CodeAid: Recommending Related Code

***Abstract*—Finding similar code for a given code query can help programmers detect situations that they had overlooked, or that they did not know how to solve. Most code-to-code search tools aim at finding syntactically or semantically similar code given some code of interest. We take a different approach to code-to-code search that recommends relevant *auxiliary* or *complementary* code. Such relevant code is not semantically similar to the code query, but could be important to work together with the given code to accomplish a complete or related functionality.**

In this paper, we describe a code recommendation technique, CodeAid, that returns code fragments that are related to a given code query in a code corpus. Specifically, we use GitHub Java projects as our corpus and focus on method-level code retrieval. Given a code fragment of interest, we use a code clone detection tool to detect all its similar counterparts across different GitHub files, and then recommend other related methods based on co-occurrence statistics.

In order to evaluate the effectiveness of CodeAid, we use Stack Overflow (SO) code snippets as queries, and measure the precision of the retrieved related methods. We gather a query code base of 21,207 SO code snippets written in Java. For 11,110 of these snippets, CodeAid discovered related code fragments in GitHub and recommended to developers. We then manually evaluate the relevance of recommended GitHub code. Using this methodology, CodeAid has a precision of 75.6%. We also performan an in-depth analysis on a sample of these code snippets and categorize the relationship between a SO query snippet and recommended code, shedding light on how CodeAid can be useful in practice.

***Index Terms*—code recommendation, related code, code search, mining software repositories**

I. Introduction

Over the past decade, code search has emerged as an interesting, but challenging, topic to both industry and research communities. Various code search techniques have been proposed in the literature [9], [15], [18], [25], and some search code engines have been implemented and are, or were, publicly available [1]–[6]. All of these code search engines take some specification as input (a query, a code fragment, or a test) and retrieve pieces of code that try to match that specification.

This work addresses the code search problem from a different angle. Consider the following scenario. A programmer is implementing a Java method for file decompression; this method gets as input the path to a zip file unpacks all files within the zip into a target directory, as shown in Listing 1. This piece of code is sufficient for a simple program task of unpacking a zip file. However, in pratice, the programmer may undertake a more complex programming task where unzipping a file is a small, integral part. Therefore, the programmer may also want to know what else may be related to this functionality. The programmer may also want to have a comprehensive understanding of the collection of related functions for zip file manipulation, in general. If an additional functionality often co-occurs with unzipping, the programmer may want to add it to her own project as needed. Listing 2 shosws an example of this kind of additional functionalities—a method that zips a list of files from a folder into the target zip file. Unzipping and zipping are two kinds of file manipulation in the opposite direction. Though these two functions work independently, they are often implemented together in a codebase to facilitate possible needs from any direction. We consider the zip method and the unzip method *related* to each other, or *complementary code fragement*, to be more specifically.

```
public static boolean unpackZip(String path, String
     zipname, String targetDirectory) {
     InputStream is;
     ZipInputStream zis;
     try {
          String filename;
          is = new FileInputStream(path + zipname);
          zis = new ZipInputStream(new
               BufferedInputStream(is));
          ZipEntry ze;
          byte[] buffer = new byte[1024];
          int count;

          while ((ze = zis.getNextEntry()) != null) {
               filename = ze.getName();

               if (ze.isDirectory()) {
                    File fmd = new File(targetDirectory
                         + filename);
                    fmd.mkdirs();
                    continue;
               }

               FileOutputStream fout = new
                   FileOutputStream(targetDirectory +
                   filename);

               while ((count = zis.read(buffer)) != -1) {
                    fout.write(buffer, 0, count);
               }

               fout.close();
               zis.closeEntry();
          }

          zis.close();
     } catch (IOException e) {
          e.printStackTrace();
          return false;
     }

     return true;
}
```

Listing 1. Query code

```
public static void zip(String baseFolder, List<File> files,
     String zipFile) {
     try {
          BufferedInputStream origin = null;
          FileOutputStream dest = new
               FileOutputStream(zipFile);

          ZipOutputStream out = new ZipOutputStream(new
               BufferedOutputStream(dest));
          byte data[] = new byte[BUFFER];
```

```
            for (File file : files) {
                    FileInputStream fi = new
                         FileInputStream(file);
                    origin = new BufferedInputStream(fi,
                         BUFFER);
                    String relativeFileName =
                         file.getAbsolutePath().replace(baseFolder
                         + File.separator , """");
                    ZipEntry entry = new
                         ZipEntry(relativeFileName);
                    out.putNextEntry(entry);
                    int count;
                    while ((count = origin.read(data, 0,
                         BUFFER)) != -1) {
                           out.write(data, 0, count);
                    }
                    origin.close();
            }

            out.close();
        } catch(Exception e) {
            e.printStackTrace();
        }
}
```

Listing 2. Recommended related code

Code-to-code search engines could potentially be used to retrieve releveant code examples to a given code query [4], [6], [14]. However, these techniques aim at finding syntactically or semantically similar code fragments only, without considering about auxiliary or complementary functionality. For instance, given an unzip function, they cannot find a complementary zip function, since neither the implementation nor the functionality of these two operations are similar. However, as discussed earlier, such complementary methods often co-occur in the same source file or the same codebase, which serves as an interesting property to exploit for recommending relevant code examples.

Pattern-based code completion tools [22], [23] also recommend completing code for a given code query. They do so by mining common API usage patterns from a large code corpus. For a given partial snippet as query, if it matches a prefix of a mined pattern, the tool recommends the rest of the pattern for completion. Again, such tools only work for the mined patterns; that is, they do not recommend code outside the mined patterns.

For both code search engines and pattern-based code completion tools, the retrieved code snippets may have extra lines of code with more functionality, but they are not designed to search for commonly used additional code.

The goal of our work is to support the search needs shown in Listings 1 and 2. In this paper, we describe `CodeAid`, a recommendation engine for related code. Given a code snippet as input query and a large corpus of code containing millions of code fragments, `CodeAid` returns a set of recommended code fragments such that:

- the recommended code fragments co-occur with similar counterparts of the input query.
- the recommended code fragments are ranked by their relevance as complements to the input query.

For the time being, we focus on method-level code fragments written in Java. Both the query snippet and the recommended related code fragments are Java methods. `CodeAid` works by first tokenizing the query and all methods in the code corpus. It then uses token similarity to detect similar counterparts to the query in the code corpus. We delegate this process to a clone detection tool, SourcererCC [27]. Finally, `CodeAid` recognizes other methods which co-occur with these similar counterparts as candidate related methods.

`CodeAid` has the following properties:

- It retrieves functionality groups, which the user may want to implement together with the input query.
- It has a clustering algorithm on top of co-occurrence to provide ranking.
- It is not restricted to any programming language. The Java parser is only used to chunk the file into methods, it can be replaced by the parser from any languages as needed.
- It has flexible granularity level. We can chunk the files into blocks of any size. All similarity comparison processes are token-based, which means as long as we have the token list representing the block, it does not matter what size the block is.
- It is fast enough to be used in real time. The most time-consuming part is similar code detection. However, generating the indexes is a one-time task and can be done before any query is processed.

In order to evaluate `CodeAid`, we use two datasets: (1) a query dataset consisting of 21,207 Java code snippets collected from Stack Overflow (SO), and (2) a large Java code base consisting of 50,826 projects which have at least five starts collected from GitHub, with over 5.8M distinct Java files. We then run each query in the query dataset through `CodeAid` and collect the recommended fragments coming from the code base. We present both a quantitative and qualitative analysis of the results. We build a chrome extension for Stack Overflow as a proof of concept and also to help with the qualitative evaluation.

The rest of the paper is organized as follows: Section II describes the algorithm `CodeAid` uses to generate recommendations. Section IV presents the manual analysis result of recommended code by `CodeAid` in terms of code relevance. We also compare `CodeAid` with existing code search engines in this section. V illustrates the instantiation of `CodeAid` as a chrome extension as well as a use scenario in a web browser. Section VI describes and Section VII concludes the paper.

II. Approach

`CodeAid` takes a code fragment as input and searches a code corpus to identify related code fragments. Given a user-selected code fragment, `CodeAid` first detects its similar methods in the corpus based on syntactic similarity. Then `CodeAid` traces back to the containing files of these similar methods and identifies other co-occurring methods in these files as candidate related methods. For each candidate related method, we further measure its similarity to methods in other files, and cluster similar methods. Then we rank candidate related methods based on the size of cluster it centers. Figure 1

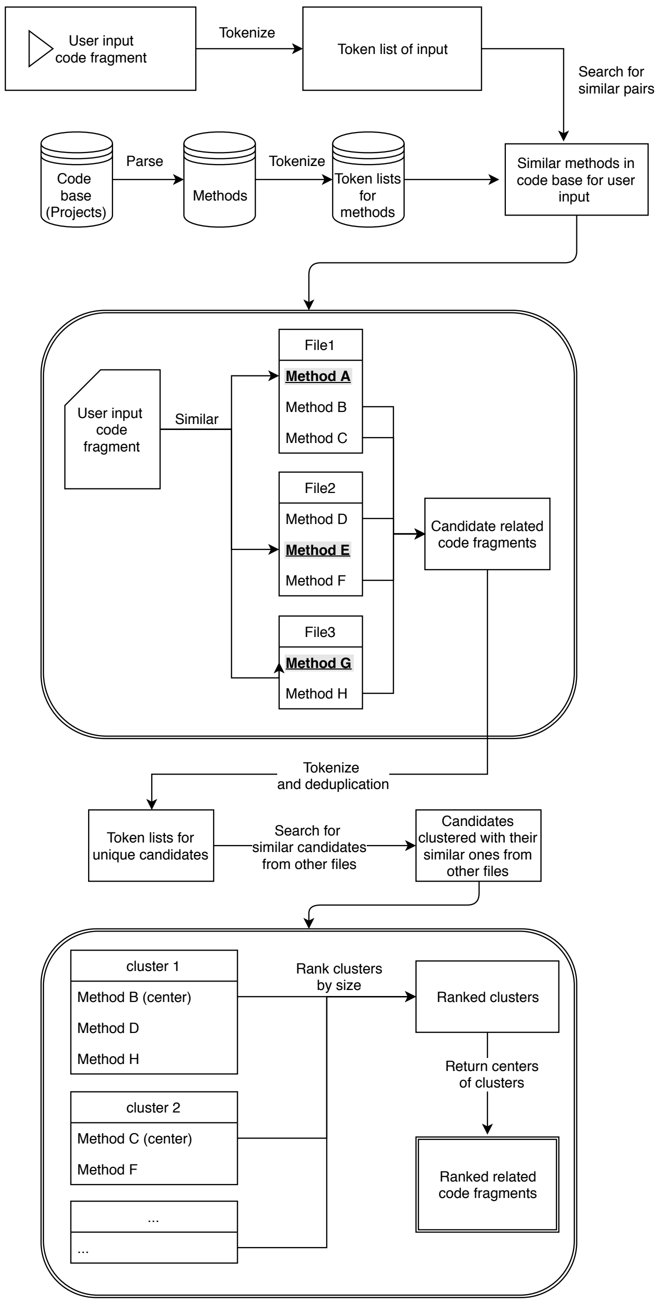


Fig. 1. The pipeline of CodeAid

describes the pipeline of finding related code fragments in `CodeAid`.

A. Retrieve similar methods

1) Parse a code corpus: We focus on method-level code fragments written in Java in this work. We parse all Java source files to abstract syntax trees (ASTs) and traverse the ASTs to extract all defined methods. Note that the approach is not limited to any programming language. We can switch to any other language by using its particular parser. We use the phrases code fragments and methods interchangeably in the paper.

2) Tokenization: Tokenization is the process of transforming source code into a bag of words. Tokenization starts from removing comments, spaces, tabs and other special characters. Then it identifies distinct tokens and count their frequencies. For each method, the result of tokenization is formated as a list of tuples such as `(token, freq)`, where the first element is a token in the method and the second element refers to the token occurrence in the method.

We tokenize both the input code fragment and all methods in the code corpus, in preparation for the next step of finding similar pairs.

3) Search for similar methods: For the input code fragment, we retrieve its similar counterparts from the code corpus using a token-based clone detection tool called SoucererCC [27]. By evaluating the scalability, execution time, recall and precision of SourcererCC, and comparing it to publicly available and state-of-the-art tools, SourcererCC has been shown to have both high recall and precision, and is able to scale to a large repository using a standard workstation. All of the above make SourcererCC a good candidate for building our code recommendation engine.

We use 70% similarity threshold, because it yields the best precision and recall on multiple clone benchmarks [27]. SourcererCC takes the token lists of the input code fragment and all methods in the code corpus, and returns the similar methods to the input in the code corpus. As shown in Figure 1, the user input has three similar counterparts in our code corpus, which are `Method A` in `File1`, `Method E` in `File2`, and `Method G` in `File3`.

B. Identify co-occurring code fragments

Given those similar code fragments identified in the previous step, we trace back to the files that contain these similar counterparts and identify co-occurring methods in the same file as a potentially related code fragment. Algorithm 1 gives a more formal description of the process.

Algorithm 1: Identify co-ocurring code fragments

```
Data: similar methods
Result: co-occurring methods
initialize $resultList$;
for $m_s$ in $similarMethods$ do
    $ghFiles$ = traceGitHubFiles(method) ;
    for $f$ in $ghFiles$ do
        $methodsInFile$ = parse($f$);
        for $m_f$ in $methodsInFile$ do
            if $m_f$ is not $m_s$ then
                $resultList$.add($m_f$) ;
            end
        end
    end
end
```

`Method A, E, G` are the three similar methods detected by SourcererCC. `File 1, 2, 3` are the three GitHub files contain these similar methods respectively. `File1` also contains `Method B, C`, `File2` has another two methods `Method D, F`, and `Method H` is `File3`. Therefore, `Method B, C, D, F, H` will be returned as co-occuring methods by Algorithm 1 and they are taken as our candidate for related code fragments.

C. Clustering and Ranking

1) Cluster candidate related code fragments: We further get the token lists for those candidate related methods identified in the previous step and remove duplicate candidates. In order to detect common co-occurring code fragments, we cluster the remaining unique candidate related methods based on their token similarity. Given each candidate related method, we compute its similarity to other candidates from different GitHub files. Each candidate will serve as the center of a cluster, we browse among other candidates from different files and add similar candidates to the current cluster.

Algorithm 2: Clustering candidate related code fragments

```
Data: $n$ candidate related methods
Result: clustered candidate methods
initialize $clusters = \{X_1, X_2, ..., X_n\}$;
for $m_i$ in $candidateMethods$ do
    $X_i$.add($m_i$) ;
    for $m_j$ in $candidateMethods$ do
        if $m_i$ and $m_j$ do not come from the same file then
            if tokenSimilarity($m_i$, $m_j$) > 0.7) then
                $X_i$.add($m_j$);
            end
        end
    end
end
```

For the candidate pool, `Method B, C, D, F, H`, each candidate will be the center of a cluster. For `Method B`, we compute token similarity with `Method D, F, H` and get two similar ones, `Method D, H`, so we add these two similar methods to the cluster, resulting in cluster size being three. Similarly, we add `Method F` to the cluster centered by `Method C` and get a cluster with size two.

2) Screen and rank clusters by size: After getting the candidate clusters, we keep only clusters with size being at least two. This means the center of the cluster has occurred at least twice among the GitHub files. We rank the remaining clusters by size and return the cluster centers as our final list of related code fragments with ranking. We will recommend `Method B` first, and then `Method C`, as our related code fragments.

III. Data Collection

We apply our approach to Stack Overflow (SO) and GitHub. We use code snippets in SO as the pool of user-input queries and use Java projects in GitHub as our code corpus to search from. We choose these two datasets not only because of their popularity within the programming community, but also because they are part of a larger system of software production. The same users that rely on the hosting and management characteristics of GitHub often have difficulties and need help on the implementation of their computer programs, seek support on SO for their specific problems, or hints of solutions from ones with a degree of similarity, and return to GitHub to apply the knowledge acquired.

Previous work have shown that developers often copy and paste code snippets from Stack Overflow to their GitHub projects and make adaptations as needed [8], [30], [32], [34]. Our approach will facilicate such opportunistic code reuse process when developers browse code snippets in SO. The use scenario will be: when a user is interested in a code snippet in SO, CodeAid recommends related code fragments from GitHub, showing what other code they may also want to investigate and integrate into their own project.

A. GitHub

We downloaded Java projects on GitHub by querying GHTorrent [11]. GHTorrent is a scalable, offline mirror of data offered through the GitHub REST API, available to the research community as a service. It provides access to all the metadata of GitHub projects, e.g., the clone url, the number of stars and committers, main programming languages in a project, etc. We use these metadata to screen the projects. Our project selection criteria are:

- We only consider GitHub projects that have at least five stars, in order to avoid toy projects that do not adequately reflect software engineering practices [13].
- We only keep non-forked projects, because project forking leads to many identical projects and would unnecessarily skew our recommendation.
- Prior work on GitHub cloning finds many identical files among GitHub projects, since developers may copy the whole file into another project without making any changes [16]. To account for this internal duplication in GitHub, we remove duplicated GitHub files using the same file hashing method as in [16].

As a result, we downloaded 50,826 non-forked Java repositories with at least five stars from GitHub. After deduplication, 5,825,727 distinct Java files remain.

B. Stack Overflow

We downloaded the SO dump taken in October 2016 [7]. From the data dump, we extract code snippets in the markdown `<code>` from SO posts with `java` or `android` tags. We consider code snippets in answer posts only, since snippets in question posts are rarely used as valid code examples. This results in 312,219 Java and Android answer posts.

Since SO snippets are often free-standing statements with low parsable rates [31], we used a customized pre-processor before tokenization. For free-standing statements, we wraps them with dummy class and method definitions, and add semicolons after statements as needed. For snippets contain multiple methods, we chunk them into individual ones. We keep only parsable SO snippets after pre-processing.

Prior work finds that larger SO snippets have more meaningful clones in GitHub [32]. Hence, we choose to study SO snippets with no less than 50 tokens after tokenization. We also remove duplicated examples within SO. As a result, we collect 186,392 distinct SO snippets.

C. *Result for similar code detection*

We run SoucererCC to find all similar pairs between SO and GitHub. We run on a server machine with 116 cores and 256G RAM. It takes 24 hours to complete. As a result, we get 21,207 distinct SO methods that have one or more similar code fragments in GitHub. We take these 21,207 SO snippets as our query code base.

IV. Evaluation

In this section, we describe the design of the assessment scenarios for `CodeAid` and report the evaluation results. Specifically, our experiments aim to address the following research questions:

- **RQ1**: Can `CodeAid` retrieve related code fragments for the queries in query code base?
- **RQ2**: Are the code fragments recommended by `CodeAid` related to the query?
- **RQ3**: What kinds of related code fragments do `CodeAid` recommend?
- **RQ4**: Can we get the recommended related code fragments from code search engines?

A. *Quantitative analysis*

We use 21,207 SO snippets as our query code base and get their similar counterparts in GitHub. The SO snippets have a median of two GitHub clones and a mean of one GitHub clones. The distribution of number of GitHub clones is shown in Figure 2. From the distribution we can see that SO snippets most commonly have zero to five similar counterparts in GitHub. Most of the SO snippets have less than twenty GitHub clones.

We collect the original GitHub files which contain these similar counterparts Then we extract all co-occurred methods from these GitHub files and treat them as candidate related code fragments. For each candidate in each GitHub file, we cluster its similar counterparts from other files. We keep only the clusters with size of at least two and return the remaining clusters in descending order of size.

For 11,110 out of 21K SO queries, `CodeAid` can retrieve related code fragments from GitHub. That is, using our SO query code base and GitHub search code corpus, `CodeAid` can recommend related code for 52.4% of the queries. The SO queries have a median of 24 recommended related methods in GitHub and a mean of 74 recommended related methods. The related methods have a median of 12 average lines of code, and a mean of 14 average lines of code. The distribution of number of related methods and distribution of average lines of code are shown in Figure 3 and 4 respectively.

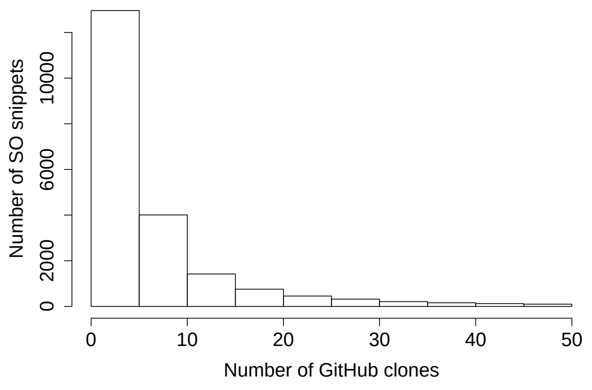


Fig. 2. Distribution of number of GitHub clones

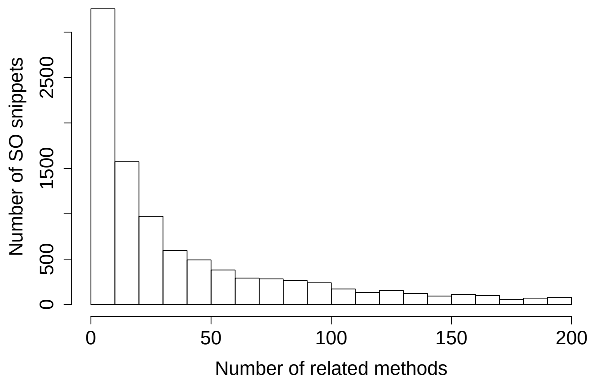


Fig. 3. Distribution of number of related methods

From the figures we can see that a SO query most commonly have less than ten recommendations for related code fragments, and most of the SO queries have less than 50 related methods in GitHub. Most related methods for a query will have five to thirty average lines of code.

B. *Manual analysis and categorization*

We randomly select 30 SO snippets with its recommended related code fragments from the 11,110 groups, and manually examine whether the recommended code fragments are related to the SO input or not, and categorize why we call the relationship a relevant one.

We use $Precision@k$ metric to evaluate `CodeAid` which is defined as follows:

$$Precision@k = \frac{1}{N}\sum_{i=1}^{N}\frac{|relevant_{i,k}|}{k} \quad (1)$$

where $|relevant_{i,k}|$ represents the number of positive related results in the top k results for query i, N is the number queries we evaluate, which is 30. k is the number of top results we examine, here we use $k = 1$ and $k = 3$.

TABLE I
COMPLEMENTARY METHOD EXAMPLES

Query Code Snippet | Recommended Related Code

```
public static byte[] encrypt(final SecretKeySpec key, final byte[] iv, final byte[]
        message)
throws GeneralSecurityException {
      final Cipher cipher = Cipher.getInstance(AES_MODE);
      IvParameterSpec ivSpec = new IvParameterSpec(iv);
      cipher.init(Cipher.ENCRYPT_MODE, key, ivSpec);
      byte[] cipherText = cipher.doFinal(message);

      log(""cipherText"", cipherText);

      return cipherText;
}
```

```
public static byte[] decrypt(final SecretKeySpec key, final byte[] iv, final byte[]
        decodedCipherText)
throws GeneralSecurityException {
      final Cipher cipher = Cipher.getInstance(AES_MODE);
      IvParameterSpec ivSpec = new IvParameterSpec(iv);
      cipher.init(Cipher.DECRYPT_MODE, key, ivSpec);
      byte[] decryptedBytes = cipher.doFinal(decodedCipherText);

      log(""decryptedBytes"", decryptedBytes);

      return decryptedBytes;
}
```

Example B: Complementary method

- The query snippet implements `encrypt` functionality for an byte array.
- The recommended related method decrypts a decoded byte array.

```
@Override
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      preferred = (TextView)findViewById(R.id.preferred);
      orientation = (TextView)findViewById(R.id.orientation);
      mgr = (SensorManager) this.getSystemService(SENSOR_SERVICE);
      accel = mgr.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
      compass = mgr.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
      orient = mgr.getDefaultSensor(Sensor.TYPE_ORIENTATION);
      WindowManager window = (WindowManager)
      this.getSystemService(WINDOW_SERVICE);
      int apiLevel = Integer.parseInt(Build.VERSION.SDK);
      if(apiLevel <8) {
            mRotation = window.getDefaultDisplay().getOrientation();
      }
      else {
            mRotation = window.getDefaultDisplay().getRotation();
      }
      }
```

```
@Override
protected void onPause() {
      mgr.unregisterListener(this, accel);
      mgr.unregisterListener(this, compass);
      mgr.unregisterListener(this, orient);
      super.onPause();
}
```

Example C: Complementary method

- The query snippet implements `onCreate` functionality for an `Android` app activity.
- The recommended related method implements `onPause` which does not have direct function call with the query snippet, but adds extra functionality to the activity.

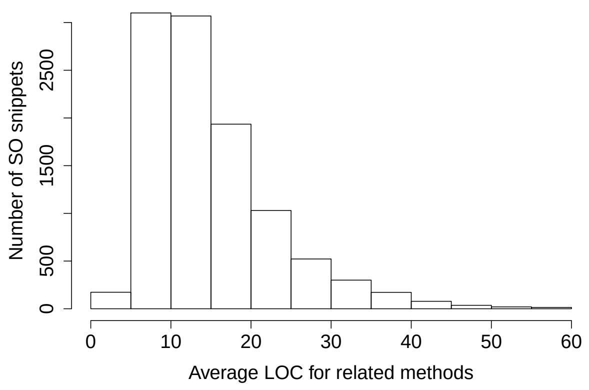


Fig. 4. Distribution of average LOC of related methods

`CodeAid` achieves 80% and 75.6% for $Precision@1$ and $Precision@3$ respectively. That is to say, for the 30 top 1 recommended results, 24 of them are manually examined as related, for the 90 top 3 recommended results, 68 of them are related.

We find the following types of relevance in our sample set:

- A complementary method that adds more functionality
- A supplmentary method that helps with, or gets help from, the query
- A different implementation for the query

1) Complementary method: In this category, the query code can function alone, but the recommended related method provides extra functionality to the query code and will further complete the user class. For the example shown as Listing 1 and 2 in Section I, the query snippet implements unzip a folder in Java. `CodeAid` recommends `zip` function. These two methods can function independently, but often implemented together to get a stronger ability for file manipulation.

Similarly, `CodeAid` recommends `decrypt` function for `encrypt` and `onPause` function for `onCreate` in Table I. The two methods in each pair do not have any direct function call association between them, but they complete each other with extra functionality and are often implemented together in real-life scenarios.

2) Supplementary method: The recommended related code serves as a helper function to the query, or vice versa. One may make function call to the other. For example the `merge` function for `sort`. `sort` calls `merge` as a helper function and cannot achieve functionality without it.

In our first example in Table II, our recommended related code `loadDrawable` calls `queueJob` inside its method body. There is another related method being recommended together with `loadDrawable`, which is shown below in Listings 3. `loadDrawable` also makes a function call to `getDrawableFromCache` inside its method body, The related methods give the user a broader picture of the whole class, point to a higher level of functionality the user may want to implement, and also direct the user to the most-frequently used

TABLE II
SUPPLEMENTARY METHOD EXAMPLES

Query Code Snippet | Recommended Related Code

```
private void queueJob(final String url, final ImageView imageView,final Drawable
        placeholder) {
        /* Create handler in UI thread. */
        final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {
                String tag = mImageViews.get(imageView);
                        if (tag != null && tag.equals(url)) {
                                if (imageView.isShown())
                                        if (msg.obj != null) {
                                                imageView.setImageDrawable((Drawable) msg.obj);
                                        } else {
                                        imageView.setImageDrawable(placeholder);
                                        //Log.d(null, "fail " + url);
                                        }
                                }
                        }
        };

        mThreadPool.submit(new Runnable() {
                @Override
                public void run() {
                        final Drawable bmp = downloadDrawable(url);
                        // if the view is not visible anymore, the image will be ready
                                for next time in cache
                        if (imageView.isShown())
                        {
                                Message message = Message.obtain();
                                message.obj = bmp;
                                //Log.d(null, "Item downloaded: " + url);

                                handler.sendMessage(message);
                        }
                }
        });
}
```

```
public void loadDrawable(final String url, final ImageView imageView) {
        imageViews.put(imageView, url);
        Drawable drawable = getDrawableFromCache(url);
        // check in UI thread, so no concurrency issues
        if (drawable != null) {
                Log.d(null, "Item loaded from cache: " + url);
                imageView.setImageDrawable(drawable);
        } else {
                imageView.setImageDrawable(placeholder);
                queueJob(url, imageView);
        }
}
```

Example D: Supplementary method

- The recommended related method calls the query snippet within its method body. It is a higher-level funtionality to the query.

```
@Override
protected void onLayout(boolean changed, int l, int t, int r, int b) {
        final int count = getChildCount();
        for (int i = 0; i < count; i++) {
                View child = getChildAt(i);
                LayoutParams lp = (LayoutParams) child.getLayoutParams();
                child.layout(lp.x+5, lp.y+5, lp.x + child.getMeasuredWidth(), lp.y +
                        child.getMeasuredHeight());
        }
}
```

```
@Override
protected LayoutParams generateLayoutParams(ViewGroup.LayoutParams p) {
        return new LayoutParams(p);
}
```

Example E: Supplementary method

- The recommended related code generates the `Layout parameters`, it will be traced by `getLayoutParam` function, which will further be called inside the query method `onLayout`. There is a dependency chain between the query and the related code.

TABLE III
DIFFERENT IMPLEMENTATION EXAMPLE

Query Code Snippet | Recommended Related Code

```
public static <K, V extends Comparable<? super V>> SortedSet<Map.Entry<K, V>>
        entriesSortedByValues(Map<K, V> map) {
        SortedSet<Map.Entry<K, V>> sortedEntries = new TreeSet<Map.Entry<K, V>>(
                new Comparator<Map.Entry<K, V>>() {
                @Override
                        public int compare(Map.Entry<K, V> e1, Map.Entry<K, V> e2) {
                                return e1.getValue().compareTo(e2.getValue());
                        }
                });
        sortedEntries.addAll(map.entrySet());
        return sortedEntries;
}
```

```
public static <K, V extends Comparable<? super V>> Map<K, V> sortByValue( Map<K, V>
        map ) {
        List<Map.Entry<K, V>> list =
                new LinkedList<Map.Entry<K, V>>( map.entrySet() );
        Collections.sort( list, new Comparator<Map.Entry<K, V>>()
        {
                public int compare( Map.Entry<K, V> o1, Map.Entry<K, V> o2 )
                {
                        return (o1.getValue()).compareTo( o2.getValue() );
                }
        } );

        Map<K, V> result = new LinkedHashMap<K, V>();
        for (Map.Entry<K, V> entry : list)
        {
                result.put( entry.getKey(), entry.getValue() );
        }
        return result;
}
```

Example F: Different implementation

- The question title of the SO post is: Sort the values in HashMap.
- The query snippet from SO uses `SortedSet` to store the map entries, while the recommended code provides an alternative, using `LinkedList`, and show how to iterate the map.

TABLE IV
CATEGORIZATION OF RELATED METHODS

Category	Top 1	Top 3
Complementary method	12	33
Supplementary method	11	31
Different implementation	1	4
Not related	6	22

higher level functionality and its auxiliaries.

```
public static Drawable getDrawableFromCache(String url) {
      if (DrawableManager.cache.containsKey(url)) {
            return DrawableManager.cache.get(url);
      }

      return null;
}
```

Listing 3. Recommended code #2

3) Different implementation: This category represents those recommended related methods that have similar functionality to the query code. The recommended result provides an alternative, or a more detailed or extended implementation for the functionality. As shown in Table III, both of the methods implement sorting values in a `Map`, the query store the map entries in a `SortedSet`, while the recommended code uses `LinkedList`, and shows how to iterate a `Map`. For the `encrypt` function in Table I, `CodeAid` also recommended an alternative implementation with `String` inputs, as shown in Listing 4.

```
public static String encrypt(final String password, String message) throws
       GeneralSecurityException {
      try {
            final SecretKeySpec key = generateKey(password);
            log("message", message);
            byte[] cipherText = encrypt(key, ivBytes, message.getBytes(CHARSET));
            //NO_WRAP is important as was getting \n at the end
            String encoded = String.valueOf(
                  Base64.encodeToString(cipherText, Base64.NO_PADDING ));
            log("Base64.NO_WRAP", encoded);
            return encoded;
      } catch (UnsupportedEncodingException e) {
            if (DEBUG_LOG_ENABLED)
                  Log.e(TAG, "UnsupportedEncodingException ", e);
            throw new GeneralSecurityException(e);
      }
}
```

Listing 4. different implementation for `encrypt`

C. Comparison with code search engines

In this experiment we compare the recommendation results of `CodeAid` with those from code search engines. We choose Google search engine since its the most popular destination when people look for programming assistance. We also compare to `FaCoY` [14], a code-to-code search engine which proved to have state-of-art precision. It uses SO snippets as its query base and indexes GitHub files as search space. `searchcode` [6] and `Krugle` [4] are another two online code-to-code search engines. We only compare with `FaCoY` because it beats `searchcode` and `Krugle` in the total number of outputs and precision of outputs when using SO snippets as queries [14]. We look for whether the search engines can also retrieve top 1 related code fragments recommended by `CodeAid` in their top 10 search results.

For one out of the ten queries, `FaCoY` can return the related code recommended by `CodeAid` in its top 10 search results. This results from the related code being very similar to the query, as shown in Table V. For the rest of nine queries, the related code is not similar to the query, so it cannot be retrieved by `FaCoY`.

As for Google search engine, for five out of the ten queries, Google can locate the GitHub file(s) which contain similar methods to the query, therefore we can find the related code recommended by `CodeAid` inside these GitHub files. However, Google can only retrieve the full files, while `CodeAid` can point to the method which is mostly used among these files.

By performing the comparison above, we can see that code search engines may not fulfill the purpose of recommending related code as `CodeAid`.

V. CHROME EXTENSION FOR STACK OVERFLOW

In the purpose of helping with qualitative analysis, we build a Chrome extension for the Stack Overflow query code base and GitHub search code corpus. Figure 5 shows a screenshot of using the Chrome extension. The highlighted yellow snippet is the query method from SO, and on the right-hand side we present the ranked list of related code fragments from GitHub. The Chrome extension provides a clearer presentation of the query and the resulting related code in the process of manual evalutation.

The Chrome extension also demonstrates a real-life use scenario of getting related methods from GitHub for a SO snippet, thus serves as a proof of our concept of recommending related code fragments. The real-life use scenario will be: suppose the user is searching on Stack Overflow for the question `how to unzip a folder in java`. The user locates the highlighted yellow snippet as the correct implementation of the query functionality. They are interested in learning the other methods that can be possibly added to their project along with the `unzip` method. So the user invokes searching on `CodeAid` and gets recommended related methods. They may investigate the recommended results and select to add a `zip` method to complete the functionalities of file manipulation.

VI. RELATED WORK

Various code search techniques have been proposed to discover relevant code components (e.g., functions, code snippets) given a user query. For example, given a keyword query, Portfolio retrieves function definitions and their usages using a combination of a PageRank model and an association model [19]. Chan et al. improve Portfolio by matching the textual similarity between containing nodes in an API usage subgraph with a keyword query [10]. CodeHow also finds code snippets relevant to a natural language query. It explores API documents to identify relationships between query terms and APIs [17]. Instead of using natural language queries, several techniques automatically recommend relevant code snippets based on contextual information such as types in a target program [12], [24], [26], [29]. CodeGenie is a test-driven code search technique that allows developers to specify desired functionality via test cases and then matches relevant

TABLE V
RELATED CODE WHICH CAN BE RETRIEVED BY FACOY

Query Code Snippet	Recommended Related Code
see listing below	see listing below

Query Code Snippet

```
public int readFramesChanel(short[] sampleBuffer, int offset, int
      numFramesToRead,int channel) throws IOException, WavFileException
{
      if (ioState != IOState.READING) throw new IOException("Cannot read from
            WavFile instance");

      for (int f=0 ; f<numFramesToRead ; f++)
      {
            if (frameCounter == numFrames) return f;

            for (int c=0 ; c<numChannels ; c++)
            {
                  if(channel==c)
                  {
                        sampleBuffer[offset] = (short) readSample();
                        offset ++;
                  }
                  else
                        readSample();
            }

            frameCounter ++;
      }

      return numFramesToRead;
}
```

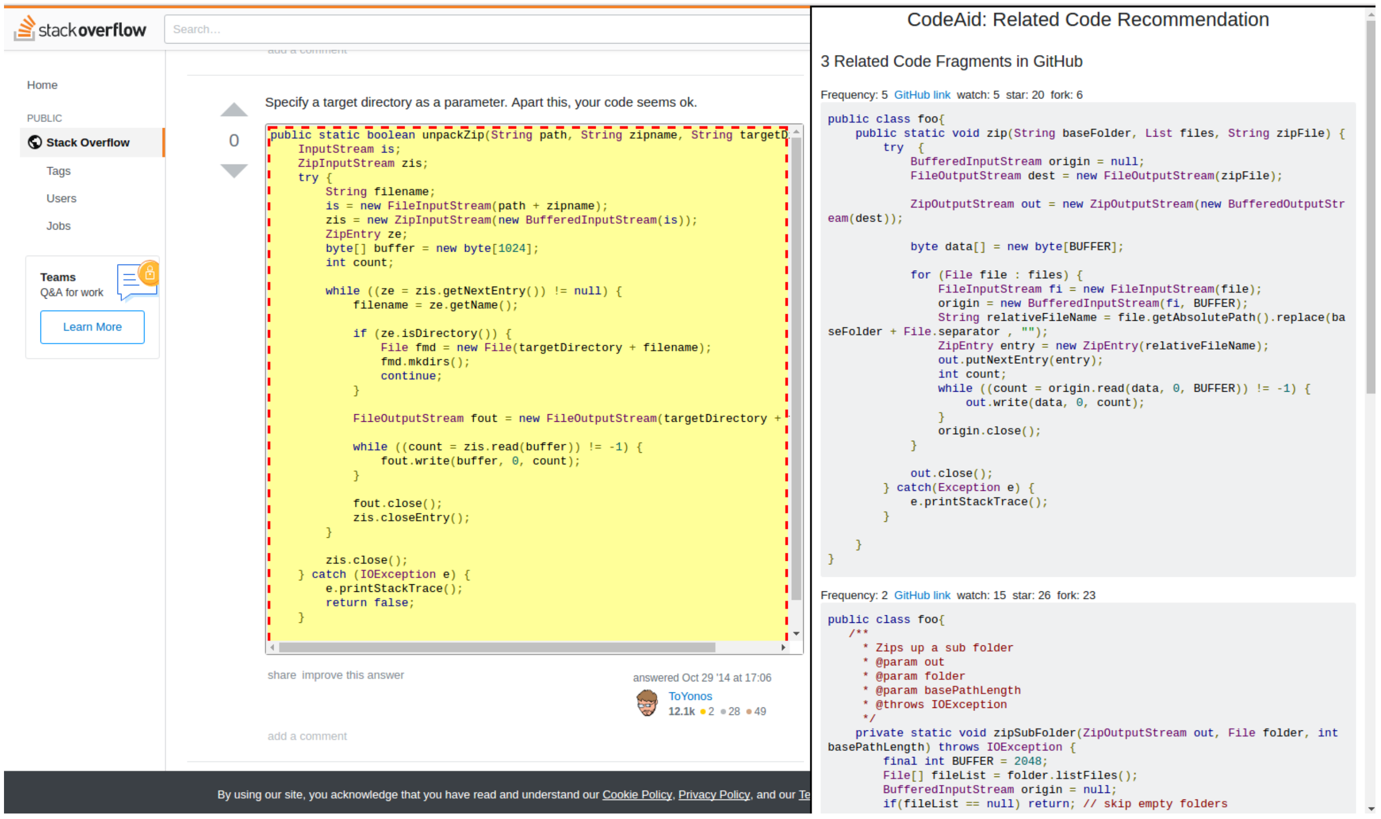
Recommended Related Code

```
public int writeFrames(int[] sampleBuffer, final int offSetIn, int
      numFramesToWrite) throws IOException
{
      if (this.ioState != IOState.WRITING) throw new IOException("Cannot write to
            WavFile instance"); //$NON-NLS-1$
      int offSet = offSetIn;
      for (int f = 0; f < numFramesToWrite; f++)
      {
            if (this.frameCounter == this.numFrames) return f;

            for (int c = 0; c < this.numChannels; c++)
            {
                  writeSample(sampleBuffer[offSet]);
                  offSet++;
            }

            this.frameCounter++;
      }

      return numFramesToWrite;
}
```



Specify a target directory as a parameter. Apart this, your code seems ok.

```
public static boolean unpackZip(String path, String zipname, String targetD
    InputStream is;
    ZipInputStream zis;
    try {
        String filename;
        is = new FileInputStream(path + zipname);
        zis = new ZipInputStream(new BufferedInputStream(is));
        ZipEntry ze;
        byte[] buffer = new byte[1024];
        int count;

        while ((ze = zis.getNextEntry()) != null) {
            filename = ze.getName();

            if (ze.isDirectory()) {
                File fmd = new File(targetDirectory + filename);
                fmd.mkdirs();
                continue;
            }

            FileOutputStream fout = new FileOutputStream(targetDirectory +

            while ((count = zis.read(buffer)) != -1) {
                fout.write(buffer, 0, count);
            }

            fout.close();
            zis.closeEntry();
        }

        zis.close();
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Te

CodeAid: Related Code Recommendation

3 Related Code Fragments in GitHub

Frequency: 5 GitHub link watch: 5 star: 20 fork: 6

```
public class foo{
    public static void zip(String baseFolder, List files, String zipFile) {
        try  {
            BufferedInputStream origin = null;
            FileOutputStream dest = new FileOutputStream(zipFile);

            ZipOutputStream out = new ZipOutputStream(new BufferedOutputStr
eam(dest));

            byte data[] = new byte[BUFFER];

            for (File file : files) {
                FileInputStream fi = new FileInputStream(file);
                origin = new BufferedInputStream(fi, BUFFER);
                String relativeFileName = file.getAbsolutePath().replace(ba
seFolder + File.separator , "");
                ZipEntry entry = new ZipEntry(relativeFileName);
                out.putNextEntry(entry);
                int count;
                while ((count = origin.read(data, 0, BUFFER)) != -1) {
                    out.write(data, 0, count);
                }
                origin.close();
            }

            out.close();
        } catch(Exception e) {
            e.printStackTrace();
        }

    }
}
```

Frequency: 2 GitHub link watch: 15 star: 26 fork: 23

```
public class foo{
   /**
     * Zips up a sub folder
     * @param out
     * @param folder
     * @param basePathLength
     * @throws IOException
     */
    private static void zipSubFolder(ZipOutputStream out, File folder, int
basePathLength) throws IOException {
        final int BUFFER = 2048;
        File[] fileList = folder.listFiles();
        BufferedInputStream origin = null;
        if(fileList == null) return; // skip empty folders
```

Fig. 5. Screenshot of Chrome extension

methods and classes with the given test [15]. To more precisely capture search intent, S6 allows developers to express desired functionality using a combination of input-output types, test cases, and keyword descriptions [25].

Code-to-code search tools are most related to our technique among all different kinds of code search techniques. Given a code snippet as input, FaCoY [14] finds semantically similar code snippets in a Stack Overflow dataset by first matching with accompanied natural language descriptions in related posts instead of matching code directly. Unlike FaCoY, several techniques infer an underlying code search pattern from a given code fragment [20], [21], [28], [33]. Sydit generalizes concrete identifiers (e.g., variable names, types, and method calls) in a given code example as an abstract code template and identifies other similar locations via AST-based tree matching [20]. Lase uses multiple code examples instead of a single example to better infer the search intent of a user [21]. Critics allows developers to construct an AST-based search pattern from a single example through manual code selection, customization, and parameterization [33]. These code-to-code search tools focus on identifying relevant code snippets that are syntactically or semantically similar to a given code snippet. However, since many programming tasks (e.g., password encryption and decryption) require multiple code snippets or functions to work together, none of the existing techniques recommend code snippets that complement a given code snippet to complete desired functionality.

VII. Conclusion

We have presented `CodeAid`, a code-to-code search technique and tool that recommends code fragments related to the code query. The information need that underlies the design of `CodeAid` is different from that of traditional code search engines: rather than retrieving similar code, it retrieves code within the functionality family of the code query. This broader context allows programmers to be aware of related code that may be useful when developing their own functions. `CodeAid` works by analyzing co-occurrences of code fragments in very large code bases, such as GitHub projects, and then clustering those potential matches for a given code query.

We evaluated `CodeAid` using a large dataset of code queries collected from Stack Overflow snippets, and a very large code base of Java projects collected from GitHub. The evaluation shows that `CodeAid` is able to find related fragments for 52.4% of the queries. A closer look at a set of 30 queries shows that `CodeAid` has a good precision of 75.6%. This is a promising result for a new approach to code search.

One limitation of `CodeAid` is that the recommendation power of `CodeAid` strongly correlates with the similarity behavior of the dataset. To retrieve related code fragments, `CodeAid` requires the query code to have similar counterparts in the search query corpus; it also asks for similar method pairs within search query corpus itself.

For future work, we may try different user-defined query code base and search code corpus to further evaluate `CodeAid`. For example the search code corpus can be the code base for a company or previous projects from a user. We may import the tool into a Eclipse plugin and help developers during real life programming process. The use scenario will be, when a user is currently writing a code fragment and uses it as query to search `CodeAid` plug-in, we will return the ranked list of related code from the company's code base or the user's own projects.

References

[1] Codase. http://www.codase.com/. Accessed: 2019-05-13.

[2] Github. https://github.com/. Accessed: 2019-05-13.

[3] Google code search. https://en.wikipedia.org/wiki/Google_Code_Search. Accessed: 2019-05-13.

[4] Krugle. http://krugle.com". Accessed: July 2017.

[5] Open hub. https://www.openhub.net/. Accessed: 2019-05-13.

[6] searchcode. https://searchcode.com/. Accessed: July 2017.

[7] Stack exchange archive. https://archive.org/details/stackexchange". Accessed: October 2016.

[8] Le An, Ons Mlouki, Foutse Khomh, and Giuliano Antoniol. Stack overflow: a code laundering platform? In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*, pages 283–293. IEEE, 2017.

[9] Sushil Bajracharya, Joel Ossher, and Cristina Lopes. Sourcerer: An internet-scale software repository. In *Proceedings of the 2009 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, pages 1–4. IEEE Computer Society, 2009.

[10] Wing-Kwan Chan, Hong Cheng, and David Lo. Searching connected api subgraph via text phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 10:1–10:11. ACM, 2012.

[11] Georgios Gousios and Diomidis Spinellis. Ghtorrent: Github's data from a firehose. In *Mining software repositories (msr), 2012 9th ieee working conference on*, pages 12–21. IEEE, 2012.

[12] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 27th International Conference on Software Engineering*, pages 117–125, New York, NY, USA, 2005. ACM Press.

[13] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101. ACM, 2014.

[14] Kisub Kim, Donsun Kim, Tegawend F. Bissyand, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. Facoy a code-to-code search engine. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, 2018.

[15] Otávio Augusto Lazzarini Lemos, Sushil Bajracharya, Joel Ossher, Paulo Cesar Masiero, and Cristina Lopes. Applying test-driven code search to the reuse of auxiliary functionality. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 476–482. ACM, 2009.

[16] Cristina V Lopes, Petr Maj, Pedro Martins, Vaibhav Saini, Di Yang, Jakub Zitny, Hitesh Sajnani, and Jan Vitek. Déjàvu: a map of code duplicates on github. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):84, 2017.

[17] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. Codehow: Effective code search based on api understanding and extended boolean model (e). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 260–270. IEEE, 2015.

[18] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Chen Fu, and Qing Xie. Exemplar: A source code search engine for finding highly relevant applications. *IEEE Transactions on Software Engineering*, 38(5):1069–1087, 2012.

[19] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. Portfolio: finding relevant functions and their usage. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 111–120. IEEE, 2011.

[20] Na Meng, Miryung Kim, and Kathrn S. McKinley. Systematic editing: Generating program transformations from an example. In *PLDI '11*, pages 329–342, San Jose, CA, 2011. ACM.

[21] Na Meng, Miryung Kim, and Kathryn S McKinley. Lase: locating and applying systematic edits by learning from examples. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 502–511. IEEE Press, 2013.

[22] Anh Tuan Nguyen, Tung Thanh Nguyen, Hoan Anh Nguyen, Ahmed Tamrawi, Hung Viet Nguyen, Jafar Al-Kofahi, and Tien N. Nguyen. Graph-based pattern-oriented, context-sensitive source code completion. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 69–79, Piscataway, NJ, USA, 2012. IEEE Press.

[23] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 383–392, New York, NY, USA, 2009. ACM.

[24] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th*

Working Conference on Mining Software Repositories, pages 102–111. ACM, 2014.

[25] Steven P Reiss. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering*, pages 243–253. IEEE Computer Society, 2009.

[26] Naiyana Sahavechaphan and Kajal Claypool. Xsnippet: mining for sample code. *ACM Sigplan Notices*, 41(10):413–430, 2006.

[27] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K Roy, and Cristina V Lopes. Sourcerercc: Scaling code clone detection to big-code. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, pages 1157–1168. IEEE, 2016.

[28] Aishwarya Sivaraman, Tianyi Zhang, Guy Van den Broeck, and Miryung Kim. Active inductive logic programming for code search. In *Proceedings of the 41th International Conference on Software Engineering*. IEEE Press, 2019.

[29] Suresh Thummalapenta and Tao Xie. 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*, pages 204–213. ACM, 2007.

[30] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. How do developers utilize source code from stack overflow? *Empirical Software Engineering*, pages 1–37, 2018.

[31] Di Yang, Aftab Hussain, and Cristina Videira Lopes. From query to usable code: an analysis of stack overflow code snippets. In *Proceedings of the 13th International Workshop on Mining Software Repositories*, pages 391–402. ACM, 2016.

[32] Di Yang, Pedro Martins, Vaibhav Saini, and Cristina Lopes. Stack overflow in github: any snippets there? In *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*, pages 280–290. IEEE, 2017.

[33] Tianyi Zhang, Myoungkyu Song, Joseph Pinedo, and Miryung Kim. Interactive code review for systematic changes. In *Proceedings of 37th IEEE/ACM International Conference on Software Engineering. IEEE*, 2015.

[34] Tianyi Zhang, Di Yang, Crista Lopes, and Miryung Kim. Analyzing and supporting adaptation of online code examples. In *Proceedings of the 41th International Conference on Software Engineering*. IEEE Press, 2019.