

The intent of this 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, a list of activities, and description and relation of each activity to the research question is shown in Table 1. The first activity is to conduct literature survey. This will assist to understand the related work in this domain. 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. On the other hand, literature survey is an ongoing process and it will be continued throughout the thesis period.

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 the technique in retrieving relevant codes. If such characteristics are found which cause the proposed

technique to produce less recall in contrast with the existing techniques, theory refinement and necessary experimental setup adjustment will be performed.

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. • Implementing existing techniques for comparative analysis.
4.	**Creating Test Bed and Evaluating The Proposed Technique**	• Selecting standard experimental datasets. • Preparing experimental environment for running the proposed technique and existing approaches. • Evaluating the proposed technique against existing approaches.
5.	**Technical Report Writing, Publications and Presentation(s)**	• At the end of each major step, one technical report will be published. • 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.

Research Timeline

Table 2: Research Timeline

Title of Activity	Timeline 1 unit time = one month																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1. Literature Survey	←	—	—	—	—	—	—	—	—	—	—	—	—	—	→			
2. Understanding Code Search, Source Code Indexing Technique, Code Searching Mechanism		←	—	—	—	→												
3. Improving Recall in Code Search						←	—	—	—	→								
4. Creating Test Bed and Evaluating The Proposed Technique											←	—	→					
5. Technical Report Writing, Publications and Presentations				←	—	→				←	—	→				←	—	→
6. Thesis Compilation													←	—	—	—	—	→

Rational for the Research

Searching is one of the most commonly performed task in software development. Developers spend around 16% of total software development time for searching sample code or reusable component [18]. Usually, sample codes are searched to gain insight about an API. On the other hand, reusable components are searched to utilize these components that have already been developed. For both purposes, it is required to provide relevant results as much as possible by the existing code search engines. However, due to not considering feature wise similarity among code fragments, traditional code search engines fails to retrieve more code fragments that are

relevant. As a result, developers are deprived of getting additional code snippets which may be more relevant to them. For not getting relevant sample codes or reusable components, developers have to put extra time and effort to understand the usage of an API or to develop a component from scratch. This induces additional time and cost in software development which is identified in this research.

In order to speed up software development and reduce cost, existing search engines should satisfy developers' needs by providing relevant code fragments as much as possible. However, the recall of current code search engines decreases for not retrieving feature wise similar codes. So, to improve the recall of these code search engines, our proposed technique checks feature wise similarity among code fragments and selects proper terms to represent similar code snippets. Thus, many relevant code snippets will be retrieved against a query and developers will get more codes to understand an API or reuse existing software components. This will assist in reducing time and effort in software development.

Now-a-days, software has become one of the major industries in Bangladesh. Many software companies are developing quality software to meet the needs not only in Bangladesh but also in other countries throughout the world. According to the vision 2021, current income from Business Process Outsourcing (BPO) is 100 million dollars and government's target is to earn $1 billion by exporting quality software in future [20]. In order to fulfill this target, software companies need to increase the development speed and reduce cost. One of the important factors to decrease development time and cost is to reuse existing code snippets as much as possible [19]. For this, developers use current code search engines which have low recall due to not retrieving feature wise similar codes. On the other hand, our proposed technique will increase recall of the code search engines by retrieving these relevant code fragments. As a result, developers will be able to reuse more codes which will then make software development faster. This is actually how this research will assist to attain the goal of vision 2021 and make Digital Bangladesh.

References

[1] Prieto-Diaz, R. (1991). Implementing faceted classification for software reuse. Communications of the ACM, 34(5), 88-97.

[2] Sindhgatta, R. (2006, May). Using an information retrieval system to retrieve source code samples. In Proceedings of the 28th international conference on Software engineering (pp. 905-908). ACM.

[3] Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval (Vol. 1, No. 1, p. 496). Cambridge: Cambridge university press.

[4] Smith, R., & Horwitz, S. (2009, March). Detecting and measuring similarity in code clones. In *Proceedings of the International Workshop on Software Clones (IWSC)*.

[5] Kamiya, T., Kusumoto, S., & Inoue, K. (2002). CCFinder: a multilinguistic token-based code clone detection system for large scale source code.*Software Engineering, IEEE Transactions on*, *28*(7), 654-670.

[6] Linstead, E., Rigor, P., Bajracharya, S., Lopes, C., & Baldi, P. (2007, November). Mining concepts from code with probabilistic topic models. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (pp. 461-464). ACM.

[7] Hummel, O., Janjic, W., & Atkinson, C. (2008). Code conjurer: Pulling reusable software out of thin air. Software, IEEE, 25(5), 45-52.

[8] Lemos, O. A. L., de Paula, A. C., Sajnani, H., & Lopes, C. V. (2015, September). Can the use of types and query expansion help improve large-scale code search?. In Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on (pp. 41-50). IEEE.

[9] Thummalapenta, S., & Xie, T. (2007, November). Parseweb: a programmer assistant for reusing open source code on the web. In Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering (pp. 204-213). ACM.

[10] Holmes, R., Walker, R. J., & Murphy, G. C. (2005, September). Strathcona example recommendation tool. In ACM SIGSOFT Software Engineering Notes (Vol. 30, No. 5, pp. 237-240). ACM.

[11] Mandelin, D., Xu, L., Bodík, R., & Kimelman, D. (2005). Jungloid mining: helping to navigate the API jungle. ACM SIGPLAN Notices, 40(6), 48-61.

[12] Sahavechaphan, N., & Claypool, K. (2006). Xsnippet: mining for sample code. ACM Sigplan Notices, 41(10), 413-430.

[13] Lazzarini Lemos, O. A., Bajracharya, S. K., & Ossher, J. (2007, October). CodeGenie:: a tool for test-driven source code search. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion (pp. 917-918). ACM.

[14] Reiss, S. P. (2009, May). Semantics-based code search. In Proceedings of the 31st International Conference on Software Engineering (pp. 243-253). IEEE Computer Society.

[15] Janjic, W., & Atkinson, C. (2012, June). Leveraging software search and reuse with automated software adaptation. In Search-Driven Development-Users, Infrastructure, Tools and Evaluation (SUITE), 2012 ICSE Workshop on (pp. 23-26). IEEE.

[16] Google Code Search Engine. http://www.google.com/codesearch.

[17] Krugle Code Search Engine. http://www.krugle.com/

[18] Sim, S. E. (2013). Finding source code on the web for remix and reuse (pp. 139-165). R. E. Gallardo-Valencia (Ed.). Springer.

[19] Ye, Y., & Fischer, G. (2002, May). Supporting reuse by delivering task-relevant and personalized information. In Proceedings of the 24th international conference on Software engineering (pp. 513-523). ACM.

[20] http://www.albd.org/index.php/en/updates/news/3257-bangladesh-s-it-export-will-exceed-garments-export-sajeeb-wazed