securityandtrust.lu

Security Design and Validation Research Group: Augmenting and Structuring User Queries to Support Efficient Free-Form Code Search

Raphael Sirres, Tegawendé F. Bissyandé, Dongsun Kim, David Lo,
Jacques Klein and Yves Le Traon

Interdisciplinary Centre for Security, Reliability and Trust
University of Luxembourg

TR-SNT-2017-2

ISBN-13: 978-2-87971-159-1

July 07, 2017

Version 1.0

UNIVERSITÉ DU LUXEMBOURG

Noname manuscript No.
(will be inserted by the editor)

Augmenting and Structuring User Queries to Support Efficient Free-Form Code Search

Raphael Sirres · Tegawendé F. Bissyandé · Dongsun Kim · David Lo · Jacques Klein · Yves Le Traon

Received: date / Accepted: date

Abstract Source code terms such as method names and variable types are often different from conceptual words mentioned in a search query. This vocabulary mismatch problem can make code search inefficient. In this paper, we present COde voCABUlary (CoCaBu), an approach to resolving the vocabulary mismatch problem when dealing with free-form code search queries. Our approach leverages common developer questions and the associated expert answers to augment user queries with the relevant, but missing, structural code entities in order to improve the performance of matching relevant code examples within large code repositories. To instantiate this approach, we build GitSearch, a code search engine, on top of `GitHub` and `Stack Overflow` Q&A data. We evaluate GitSearch in several dimensions to demonstrate that (1) its code search results are correct with respect to user-accepted answers; (2) the results are qualitatively better than those of existing Internet-scale *code search engines*; (3) our engine is competitive against *web search engines*, such as Google, in helping users complete solve programming tasks; and (4) GitSearch provides code examples that are acceptable or interesting to the community as answers for `Stack Overflow` questions.

Keywords Code search · GitHub · Free-form search · Query augmentation · StackOverflow · Vocabulary mismatch

1 Introduction

Code search is an important activity in software development since developers are regularly searching [40] for code examples dealing with diverse programming concepts, APIs, and specific platform peculiarities. Such examples can indeed help them practice programming against a library and platform, or they can immediately be used for inspiration in software development tasks. Because contemporary programmers often

R. Sirres, T. F. Bissyandé, D. Kim, J. Klein, and Y. Le Traon
SnT, University of Luxembourg
E-mail: {firstname.lastname}@uni.lu

D. Lo
Singapore Management University
E-mail: davidlo@smu.edu.sg

implement most of program elements (e.g., classes and methods) based on existing programs already written by other programmers [33], an effective code search engine is a critical factor for programming productivity.

Open source project hosting platforms, such as GitHub, SourceForge, and BitBucket now offer an opportunity for students, researchers and developers to access real-world software projects for improving their work. It is, however, challenging to locate relevant source code due to the enormous size of existing code repositories. For instance, as of August 2015, GitHub is hosting more than 25 millions private and public code repositories[1]. To help developers search for source code, several Internet-scale code search engines [16], such as OpenHub [2] and Codota [3] have been proposed. The advantage of these engines is that users can express their queries in a list of keywords (i.e., free-form queries) rather than specific program elements such as API classes and methods.

Unfortunately, these Internet-scale code search engines have an accuracy issue since they treat source code as natural language documents. Source code, however, is written in a programming language while query terms are typically expressed in natural language. As a result, searching source code with query keywords in natural language often leads to irrelevant and low quality search results unless the keywords exactly correspond to program elements. According to Hoffmann *et al.* [23], however, around 64% of programmer web queries for code are merely descriptive but do not contain actual names of APIs, packages, types, etc.

As in any search engine, the terms in a code search query must be mapped with an index built from the code. Unfortunately, the construction of such an index as well as the mapping process are challenging since “no single word can be chosen to describe a programming concept in the best way” [15]. This is known in the literature as the vocabulary mismatch problem: user search queries frequently mismatch a majority of the relevant documents [15,20,45,46]. This problem occurs in various software engineering research work such as retrieving regulatory codes in product requirement specifications [12], identifying bug files based on bug reports [37], and searching code examples [20–22].

The vocabulary mismatch problem is further exacerbated in code search engines where the source code may be poorly documented or may use non explicit names for variables and method names [26]. To work around the translation issue between the query terms and the relevant code, one can leverage a developer community. Actually, developers often resort to web-based resources such as blogs, tutorial pages and Q&A sites. Stack Overflow is one of such leading discussion platforms, which has gained popularity among software developers. In Stack Overflow, an answer to a question is typically short texts accompanied by code snippets that demonstrate a solution to a given development task or the usage of a particular functionality in a library or framework. Stack Overflow provides social mechanisms to assess and improve the quality of posts that leads implicitly to high quality source code snippets.

While code snippets found in Q&A sites certainly accelerate the software development process, they fail to explore the potential of large code repositories. Typically, those code snippets are manually crafted by developers rather than being actual examples from source code repositories. Thus, snippets often omit context information (e.g., variable types and initialization values) that might be necessary to understand interactions with other relevant components. On the other hand, actual examples in source code repositories can provide different views on how a single functionality can

[1] https://github.com/about/press (verified 14.08.2015)

be implemented by different APIs. Source code repositories also contain concrete code that demonstrates the interaction between various modules and APIs of interest. Besides, usually, in Q&A sites, an acceptable answer only exists when the question, or a very similar one, has been asked before. Otherwise, the questioner must wait for other experienced developers to provide answers.

Our work focuses on building an approach to automatically expanding developer code search queries. Specifically, we aim at translating free-form queries to augment them with relevant program elements. To augment a user query, we consider first finding similar (in terms of natural language words) queries for which we have some sketched answers. Then we can collect from these answers some important code keywords. Finally, such code keywords are simply used to enrich the user's initial free-form terms. This query expansion is effective in retrieving relevant code search results even when the user has not provided in his query terms essential information such as API names.

Contributions We propose a novel approach to augmenting user queries in a free-form code search scenario. This approach aims at improving the quality of code examples returned by Internet-scale code search engines by building a COde voCABUlary (CoCaBu). The originality of CoCaBu is that it addresses the vocabulary mismatch problem, by expanding/enriching/re-targeting a user's free-form query, building on similar questions in Q&A sites so that a code search engine can find highly relevant code in source code repositories.

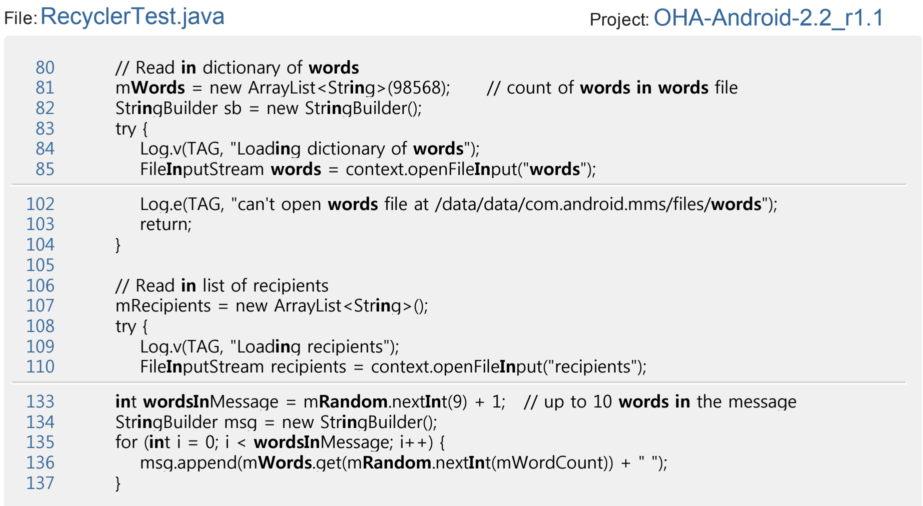
Overall, this paper makes the following contributions:

- CoCaBu **approach to the vocabulary mismatch problem:** We propose a technique for finding relevant code with free-form query terms that describe programming tasks, with no a-priori knowledge on the API keywords to search for. In this regard, we differ from several state-of-the-art techniques, which perform by searching relevant usage examples of APIs that the user can already list as relevant for his task [10,25,31,35].
- GitSearch **free-form search engine for GitHub:** We instantiate the CoCaBu approach based on indices of Java files built from `GitHub` and Q&A posts from `Stack Overflow` to find the most relevant source code examples for developer queries.
- **Empirical user evaluation**: We present the evaluation results implying that GitSearch accurately extends user queries to produce correct (i.e., relevant) results. Comparison with popular code search engines further shows that GitSearch is more effective in returning acceptable code search results. In addition, Comparison against web search engines indicates that GitSearch is a competitive alternative. Finally, via a live study, we show that users on Q&A sites may find GitSearch's real code examples acceptable as answers to developer questions.

The remainder of this paper is organized as follows. Section 2 motivates our work further, listing some limitations in the state-of-the-art and introducing the key ideas behind our approach. Section 3 then overviews the CoCaBu approach. We provide evaluation results in Section 4 and discuss related work in Section 5. Finally, Section 6 concludes the paper.

2 Motivation

The literature contains a large body of approaches that attempt to solve the vocabulary mismatch problem. They either 1) use a *controlled vocabulary* [27] maintained by



```
        // Read in dictionary of words
        mWords = new ArrayList<String>(98568);      // count of words in words file
        StringBuilder sb = new StringBuilder();
        try {
            Log.v(TAG, "Loading dictionary of words");
            FileInputStream words = context.openFileInput("words");
```

```
            Log.e(TAG, "can't open words file at /data/data/com.android.mms/files/words");
            return;
        }

        // Read in list of recipients
        mRecipients = new ArrayList<String>();
        try {
            Log.v(TAG, "Loading recipients");
            FileInputStream recipients = context.openFileInput("recipients");
```

```
        int wordsInMessage = mRandom.nextInt(9) + 1;   // up to 10 words in the message
        StringBuilder msg = new StringBuilder();
        for (int i = 0; i < wordsInMessage; i++) {
            msg.append(mWords.get(mRandom.nextInt(mWordCount)) + " ");
        }
```

Fig. 1: Top result provided by OpenHub for the free-form code search query "*Generating random words in Java?*"

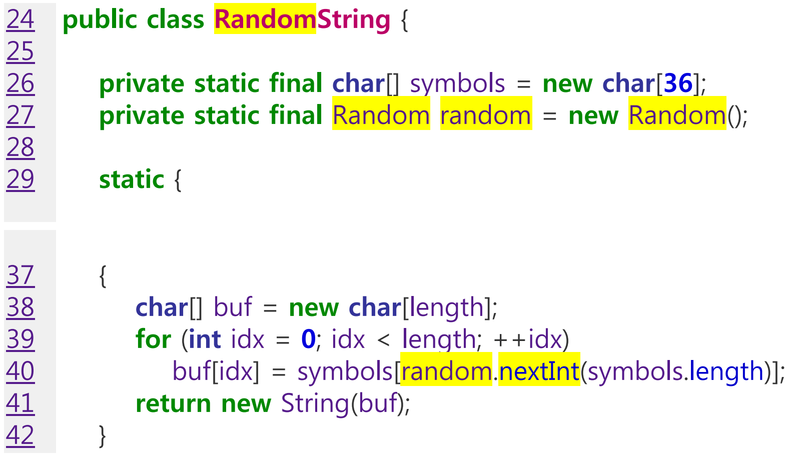
experts in specific and restricted domains; or 2) automatically derive a *thesaurus* [14], e.g., word co-occurrence statistics in an exhaustive corpus; or 3) *interactively expand* user queries [39], e.g., by recommending other terms from previous query logs; or 4) *automatically expand* queries [9] by adding derived words from the terms included in the original query, e.g., add the `integer` word in a query with `int`; or 5) *rewrite* the query automatically [17]. Most of these approaches are not suitable in the settings of a code search engine, since the domain is not restricted, the corpus is not finite and query logs are not always available.

Furthermore in practice, implementing a *code* search engine has its own additional tasks: (1) relevant data is hidden in the deep web and unlinked; (2) the variety of concepts in programming languages, APIs, platforms or development environment challenges indexing; (3) the vocabulary mismatch problem complicates query processing; and (4) granularity of search output (e.g., code snippets, files, or applications) is also challenging to determine and satisfy.

Among the above tasks, query processing is one of the key components since search engine must match the query terms with relevant keywords from the index. The indexing step itself can improve speed and performance in finding relevant documents (source code files in our case) corresponding to a given search query. It often uses the salient keywords in a document. In code search, however, such keywords may not include API names since a single programming concept can be translated and implemented by several different classes and methods. This mismatch may degrade the quality of code search results.

2.1 Limitations of the state-of-the-art

Online code search engines such as OpenHub [2] and Codota [3] perform basic string matching between user free-form queries and the code (which is then strictly consid-



```
public class RandomString {

    private static final char[] symbols = new char[36];
    private static final Random random = new Random();

    static {


    {
        char[] buf = new char[length];
        for (int idx = 0; idx < length; ++idx)
            buf[idx] = symbols[random.nextInt(symbols.length)];
        return new String(buf);
    }
```

Fig. 2: Top result provided by a CoCaBu-based search engine (see Section 3) for the same query used in Figure 1. This code snippet was found in class `org.neo4j.vagrant.RandomString` of *simpsonjulian/neophyte* project from `GitHub`.

ered as a text document, with no distinction between code and documentation). This however produces very low-quality results since programming language terms do not always match natural language words [41].

Figure 1 shows an example of OpenHub's search results for the query "Generating random words in Java?"[2]. This top result from the search engine is not relevant: the returned snippet is for a program that *randomly selects a word from an array of words* rather than generating random words. This inaccurate search result occurs because the words used in the query are not appropriate for direct match with source code terms; "random *string* in Java' is the correct terminology that would have matched a more relevant program. Following results from the search engine were found irrelevant as well. The described example shows the limitation of the current practice in face of the vocabulary mismatch problem.

Our goal is to resolve this vocabulary mismatch problem in order to allow code search engines to return highly relevant code snippets for user free-form queries. Indeed, if we can appropriately transform words used in a search query to keywords found in source code, the search result would be more accurate as shown in Figure 2; this is an actual search result of our approach described in Section 3. The produced code snippet, extracted from real world code, is practically identical to the manually crafted accepted answer for the question in the Q&A post.

Note that state-of-the-art approaches in the literature, such as Muse [35] and MAPO [44], focus on finding usage examples of API methods whose names must be explicitly indicated in the query. Thus, they may not be suitable for development tasks where users do not know the source code keywords of the relevant APIs. In particular, novice programmers may fail to get relevant code usage examples without knowing exactly necessary class or method names.

[2] This is a real question asked by a user in this post: `http://stackoverflow.com/questions/4951997/generating-random-words-in-java`

Other techniques such as Sourcerer [4] have proposed infrastructures to collect and model open source code data that users can query programmatically (e.g., SQL query statements). The Portfolio [34] search engine returns output relevant functions and their usage scenarios. However, these approaches also simply match query terms with function names in the code base.

In summary, because of the vocabulary mismatch problem, current state-of-the-art approaches to code search fail to support entirely free-form and complex queries such as the ones developers are asking to other experienced developers on Q&A sites (cf. query in Figure 1).

2.2 Key Intuition

Q&A posts contain a wealth of information that can be automatically leveraged by a code search engine. A typical Q&A post is a developer question accompanied with answers provided by experienced developers:

- In Q&A sites, developer questions, which are also often rewritten to make them explicit and limit the opportunities for duplicate questions, are good summaries of typical developer query terms.
- Code snippets embedded in experienced developer answers are a good starting point to systematically list relevant source code information related to developer question.

Thus, by leveraging developer questions from Q&A sites, and the associated code snippets, we can document concept mappings, i.e., the mappings between human concepts, which are expressed in questions, and program elements, which can be identified in code snippets. Once a large corpus of such mappings becomes available, the vocabulary mismatch problem can be alleviated. Indeed, any developer query, written in natural language, can be translated into a program query that explicitly makes references to specific program elements such as method and class names. This new query can then be directly matched against any source code file.

3 Our Approach

CoCaBu is about retrieving most relevant source code snippets to answer a free-form query given by a user. To resolve the vocabulary mismatch problem illustrated in Section 2.1, our approach leverages the intuition described in Section 2.2. Figure 3 provides an overview of our approach.

The search process begins with a free-form query from a user, i.e., a sentence written in a natural language:

(a) – For a given query, CoCaBu first searches for relevant posts in Q&A forums. The role of the Search Proxy is then to forward developer free-form queries to web search engines that can collect and rank entries in Q&A with the most relevant documents for the query.

(b) – CoCaBu then generates an augmented query based on the information in the relevant posts. To that end, it mainly leverages code snippets in the previously identified posts. Since these snippets are approved by developers as acceptable code examples from the posted question, CoCaBu can consider them translations of human concepts

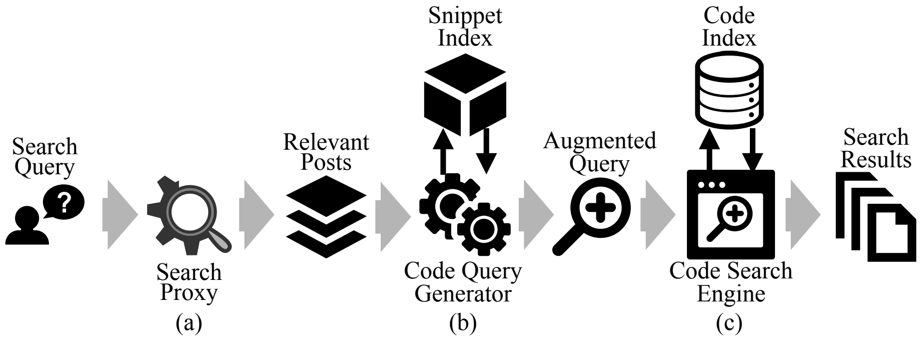


Fig. 3: Overview of CoCaBu.

into program elements. CoCaBu's Code Query Generator then creates another query which includes not only the initial user query terms, but also program elements, such as method and class names, from the extracted snippets. To accelerate this step in the search process, CoCaBu builds upfront a snippet index for Q&A posts.

(c) – Once the augmented query is constructed, CoCaBu searches source code files for code locations that match the query terms. For this step, we can crawl a large number of public code repositories and build upfront a code index for program elements in source code. It then leverages the code index to produce search results for a given augmented query. This search result can be presented to a user at different granularity level (e.g., relevant source code file, or code snippet).

The remainder of this section details the design of CoCaBu components (Sections 3.2 – 3.4) and discusses an implementation case for `GitHub` and `Stack Overflow` (Section 3.5). Before presenting these design and implementation details, we overview a cornerstone aspect element of our approach: the definition and extraction of structural code entities for indexing (Section 3.1).

3.1 Extraction of Structural Code Entities

To efficiently search source code in repositories for relevant code locations that match information from Q&A posts, CoCaBu makes indices of structural code entities in code snippets and source code files. This section describes the structural entities and how to extract them from snippets and source code.

Previous studies on code search and recommendation systems have already proposed to take advantage of structural code information (e.g., method identifiers and class types) to improve query results. Indeed, if provided by user query, this information enables to map source code based on specific program elements. We use similar structural entities to those leveraged in many of previous work [5, 6, 11, 28]. Table 1 enumerates structural code entities that the CoCaBu collects when parsing snippets from Q&A posts and source code files from code repositories.

Wrapping code snippets: While source code from public repositories is mostly compilable, code snippets from Q&A posts are inherently incomplete since they only

```java
URL url = new URL(urlToRssFeed);
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
XMLReader xmlreader = parser.getXMLReader();
RssHandler theRSSHandler = new RssHandler();
xmlreader.setContentHandler(theRSSHandler);
InputSource is = new InputSource(url.openStream());
xmlreader.parse(is);
return theRSSHandler.getFeed();
```

(a) Snippet before recovering name qualification.

```java
URL url = new URL(urlToRssFeed);
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = SAXParserFactory.newSAXParser();
XMLReader xmlreader = SAXParser.getXMLReader();
RssHandler theRSSHandler = new RssHandler();
XMLReader.setContentHandler(theRSSHandler);
InputSource is = new InputSource(URL.openStream());
XMLReader.parse(is);
return RssHandler.getFeed();
```

(b) Snippet after recovering name qualification.

Fig. 4: Recovery of qualification information.

Table 1: Structural Code Entities.

Field	Description
import	Name of import declarations
super	Direct superclass and implemented interfaces
class	Name of used classes
method_declaration	Name of method declarations
nq_method_invocation	Non-qualified method invocations
pq_method_invocation	Partially qualified method invocations
instance	Class instance creations
literal	String Literals

include the necessary statements to convey expert responder explanations on a question. Although few code snippets may contain a complete class declaration, in most cases a code snippet consists of a block of code statements. Snippet authors furthermore frequently use ellipses (i.e., "...") before and after code blocks. Thus, CoCaBu removes ellipses and wraps code snippets by using a custom dummy class and method templates to make it able to parse by standard Java parsers.

Qualifying non-qualified names: In addition to wrapping snippets, our approach reasons about qualified names in code snippets. Enclosing class names of methods in snippets are often ambiguous [13] (i.e., method name qualification). For example, Subramanian et al. [42] found that there are unqualified method name `getId()` more than 27,000 times in their oracle containing 1.6 million types (i.e., classes and method-/field signatures) whereas partially qualified name `Node.getId()` can be identified only few times. Thus, recovering unqualified names can improve the accuracy of code search.

To recover qualified names of methods, CoCaBu transforms unqualified names to partially qualified names using structural information collected during AST traversal. Specifically, it converts variable names on which methods are called through their respective classes. Figure 4 illustrates this processing step with an example of code snippet before and after the method qualification.

Text processing: In addition to structural entities, our approach collects textual information as well. By treating source code as text, the approach conducts preprocessing such as tokenization (e.g., splitting camel case), stop word removal[3] [32], and stemming.

Indexing: With the collected set of information, CoCaBu can build an index of text terms as well as structural code entities found in the source code. To create an index, we build our approach on top of the Lucene[4]. Lucene stores data as an index, each consisting of a set of fields, where each field value represents a basic code element for search in our case. Fields are populated with the structural and textual information, produced by the above process, along with the index specific metadata. Further details on this process are provided in Section 3.4.

3.2 Search Proxy

The search proxy takes a free-form query as an input and returns a set of relevant posts collected from developer Q&A sites as an output. The goal of this component is **to collect sufficient data so that the search engine can later find out how natural language concepts can be translated into program elements.** Indeed, code snippets in answers of Q&A posts can provide potential translation rules from concepts written in natural languages to program elements such as API methods or classes. As discussed in Section 2, such translation rules facilitate the subsequent code search process by alleviating the vocabulary mismatch problem that exists between user queries and source code elements.

Relying on general purpose engines such as Google Web Search, Bing, and Yahoo Search, CoCaBu can search several different forums and rank the search results according to their relevancy to the query. Thus, in practice, once a user submits a code search query, the search proxy forwards it to a general-purpose web search engine to obtain related questions in the web. Since these search engines are specialized for text search, we assume that they are better than other built-in search engines in Q&A forums. Web search results are then filtered by the search proxy to eliminate URLs not related to Q&A posts. For example, if we want to consider only `Stack Overflow` posts, the search proxy would try to match the following pattern to collect relevant posts:

```
http://stackoverflow.com/questions/<ID>/<TITLE>
```

The ranking of relevant posts is directly preserved from the sorting order proposed by the general-purpose search engine. If we consider for example the question "Gener-

[3] Lucene's (version 4) English default stop word set.

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

ating random words in Java?" described in Section 2, the search proxy supported by Google Web Search returns the relevant posts[5] as listed in Table 2.

Table 2: List of Q&A posts relevant to 'Generating random words in Java?'

Q&A site	Post title	Post ID
Stack Overflow	Generating random words of a certain length in java?	27429181
Stack Overflow	Random word from array list	20358980
dummies.com	How to Generate Words Randomly in Java	-
java2notice.com	How to create random string with random characters?	-
coderanch.com	Random string generation	374794

3.3 Code Query Generator

The Code Query Generator creates a code search query that augments and structures the free-form query taken by the search proxy (Section 3.2). This augmented query is a list of program elements, such as class and method names (e.g., `Math.random`), as well as natural language terms which can be used to match documentation.

To generate the augmented query, CoCaBu must extract structural code entities from code snippets embedded in the answers to the questions in the relevant posts returned by the search proxy (Figure 5(b)). The code query generator component only considers accepted answers, i.e., answers approved by the Q&A site community.

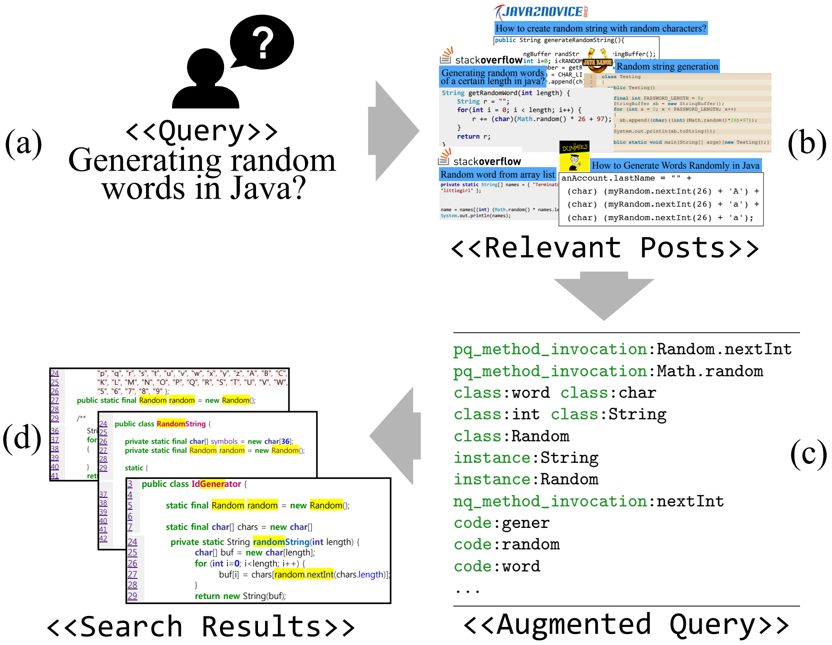
The augmented query produced by the code query generator is illustrated in Figure 5(c) based on the Lucene search engine query format. The reader can observe the following from the illustrated example query whose field semantics are previously described in Table 1:

- terms, excluding stop words, in the user free-form query (i.e., Figure 5(a)) are kept, after stemming, in the augmented query (e.g., `code:gener`).
- structural code entities collected from Q&A snippets (i.e., Figure 5(b)) are mentioned with their type (e.g., non-qualified/partially qualified method invocation, or class) in the augmented query (e.g., `pq_method_invocation:Random.nextInt`).

To accelerate code query generation, CoCaBu builds an index of posts. Typically, Q&A forums provide archives of their posts. These posts are often formatted by a structural language such as XML. For example, in Stack Overflow posts, code snippets are enclosed in `<code> ...</code>`. As shown in Figure 6, our approach takes pre-downloaded posts from a Q&A site and extracts metadata (post ID, question title) and code snippets for each post. Each code snippet is then analyzed to retrieve the structural code entities. This phase presents challenges that will be addressed in Section 3.1.

Building an index upfront reduces the query generation time when the target post is already indexed. For new posts collected, the component follows the process shown in Figure 6 to insert it into the index.

[5] In this illustrative example, we excluded the actual post (http://stackoverflow.com/questions/4951997/generating-random-words-in-java) where this question is asked. To eliminate bias, in all experiments described in Section 4, in which we selected a question of a Q&A site as a subject, we removed the corresponding posts from the list of relevant posts to be used for augmenting the query.



How to create random string with random characters?

```
public String generateRandomString(){
```

Generating random words of a certain length in java?

```
String getRandomWord(int length) {
    String r = "";
    for(int i = 0; i < length; i++) {
        r += (char)(Math.random() * 26 + 97);
    }
    return r;
}
```

Random string generation

```
class Testing
{
  public Testing()
  {
   final int PASSWORD_LENGTH = 8;
   StringBuffer sb = new StringBuffer();
   for (int x = 0; x < PASSWORD_LENGTH; x++)
   {
     sb.append((char)((int)(Math.random()*26)+97));
   }
   System.out.println(sb.toString());
  }
  public static void main(String[] args){new Testing();}
```

Random word from array list

```
private static String[] names = { "Terminat
"littlegirl" };

name = names[(int) (Math.random() * names.le
System.out.println(names);
```

How to Generate Words Randomly in Java

```
anAccount.lastName = "" +
  (char) (myRandom.nextInt(26) + 'A') +
  (char) (myRandom.nextInt(26) + 'a') +
  (char) (myRandom.nextInt(26) + 'a');
```

```
pq_method_invocation:Random.nextInt
pq_method_invocation:Math.random
class:word class:char
class:int class:String
class:Random
instance:String
instance:Random
nq_method_invocation:nextInt
code:gener
code:random
code:word
...
```

```
        "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "A", "B", "C",
        "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W",
        "5", "6", "7", "8", "9" };
    public static final Random random = new Random();
```

```
public class RandomString {

    private static final char[] symbols = new char[36];
    private static final Random random = new Random();

    static {
```

```
public class IdGenerator {

        static final Random random = new Random();

        static final char[] chars = new char[]
         private static String randomString(int length) {
                char[] buf = new char[length];
                for (int i=0; i<length; i++) {
                        buf[i] = chars[random.nextInt(chars.length)];
                }
                return new String(buf);
```

Fig. 5: Illustrative input, intermediate results, and output of a CoCaBu-based code search engine.

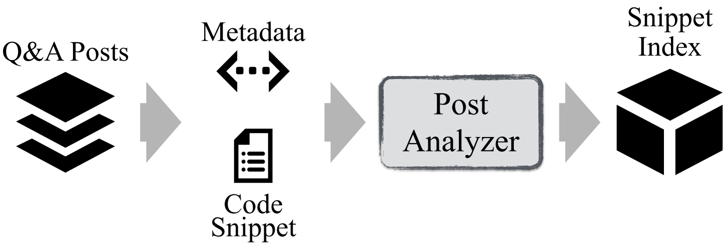


Fig. 6: Creating an index for metadata and code snippets of Q&A posts.

3.4 Code Search Engine

The code search engine takes an augmented query from the code query generator and provides a list of search results to the user who issued the original query. The search results are of two granularity levels:

- In case the query is augmented, granularity is further controlled since the structural code entities matched within a source file and the search result can focus on showing only the excerpt with code lines where a match occurred.
- When the query has not been augmented (i.e., the search proxy did not find any Q&A post link within the top ten web search results set), the search engine returns for each result a whole file.

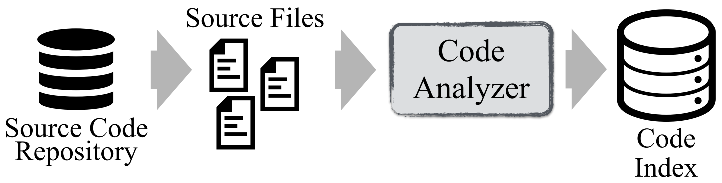


Fig. 7: Creating an index for source code in code repositories up front.

To efficiently provide answers for augmented queries, the code search engine builds an index of source code files found in repositories (cf. Figure 7). The matching then becomes straightforward as the structural entities in the augmented queries as well as the NLP terms are directly search for using the index which will list the most relevant files.

Since the snippet index and the code index (shown in Figure 6 and 7, respectively) store indices in the same format, full-text search can be effective to obtain search results. Source code files are then the documents while structural code entities represents the search terms.

Once search results are retrieved, the code search engine computes rankings of the source code files based on a scoring function that measures the similarity between the matched files and query terms. The current implementation of CoCaBu uses the scoring function implemented in the Lucene library. This function combines the Boolean Model (BM) and the Vector Space Model (VSM) to determine the relevancy of a document given for a user query[6]. BM is used for reducing the amount of documents that need to be scored by using Boolean logic in the query specification. Each document is represented as a vector $d = (w_1, w_2, ..., w_n)$ where w_i corresponds to the weight of a term occurring in that document. To compute these weights we use the TF-IDF weighting scheme implemented in Lucene. With these weights, VSM computes the similarity between the documents by using the cosine similarity measure[7].

Since displaying the entire content of a source code file is often ineffective for users to understand code examples, the code search engine shows the files after summarizing the content and highlighting lines of code relevant to a given query [32]. To summarize and highlight search results, CoCaBu uses a query-dependent approach that displays segments of code based on the query terms occurring in the source file. Specifically, the component displays a set of adjacent lines of code containing the matching query keyword. Finally, we highlight query words occurring in the summarized file to ease their identification.

3.5 The GitSearch Code Search Engine

This section describes an example instantiation of the CoCaBu approach. We build GitSearch, a code search engine on top of `GitHub` and `Stack Overflow` to explore the large amounts of source code and Q&A posts. In the remainder of this section we detail the implementation choices that were made in GitSearch.

[6] https://goo.gl/MqETzP (last accessed 12.07.2015)

[7] `https://goo.gl/VPvxnX` (last accessed 12.07.2015)

Table 3: Statistics of collected projects from GitHub.

Feature	Value
Number of projects	7,601
Number of files	1,705,677
Number of duplicate files	182,043
LOCs	> 297 M

Table 4: Descriptive statistics of the snippet index and code index built from Stack Overflow posts and GitHub projects, respectively.

Feature	Code Index	Snippet Index
	from GitHub	from Stack Overflow
# of Documents	1,310,954	230,416
pq_method_invocation	5,243,472	75,079
method_declaration	3,463,861	50,900
class	2,031,608	120,468
nq_method_invocation	1,994,667	82,253
literals	1,526,440	82,253
instance	887,861	40,131
super	296,654	8,329

To build GitSearch, we selected Stack Overflow as the Q&A site where to retrieve relevant developer-approved code snippets. For the search proxy, our implementation directly leverages Google web search[8]. User queries are sent to Google Search for retrieving all relevant Q&A posts (i.e., text similarity matching). Note that it is possible for other implementations to use other web search engines including built-in search services of Q&A sites.

We used a dump of Stack Overflow posts between July 2008 and March 2015 containing 1,363,002 Java and Android tagged questions to build the snippet index. Java was selected in this instantiation since it is one of the most popular programming languages and represents a large developer base [8]. In this work, we made use of the posts.xml documents that have an actual post (i.e., question and answer pair) and other associated metadata such as tags, creation date, question ID, view count of the post, and score of answers. In addition, we extracted snippets from answers that were accepted and had a positive score to ensure high quality of code examples. To account for updates in posts, we leveraged the StackExchange REST API[9] with which we could extract metadata and snippets. Users of CoCaBu may collect and use posts from other multiple Q&A forums to extend the opportunity to search for more code snippets.

For the code index, we considered GitHub projects that were forked at least once, to avoid toy and/or inactive projects. Since we focused on Java and Android, we collected GitHub projects in which its major language is "Java" and then removed all non-Java files from the projects when building the code index. As a result, Table 3 shows the statistics of GitHub projects we collected in this work.

Table 4 provides a summary of the resulting indices (i.e., the snippet and code indices shown in Figures 6 and 7) built from Stack Overflow posts and GitHub open source code repositories.

[8] www.google.com
[9] https://api.stackexchange.com/

4 Evaluation

This section describes our evaluation design and reports its results. Our evaluation consists of four studies: a manual verification, online survey, controlled user study, and live study, focusing on answering the following research questions, respectively:

- **RQ1**: Can GitSearch effectively produce relevant code examples for developer queries?
- **RQ2**: Does GitSearch outperform existing code search engines with more acceptable results?
- **RQ3**: Is GitSearch competitive against general search engine for helping to solve programming tasks?
- **RQ4**: Can `Stack Overflow` users accept the search results of GitSearch as answers?

4.1 RQ1: Verification against a community ground truth

First, we investigate the relevance of the results yielded by GitSearch. To evaluate the relevance, we consider comparing the output code examples against the ground truth of code snippets in answers accepted by the `Stack Overflow` community. This type of verification, which is commonly used in the literature [4,29], is essential since developers can be quickly deterred by search engine producing many irrelevant results.

Study Design: We collect well-known developer questions from `Stack Overflow` posts based on two requirements: (i) a question in a post must relate to "Java" and (ii) its answer must include code snippets. We select the top 10 posts with the highest 'view count' values (for their questions) to ensure that the study focuses on representative and popular developer tasks. Table 5 lists the queries used in this study. Note that this process does not bias in favor of our approach. Indeed, for fair comparison, the actual post where the question is asked is filtered out from the relevant posts, returned by the search proxy that GitSearch uses to augment user queries.

Table 5: Free-form queries used for RQ1 and RQ2.

ID	Query Terms
Q1	How to add an image to a JPanel?
Q2	How to generate a random alpha-numeric string?
Q3	How to save the activity state in Android?
Q4	How do I invoke a Java method when given the method name as a string?
Q5	Remove HTML tags from a String
Q6	How to get the path of a running JAR file?
Q7	Getting a File's MD5 Checksum in Java
Q8	Loading a properties file from Java package
Q9	How can I play sound in Java?
Q10	What is the best way to SFTP a file from a server?

We evaluate the top 5 code search examples by GitSearch. To assess the relevancy of a GitSearch code example, two authors of this paper compared it against the accepted answer on `Stack Overflow` for the associated query. We consider that the example is indeed relevant when it includes the necessary API methods and classes

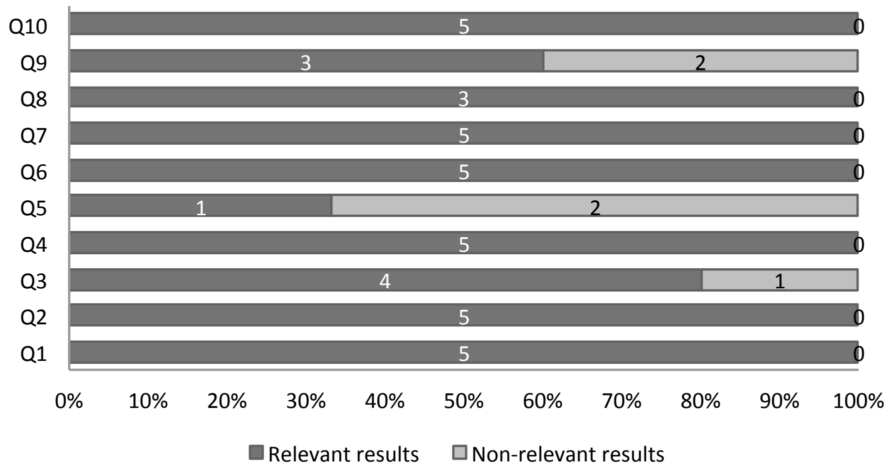


Fig. 8: Relevance of top 5 GitSearch results for popular queries listed in Table 5.

required in the `Stack Overflow` answer's code snippet. To increase confidence, both authors must unanimously agree on the relevance of a GitSearch result.

Results: Figure 8 shows that GitSearch results are largely relevant for the user query, indirectly demonstrating the accuracy of the query expansion approach.

We also evaluate the effectiveness of GitSearch using the *Precision@k* metric:

$$Precision@k = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{|relevant_{i,k}|}{k} \tag{1}$$

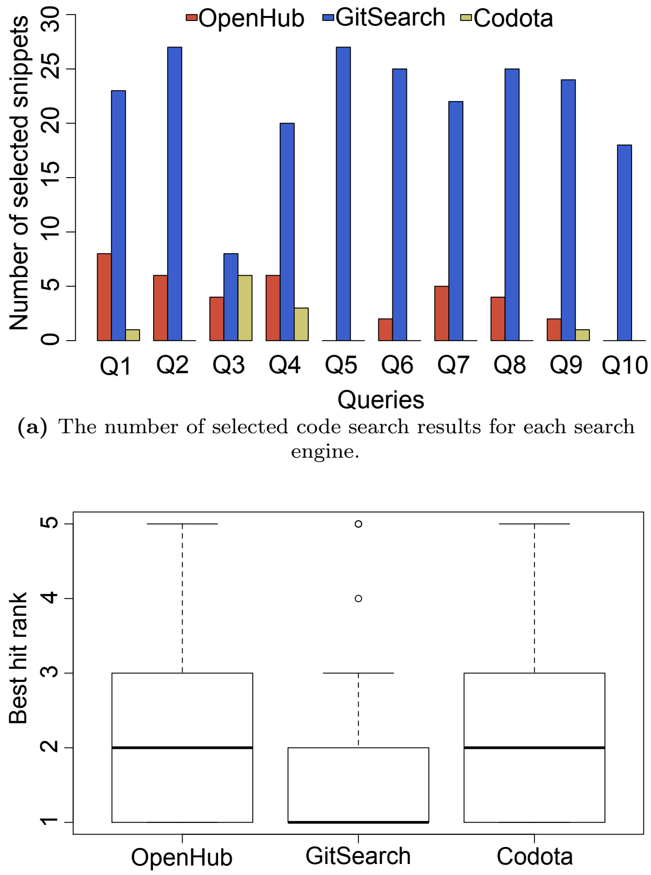
where $relevant_{i,k}$ represents the relevant code search results for query i in the top k returned results, and Q is a set of queries. *Precision@k* takes an average on all queries whose relevant answers could be found by inspecting the top k (k = 1, 2, 5) of the returned code examples. An effective code search engine should allow developers to find the relevant code examples by examining fewer returned results. Thus, the higher *Precision@k*, the better code search performance. We found that GitSearch achieves 90%, 90% and 88% scores for *Precision*@1, *Precision*@2 and *Precision*@5, respectively. We could not define *recall@k* because it is impossible to compile the "complete" set of all possible correct answers for a code search query.

In addition, we applied the same queries in Table 5 to other code search engines: OpenHub [2] and Codota [3]. These code search engines were selected since they are state-of-the-art Internet-scale code search engines and currently available online. On the other hand, we could not compare GitSearch against other recent state-of-the-art approaches from the literature because of the reasons listed in Table 6.

The *Precision@k* values of those two search engines are lower than that of GitSearch. OpenHub resulted in 60%, 60%, and 38% scores for *Precision*@1, *Precision*@2, and *Precision*@5, respectively. For Codota, these values are 10%, 10%, and 12%, respectively.

Table 6: Unavailability of code search tools and techniques.

Portfolio [34] is not available (anymore) and supports only C++.
Exemplar [33] is no longer available.
Sourcerer [4]'s team did not reply about the use of their SAS code search engine.
Muse [35] is not relevant: - focuses on API - cannot be queried for snippets.
SNIFF [10] engine could not work (issue with the Eclipse plugin).
Keivanloo et al's [25]' tool is no longer available (lead developer left the project).
CodeHow [29] is not available - only a demo video online.



(a) The number of selected code search results for each search engine.

(b) Distribution of rankings for the selected search results (the lower the better).

Fig. 9: Comparison between GitSearch, Codota and OpenHub.

4.2 RQ2: Comparison against other code search engines

We conduct a user study where we ask developers to check the effectiveness of different code search engines, to assess the usefulness of GitSearch from the perspective of practitioners.

Study Design: For this study, we recruited participants by posting online survey invitations in software developer communities (750 GitHub, Mozilla, and Eclipse developers, and developers in a Korean company). In all survey invitations, we clearly stated that only developers/students who have Java experience are invited. To facilitate the study, we built a web-based survey tool displaying the code search results from OpenHub, Codota, and GitSearch in three anonymized columns. To avoid bias of people toying with the tool, we only consider the entries of participants who entirely completed the study using the queries in Table 5.

Participants can select code examples based on their preference. They can select multiple search results (up to three). We also clearly ask them to select no result if none is satisfying them. In addition to anonymization, the survey tool excludes the source `Stack Overflow` posts listed in Table 5 from the training data of GitSearch to avoid any bias.

Results: At the end of the study, we had 47 participants who tried the tool (at least one response). Some of them did not complete the study. Among them, 14 participants completed this study.

Figure 9(a) shows the number of selected search results for each code search engine. Participants selected more code examples returned by GitSearch than other engines for all queries. In particular, the number of selected results was more than double compared to others except for Query Q3.

In addition, we computed the distribution of rankings for the selected search results. If multiple search results of an engine were selected by a user, we counted the highest ranked result only. As shown in Figure 9(b), the median value of GitSearch is equal to 1 while the values of other engines are 2.

Discussion: Although we could not compare against the most recent CodeHow tool [29], note that its authors reported that it produces about 20% more relevant results than OpenHub[10] while Figure 9(a) indicates that GitSearch provides 50% more relevant results[11] than OpenHub.

4.3 RQ3: Comparison against general search engines

We conducted a comparative study between GitSearch and general web search engines (Google and Baidu). Since many developers rely on general search engines to find solutions to programming tasks, we evaluate the competitiveness of GitSearch in comparison to such engines.

Study Design: For this study, we recruited 20 graduate students from three universities (Pierre and Marie Curie University in France, University of Luxembourg, and Zhejiang University in China). No author of this paper took part in the study. Each student was asked to find code examples for solving the following two programming tasks from a previous code search study [29]:

- *Task 1:* Sending emails - write a Java program to read a list of email addresses from a text file, and then send an email with an attachment file to all the email addresses.
- *Task 2:* Image format conversion - write a Java program to read an image in JPEG format, rotate it 180, and then convert it to PNG format.

[10] Ohloh is now OpenHub.

[11] Despite different queries, our query sets are similar to those of [29] and representatives of common developer search queries.

Table 7: Performance of GitSearch vs. general search engine.

	Percentage of successful queries†		**MRR**	
	GitSearch	Google/Baidu	GitSearch	Google/Baidu
Task 1	**93.76%**	90.00%	0.83	**0.96**
Task 2	75.00%	**100.00%**	**0.89**	0.84

† We compute the ratio of queries having produced satisfying results vs. the total number of queries entered per task.

Participants to the controlled study have been asked to solve one task with GitSearch and the other task with Google or Baidu. We specify ourselves the combinations (task, tool) for every participant in order to ensure an even distribution. Each participant fills a form indicating the different free-form queries used for code search as well as the rank of the returned results that he/she found relevant for the task. We specified that only top 10 results returned by the tools could be examined.

We assess the efficiency of the engines through the Mean Reciprocal Rank (MRR), a statistical metric used to evaluate a process that produces a list of possible responses to a query [18]. The reciprocal rank of a query is the multiplicative inverse of the rank of the first relevant answer. The mean reciprocal rank is the average of the reciprocal ranks of results of a set of queries Q. MRR is computed by using the formula:

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i} \tag{2}$$

where $rank_i$ represents the rank of the first search results that users find satisfying for query i. MRR values range between 0 and 1, and the higher MRR value the better the performance.

Results: Participants to the study entered 77 (37 for Task 1 and 40 for Task 2) distinct free-form queries.

Table 7 shows the percentage of relevant search results that participants in the study marked for the different search engines. GitSearch provides more satisfying results for Task 1 while users found more satisfying results with Google/Baidu for Task 2. In contrast, for Task 2, GitSearch outperforms Google/Baidu in terms of MRR, returning in higher ranks the satisfying results. On the other hand, GitSearch has a lower MRR for Task 1 results. These results suggest that GitSearch is competitive against web search engines. We do not, however, take into account the effort required in web search to follow link redirections and parse web pages to find potentially incomplete code snippets. GitSearch on the other hand provides immediately real world working code examples.

Discussion: We investigated the 77 queries entered by participants in this study. We figured out an interesting pattern: queries entered on web search engines appeared to be more "complete" and more redundant across participants than queries entered on GitSearch. Participants to the study admitted that they followed auto-completion suggestions by web search engines.

We perform a cross-validation experiment by randomly sampling 10 queries entered by participants on web search engines and use them on GitSearch. Similarly, we randomly sample 10 queries entered by participants on GitSearch and use them on Google search engine. We record improved MRR values of 0.94 and 0.90 with Task 1 and Task 2 respectively for GitSearch. In contrast, MRR values for the web search engine has decreased to 0.72 and 0.65 with Task 1 and Task 2 respectively.

These results suggest a future work on GitSearch where we must include logging and feedback mechanisms to record successful queries and propose them to auto-complete queries of future requesters.

4.4 RQ4: Live study into the wild

To assess the usefulness of code search engines in Q&A forums, we posted code search results as answers to `Stack Overflow` questions. This study investigates how developers interpret working code examples when they have programming issues. Although GitSearch is not designed to directly answer developers' questions, it might help them find a starting point of a programming task. In particular, GitSearch can be a good first responder in the context of `Stack Overflow` since there are many unanswered questions (not only "no answer selected by questioners" but also literally "no answer") in `Stack Overflow`.

Study Design: We monitored questions with **Java** tags and selected 25 out of them based on the following criteria:

- Questions about Java programming.
- "How-To" questions such as "List all files in resources directory in java project".
- No tool usage questions such as "How to create a project in Eclipse?"
- No conceptual questions such as "What is the difference between **A** or **B**" and "why this class is so slow?".
- Questions not answered by anyone yet.

For each question, we extracted its title and put it into GitSearch to obtain code search results. We took the topmost result among the search results and posted it as an answer. The answer consists of 1) the most relevant code fragments selected by GitSearch and 2) hyperlink for the original source code where GitSearch found the fragments from. The latter is important since developers can figure out more context about the working code examples. In addition, we repeated the same procedure with OpenHub to compare its effectiveness with our technique. We do not post Codota's results on `Stack Overflow` to avoid "spamming" requesters, as we could see ourselves that its topmost result was irrelevant for most questions.

Results: GitSearch could answer more questions than OpenHub as shown in Table 8 ("Resp" of "#Ans"). GitSearch responded 25 questions by using its search results while OpenHub did only for 18 out of 25 questions. For the other seven questions, OpenHub could not produce any search result for reasons that are unknown to us (perhaps, due to an issue of the engine's query matching implementation). In addition, at the end of the study, `Stack Overflow` users eventually answered only 8 out of 25 questions. Note that three answers were accepted by the questioners among the 25 answers by GitSearch. None of 18 answers by OpenHub were accepted. For human answers, questioners accepted four out of 8 answers.

Our technique received more up and downvotes than OpenHub while human answers took more upvotes and less downvotes. Six out of 25 answers by GitSearch received at least one upvote while other five of them took at least one downvote (six upvotes and 10 downvotes in total). Four of the six were the most-upvoted answers in their posts. OpenHub's answers had only two upvotes and three downvotes, respectively. Human answers took 9 upvotes (from four different answers) and 1 downvote (note that a single answer took 4 upvotes) where three answers were most-voted. In

Table 8: Results of our live study between GitSearch, OpenHub, and Human. "Resp" in "#Ans" is the number of questions answered by each technique while "Acc" is the number of answers accepted by the questioners. "Up and down" votes are the number of votes given by `Stack Overflow` users. "Pos." and "Neg. Comm." comments are positive and negative comments made by the users for each answer. "Most voted?" represents the number of answers that received the most number of upvotes. $|x|$ indicates the number of answers with at least one up/down vote and positive/negative comment. Σ is the sum of occurrences while the numbers in parentheses are average (i.e., $\Sigma/|x|$).

	#Ans		Upvotes		Downvotes		Pos. Comm.		Neg. Comm.		Most
	Resp	Acc	$\|x\|$	Σ	$\|x\|$	Σ	$\|x\|$	Σ	$\|x\|$	Σ	voted?
GitSearch	25	3	6	6 (0.24)	5	10 (0.40)	7	7 (0.28)	3	5 (0.20)	4 (0.16)
OpenHub	18	0	2	2 (0.11)	2	3 (0.17)	2	2 (0.11)	2	3 (.17)	0
Human	8	4	4	9 (1.13)	1	1 (0.13)	3	6 (0.75)	2	2 (0.25)	3 (0.38)

`Stack Overflow`, votes imply that those users would encourage (or discourage) the answer. While its up and downvotes were almost tied with human results, it is obvious that GitSearch had more interest from users than OpenHub.

In addition, GitSearch initiated user discussions more frequently. We counted comments made by `Stack Overflow` users and examined whether each comment is positive and negative. Our answers took 7 positive and 5 negative comments while OpenHub's results were followed by two and three, respectively. GitSearch does not explicitly outperform human answers (6 positive and 2 negative) but note that there were 17 of out 25 questions unanswered yet by human users. For the 17 questions, our technique answered them and received one upvotes and three downvotes as well as three positive and two negative comments.

Discussion: The results of this study implies that GitSearch can be a better first responder than OpenHub. As shown in Table 8, many questions in `Stack Overflow` remain unanswered for several days. Our technique can provide a starting point of questions even if they are not complete answers as many users would follow up the answers by giving their votes and adding comments. Once users are interested in a question, there might be more probability to discuss solutions for the question.

In addition, code search results by GitSearch can be selected by `Stack Overflow` users as accepted answers, which implies that the results are highly relevant and appropriate to the questions. For three out of 25 questions, the questioners accepted our results even though the answers have only code excerpt from real source code without any additional explanation. Note that a questioner can select only one answer as the accepted one. This may indicate that questioners would take advantage of code search results to deal with their problems shown in the question. Furthermore, this can imply that code search engines would be an automatic answer generator for some questions in `Stack Overflow` if their accuracy is improved.

4.5 Threats to Validity

The design of CoCaBu and the implementation of GitSearch raises a number of threats to validity that we have tried to mitigate. We list them below:

Internal validity: the user study was performed with a limited total number of 34 (=14+20) participants compared to the large number of participants used by

Muse [35] authors for their API example search engine. However, among free-form code search works, some do not perform user studies (e.g., [4]), while others use fewer participants than us (e.g., CodeHow (20), Portfolio (19), SNIFF (undisclosed)). We have attempted to reach representativity by inviting professional developers as well as graduate students.

In addition, throughout the live study (Section 4.4), we tried to take feedback from `Stack Overflow` overflow users in the loop of problem solving. This implies that an additional number of participants were involved in our evaluation.

External validity: we used only English as a query language, focused on Java-related questions, and explored only `Stack Overflow` and `GitHub` in our implementation. This threat should be limited by the fact that (1) English is a popular language in the programming community, (2) Java is one the most popular programming languages, and furthermore, (3) `GitHub` and `Stack Overflow` are the largest code hosting site and Q&A forum respectively.

Construct validity: we only focus on queries with no exact name of APIs. This threat, however, is limited since for new tasks, developers often do not know the name of the relevant APIs [23].

5 Related Work

There are several research work that relates to our approach. We list their main contributions in each category.

5.1 API usage examples search

Recently, there have been been a number of code search techniques [6,19,25,31,35,43], focusing on locating API usage examples. Searching for specific API usages is a subset of code search activities. Compared to general code search, developers tend to be aware of the exact (or similar) name of a target API, which facilitates search. Thus, these techniques focus on creating an index of API call sites only.

Moreno et al. [35] proposed Muse, an approach to mining and ranking code examples that show how to use a given method. Muse and CoCaBu differ on three main aspects. First, CoCaBu supports free-form queries, while Muse takes as input an API method signature. Second, Muse provides a code snippet for a specific method. CoCaBu, on the other hand, is not attached to a single API, and shows a set if APIs used to solve the task at hand. Lastly, Muse requires fully compilable client projects in order to apply static slicing. In contrast, CoCaBu is able to handle incomplete source code.

Chatterjee et al. presented SNIFF [10], a technique that combines API documentation with publicly available Java code. SNIFF annotates each method call statement with its corresponding API documentation. This allows free-form English queries about the task at hand, which relaxes the need to know the appropriate API beforehand. Although SNIFF returns usage code examples as well, it requires a fully compilable code unit and the accompanying API documentation as well as external libraries. Additionally, the before-mentioned code intersection is not suitable for Internet-scale code search, because it has a complexity of O(n^2), where n is the number of hits.

5.2 Source code search

There have been several approaches to code search, which are relevant to CoCaBu. CodeHow, Sourcerer and Portfolio constitute the state-of-the-art of such approaches in the literature. CodeHow [29] leverages code documentation to recognize the potential APIs a query refers to and expands the query with these APIs to improve the accuracy of the search results. In contrast, CoCaBu assumes that 1) documentation is not always available, and 2) leveraging independent API documentation may create noise in a query whose answer requires a specific set of related APIs. Furthermore, CoCaBu augments queries based on information of code terms in source code snippets.

Sourcerer [4] is an infrastructure that facilitates the collection and analysis of large scale open-source repositories. On top of that infrastructure, Sourcerer provides programmatic access to all the artifacts stored and managed through a set of services. Sourcerer crawls Java projects from several types of code repositories such open code repositories (e.g. Sourceforge and Apache) and web sites. Similar to CoCaBu, Sourcerer leverage structural code information to perform fine-grained code search. However, the construction of the search index requires a complete compilation unit (i.e., all dependencies must be resolved). Moreover, we exploit high-quality code snippets from `Stack Overflow` to improve the quality of code search results.

Portfolio [34] retrieves and visualizes relevant functions and their usage scenarios to highlight a chain of function invocations. To realizes their objective, Portfolio computes the textual similarity between a user query and the function signatures. Subsequently, a function call graph is employed to locate functions which are relevant to a task, even if those function signatures do not include any keywords of the query. Compared to Portfolio, CoCaBu focuses on usage examples that answer complex queries by leveraging `Stack Overflow` code snippets.

OpenHub Code Search [2] (formerly ohloh.net) is a free web-based code search engine. Although OpenHub has indices of more than 21 billion lines of code collected from open source projects in the Internet, it directly matches query terms with terms in source files. This is a common limitation of several Internet-scale search engines, including Codota [3]. Contrary to them, we resolve the vocabulary mismatch problem by augmenting user queries.

5.3 Miscellaneous

Code recommendation: Recommendation engines assist developers in their use of complex libraries or frameworks by presenting them with reusable code fragments in other locations of their code, with documentation, or with pointers to blogs and Q&A sites. Strathcona [24] is an approach in which a query is generated from a user's source code and matched with an example repository that uses a target library of framework. They thus require prior knowledge on the relevant library.

Prompter [38], on the contrary, does not provide code snippets but matches the current code context with relevant `Stack Overflow` posts. The technique relies on different features to capture the similarity between `Stack Overflow` discussions and the current code context. In contrast, our approach does not recommend discussions but use `Stack Overflow`'s code snippets to search for similar usage examples in a large code repository.

Stack Overflow: Several studies have explored `Stack Overflow` questions and answers [1,7,30,36]. However, to the best of our knowledge, its data has never been leveraged to improve code search engine results.

6 Conclusion

We have presented CoCaBu, a novel approach to addressing the vocabulary mismatch problem in code search. CoCaBu augments free-form queries by leveraging code snippets in answers of related posts from Q&A sites. The key insight from our work is that it is possible to map human concepts expressed in queries (which are often written with similar terms by developers) with structural code entities (which are the most relevant terms for matching source code with high relevance). We implemented a code search engine, GitSearch, following the CoCaBu approach for the `GitHub` super-repository of projects. To that end, we leveraged `Stack Overflow` posts to find the best mappings between developer query terms and structural code entities. Our evaluation with user studies demonstrated that GitSearch outperforms Internet-scale code search engines and is competitive against established web search engines for resolving programming tasks. We also found with a live study that users in Q&A forums show interest in the real-world code examples yielded by GitSearch.

Availability

We make all our data available: source code of GitSearch, search indices, user study results. See `https://github.com/serval-snt-uni-lu/cocabu`. A prototype implementation of cocabu-based search engine, GitSearch, is live at `http://www.cocabu.com`.

References

1. Augmenting api documentation with insights from stack overflow. In: (to Appear) Proceedings of the 2016 International Conference on Software Engineering, ICSE '16 (2016)
2. http://code.openhub.net (2016). Last accessed 12.03.2016
3. http://www.codota.com (2016). Last accessed 12.03.2016
4. Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., Lopes, C.: Sourcerer: a search engine for open source code supporting structure-based search. In: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, pp. 681–682. ACM (2006)
5. Bajracharya, S.K.: Facilitating internet-scale code retrieval. Ph.D. thesis, Long Beach, CA, USA (2010). AAI3422111
6. Bajracharya, S.K., Ossher, J., Lopes, C.V.: Leveraging usage similarity for effective retrieval of examples in code repositories. In: Proceedings of FSE, pp. 157–166. ACM (2010)
7. Barzilay, O., Treude, C., Zagalsky, A.: Facilitating crowd sourced software engineering via stack overflow. In: Finding Source Code on the Web for Remix and Reuse, pp. 289–308. Springer (2013)
8. Bissyande, T., Thung, F., Lo, D., Jiang, L., Reveillere, L.: Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In: Computer Software and Applications Conference (COMPSAC), 2013 IEEE 37th Annual, pp. 303–312 (2013). DOI 10.1109/COMPSAC.2013.55
9. Carpineto, C., de Mori, R., Romano, G., Bigi, B.: An information-theoretic approach to automatic query expansion. ACM Trans. Inf. Syst. **19**(1), 1–27 (2001). DOI 10.1145/366836.366860. URL `http://doi.acm.org/10.1145/366836.366860`

10. Chatterjee, S., Juvekar, S., Sen, K.: Sniff: A search engine for java using free-form queries. In: Fundamental Approaches to Software Engineering, pp. 385–400. Springer (2009)
11. Chen, T.H., Thomas, S.W., Nagappan, M., Hassan, A.E.: Explaining software defects using topic models. In: Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, MSR '12, pp. 189–198. IEEE Press, Piscataway, NJ, USA (2012). URL http://dl.acm.org/citation.cfm?id=2664446.2664476
12. Cleland-Huang, J., Czauderna, A., Gibiec, M., Emenecker, J.: A machine learning approach for tracing regulatory codes to product specific requirements. In: ACM/IEEE 32nd International Conference on Software Engineering, vol. 1, pp. 155–164 (2010). DOI 10.1145/1806799.1806825
13. Dagenais, B., Robillard, M.P.: Recovering traceability links between an api and its learning resources. In: Software Engineering (ICSE), 2012 34th International Conference on, pp. 47–57. IEEE (2012)
14. Eckert, K., Stuckenschmidt, H., Pfeffer, M.: Interactive thesaurus assessment for automatic document annotation. In: Proceedings of the 4th International Conference on Knowledge Capture, K-CAP '07, pp. 103–110. ACM, New York, NY, USA (2007). DOI 10.1145/1298406.1298426. URL http://doi.acm.org/10.1145/1298406.1298426
15. Furnas, G.W., Landauer, T.K., Gomez, L.M., Dumais, S.T.: The vocabulary problem in human-system communication. Commun. ACM **30**(11), 964–971 (1987). DOI 10.1145/32206.32212. URL http://doi.acm.org/10.1145/32206.32212
16. Gallardo-Valencia, R.E., Elliott Sim, S.: Internet-scale code search. In: Proceedings of the 2009 Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation, SUITE
17. Gollapudi, S., Ieong, S., Ntoulas, A., Paparizos, S.: Efficient query rewrite for structured web queries. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, pp. 2417–2420. ACM, New York, NY, USA (2011). DOI 10.1145/2063576.2063981. URL http://doi.acm.org/10.1145/2063576.2063981
18. Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., Cumby, C.: A search engine for finding highly relevant applications. In: Software Engineering, 2010 ACM/IEEE 32nd International Conference on, vol. 1, pp. 475–484 (2010). DOI 10.1145/1806799.1806868
19. Gu, X., Zhang, H., Zhang, D., Kim, S.: Deep api learning. In: International Symposium on Foundations of Software Engineering (FSE) (2016)
20. Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., Menzies, T.: Automatic Query Reformulations for Text Retrieval in Software Engineering. In: Proceedings ICSE (2013)
21. Haiduc, S., De Rosa, G., Bavota, G., Oliveto, R., De Lucia, A., Marcus, A.: Query quality prediction and reformulation for source code search: The refoqus tool. In: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, pp. 1307–1310. IEEE Press, Piscataway, NJ, USA (2013). URL http://dl.acm.org/citation.cfm?id=2486788.2486991
22. Hill, E., Roldan-Vega, M., Fails, J.A., Mallet, G.: Nl-based query refinement and contextualized code search results: A user study. In: 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014, pp. 34–43 (2014). DOI 10.1109/CSMR-WCRE.2014.6747190. URL http://dx.doi.org/10.1109/CSMR-WCRE.2014.6747190
23. Hoffmann, R., Fogarty, J., Weld, D.S.: Assieme: finding and leveraging implicit references in a web search interface for programmers. In: Proceedings of the 20th annual ACM symposium on User interface software and technology, pp. 13–22. ACM (2007)
24. Holmes, R., Murphy, G.C.: Using structural context to recommend source code examples. In: Proceedings of ICSE. ACM (2005)
25. Keivanloo, I., Rilling, J., Zou, Y.: Spotting working code examples. In: Proceedings of ICSE (2014)
26. Kim, S., Kim, D.: Automatic identifier inconsistency detection using code dictionary. Empirical Software Engineering pp. 1–40 (2015)
27. Liu, L.M., Halper, M., Geller, J., Perl, Y.: Controlled vocabularies in oodbs: Modeling issues and implementation. Distrib. Parallel Databases **7**(1), 37–65 (1999). DOI 10.1023/A:1008682210559. URL http://dx.doi.org/10.1023/A:1008682210559
28. Lozano, A., Kellens, A., Mens, K.: Mendel: Source code recommendation based on a genetic metaphor. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, ASE '11, pp. 384–387. IEEE Computer Society, Washington, DC, USA (2011). DOI 10.1109/ASE.2011.6100078. URL http://dx.doi.org/10.1109/ASE.2011.6100078

29. Lv, F., Zhang, H., guang Lou, J., Wang, S., Zhang, D., Zhao, J.: Codehow: Effective code search based on api understanding and extended boolean model (e). In: 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 260–270 (2015)
30. Mamykina, L., Manoim, B., Mittal, M., Hripcsak, G., Hartmann, B.: Design lessons from the fastest q&a site in the west. In: Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 2857–2866. ACM (2011)
31. Mandelin, D., Xu, L., Bodík, R., Kimelman, D.: Jungloid mining: helping to navigate the api jungle. ACM SIGPLAN Notices **40**(6), 48–61 (2005)
32. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)
33. McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C., Xie, Q.: Exemplar: A source code search engine for finding highly relevant applications. IEEE Transactions on Software Engineering **38**(5), 1069–1087 (2012). DOI http://doi.ieeecomputersociety.org/10.1109/TSE.2011.84
34. McMillan, C., Grechanik, M., Poshyvanyk, D., Xie, Q., Fu, C.: Portfolio: Finding relevant functions and their usage. In: Proceedings of ICSE (2011)
35. Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A.: How can i use this method? In: ICSE (2015)
36. Nasehi, S.M., Sillito, J., Maurer, F., Burns, C.: What makes a good code example?: A study of programming q&a in stackoverflow. In: Software Maintenance (ICSM), 2012 28th IEEE International Conference on, pp. 25–34. IEEE (2012)
37. Nguyen, A.T., Nguyen, T.T., Al-Kofahi, J., Nguyen, H.V., Nguyen, T.: A topic-based approach for narrowing the search space of buggy files from a bug report. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 263–272 (2011). DOI 10.1109/ASE.2011.6100062
38. Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., Lanza, M.: Mining stackoverflow to turn the ide into a self-confident programming prompter. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 102–111. ACM (2014)
39. Ruthven, I.: Re-examining the potential effectiveness of interactive query expansion. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, SIGIR '03, pp. 213–220. ACM, New York, NY, USA (2003). DOI 10.1145/860435.860475. URL http://doi.acm.org/10.1145/860435.860475
40. Sadowski, C., Stolee, K.T., Elbaum, S.: How developers search for code: A case study. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 191–201. ACM, New York, NY, USA (2015). DOI 10.1145/2786805.2786855. URL http://doi.acm.org/10.1145/2786805.2786855
41. Stylos, J., Myers, B.A.: Mica: A web-search tool for finding api components and examples. In: Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on, pp. 195–202 (2006). DOI 10.1109/VLHCC.2006.32
42. Subramanian, S., Inozemtseva, L., Holmes, R.: Live api documentation. In: Proceedings of the 36th International Conference on Software Engineering, pp. 643–652. ACM (2014)
43. Thummalapenta, S., Xie, T.: 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 (2007)
44. Xie, T., Pei, J.: Mapo: Mining api usages from open source repositories. In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR '06, pp. 54–57. ACM, New York, NY, USA (2006). DOI 10.1145/1137983.1137997. URL http://doi.acm.org/10.1145/1137983.1137997
45. Zhao, L., Callan, J.: Term necessity prediction. In: Proceedings of the 19th ACM International Conference on Information and Knowledge Management, CIKM (2010)
46. Zhao, L., Callan, J.: Automatic term mismatch diagnosis for selective query expansion. In: Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR (2012)