FACER: An API Usage-based Code-example Recommender for Opportunistic Reuse

Shamsa Abid (15030049@lums.edu.pk)
Lahore University of Management Sciences
Shafay Shamail
Lahore University of Management Sciences
Hamid Abdul Basit
Prince Sultan University
Sarah Nadi
University of Alberta

Research Article

Keywords: code recommendation, code search engine, software features, API usage, code clones

Posted Date: March 4th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-260432/v1

Version of Record: A version of this preprint was published at Empirical Software Engineering on August 18th, 2021. See the published version at https://doi.org/10.1007/s10664-021-10000-w.

Abstract

To save time, developers often search for code examples that implement their desired software features. Existing code search techniques typically focus on finding code snippets for a single given query, which means that developers need to perform a separate search for each desired functionality. In this paper, we pro-pose FACER (Feature-driven API usage-based Code Examples Recommender), a technique that avoids repeated searches through opportunistic reuse. Specifically, given the selected code snippet that matches the initial search query, FACER finds and suggests related code snippets that represent features that the developer may want to implement next. FACER first constructs a code fact repository by parsing the source code of open-source Java projects to obtain methods' textual information, call graphs, and Application Programming Interface (API) usages. It then detects unique features by clustering methods based on similar API us-ages, where each cluster represents a feature or functionality. Finally, it detects frequently co-occurring features across projects using frequent pattern mining and recommends related methods from the mined patterns. To evaluate FACER, we run it on 120 Java Android apps from GitHub. We first manually validate that the detected method clusters represent methods with similar functionality. We then perform an automated evaluation to determine the best parameters (e.g., similarity threshold) for FACER. We recruit 10 professional developers along with 39 experienced students to judge FACER's recommendation of related methods. Our results show that, on average, FACER's recommendations are 80% precise. We also survey a total of 20 professional Android and Java developers to understand their code search and reuse experiences, and also to obtain their feedback on the usability and usefulness of FACER. The survey results show that 95% of our surveyed professional developers find the idea of related method recommendations useful during code reuse.

Full Text

Due to technical limitations, full-text HTML conversion of this manuscript could not be completed. However, the latest manuscript can be downloaded and accessed as a PDF.

Figures

```
public static Intent getPickImageChooserIntent(
   @NonNull Context context, CharSequence title, boolean includeDocuments) {
       List<Intent> allIntents = new ArrayList<>();
       PackageManager packageManager = context.getPackageManager();
       if (!isExplicitCameraPermissionRequired(context)) {
           allIntents.addAll(getCameraIntents(context, packageManager));
       }
       List<Intent> galleryIntents = getGalleryIntents(
      packageManager, Intent.ACTION_GET_CONTENT, includeDocuments);
       if (galleryIntents.size() == 0) {
           galleryIntents = getGalleryIntents(packageManager,
           Intent.ACTION_PICK, includeDocuments);
       }
       allIntents.addAll(galleryIntents);
       Intent target;
       if (allIntents.isEmpty()) {
           target = new Intent();
       } else {
           target = allIntents.get(allIntents.size() - 1);
           allIntents.remove(allIntents.size() - 1);
       }
       Intent chooserIntent = Intent.createChooser(target, title);
       chooserIntent.putExtra(Intent.EXTRA_INITIAL_INTENTS,
        allIntents.toArray(new Parcelable[allIntents.size()]));
       return chooserIntent;
}
```

(a) Selected code snippet

```
private static Bitmap cropBitmapObjectWithScale(Bitmap bitmap, float[] points,
   int degreesRotated, boolean fixAspectRatio, int aspectRatioX, int aspectRatioY,
   float scale) {
   Rect rect = getRectFromPoints(points, bitmap.getWidth(), bitmap.getHeight(),
            fixAspectRatio, aspectRatioX, aspectRatioY);
   Matrix matrix = new Matrix();
   matrix.setScale(scale, scale);
   matrix.postRotate(degreesRotated, bitmap.getWidth() / 2, bitmap.getHeight() / 2);
   Bitmap result = Bitmap.createBitmap(bitmap, rect.left, rect.top,
           rect.width(), rect.height(), matrix, true);
   if (result == bitmap) {
     result = bitmap.copy(bitmap.getConfig(), false);
   }
   if (degreesRotated % 90 != 0) {
     result = cropForRotatedImage(result, points, rect, degreesRotated,
            fixAspectRatio, aspectRatioX, aspectRatioY);
   }
   return result;
}
```

(b) Crop image

```
   @Override
   public View getView(int i, View view, ViewGroup viewGroup)
   {
      ImageView imageView;
      if (view == null) {
      int gridWidth = fragment.getScreenWidth();
      imageView = new ImageView(mContext);
      imageView.setLayoutParams(
      new GridView.LayoutParams(gridWidth/5 - 30, gridWidth/5 - 30));
      imageView.setScaleType(ImageView.ScaleType.FIT_CENTER);
      imageView.setPadding(5, 5, 5, 5);
      }else {imageView = (ImageView) view;}
      Bitmap bmp = getResizedBitmap(loadImage(imageFileNames.get(i)), 200);
      imageView.setImageBitmap(bmp);
      return imageView;
   }
```

(c) Show image in ImageView

Figure 1

Motivating example for code recommendations related to “select image from gallery”. Figure 1a shows the selected code snippet based on the initial search query and Figures 1b and 1c show code snippets corresponding to two related features, as recommended by FACER.

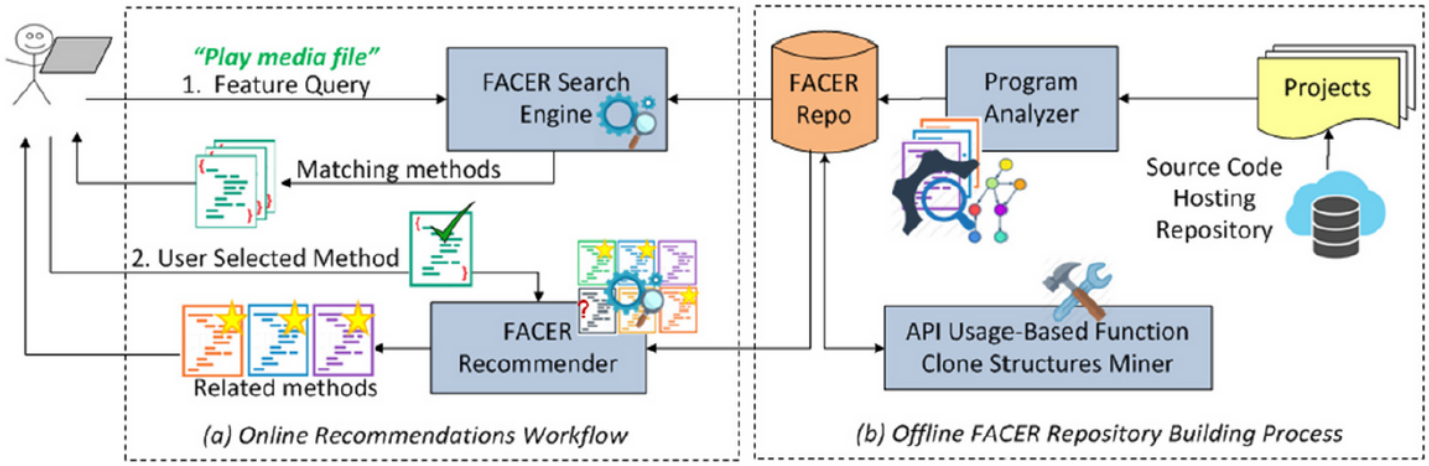


Figure 2

FACER System Components and Workflow

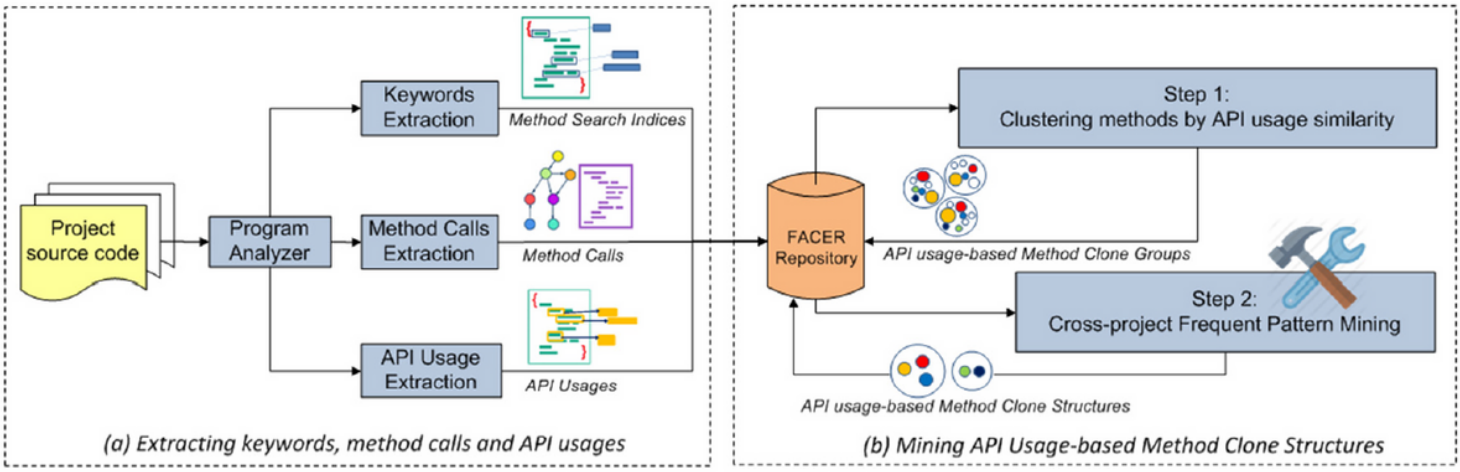
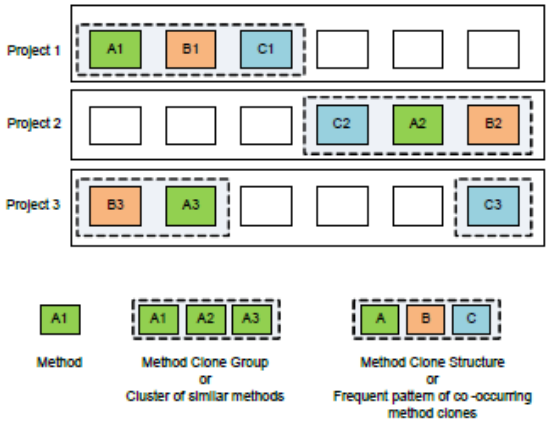


Figure 3

Offline FACER Repository Building Components



(a) Abstract Method Clone Structure Across Projects

```
public void startMeasureResult() {
   btAdapter = BluetoothAdapter.getDefaultAdapter();
   BluetoothDevice device =
      btAdapter.getRemoteDevice(address);
   try {
      btSocket = createBluetoothSocket(device);
   } catch (IOException e) {
      errorExit("Fatal Error",
        "In onResume() and socket create failed: " +
        e.getMessage() +
        ".");}
   btAdapter.cancelDiscovery();
   Log.d("bluetooth", "...Connecting...");
   try {
      btSocket.connect();
      Log.d("bluetooth", "....Connection ok...");
   } catch (IOException e) {
      try {
         btSocket.close();
      } catch (IOException e2) {
         errorExit("Fatal Error",
           "In onResume() and unable to close " +
           "socket ..." +
           e2.getMessage() +
           ".");}}
   Log.d("bluetooth", "...Create Socket...");
   mConnectedThread = new ConnectedThread(btSocket);
   mConnectedThread.start();
}
```

(b) Method A1 from Project 1

```
public void run() {
   Log.i(TAG, "BEGIN mConnectThread SocketType:"
      + mSocketType);
   setName("ConnectThread" + mSocketType);
   // Always cancel discovery because
   //it will slow down a connection
   mAdapter.cancelDiscovery();
   // Make a connection to the BluetoothSocket
   try {
      // This is a blocking call and will
      // only return on a successful
      // connection or an exception
      mmSocket.connect();
   } catch (IOException e) {
      // Close the socket
      try {
         mmSocket.close();
      } catch (IOException e2) {
         Log.e(TAG, "unable to close() " +
            mSocketType +
           " socket during connection failure", e2);
      }
   connectionFailed();
   return;
   }
   // Reset the ConnectThread because we're done
   synchronized (BluetoothChatService.this) {
      mConnectThread = null;
   }
   connected(mmSocket, mmDevice, mSocketType);
}
```

(c) Method A2 from Project 2

```
public ConnectedThread(BluetoothSocket socket) {
   InputStream tmpIn = null;
   OutputStream tmpOut = null;
   // Get the BluetoothSocket input and output streams
   try {
      tmpIn = socket.getInputStream();
      tmpOut = socket.getOutputStream();
   } catch (IOException e) { }
   mmInStream = tmpIn;
   mmOutStream = tmpOut;
}
```

(d) Method B1 from Project 1

```
public ConnectedThread(BluetoothSocket socket,
      String socketType) {
   Log.d(TAG, "create ConnectedThread: " +
      socketType);
   mmSocket = socket;
   InputStream tmpIn = null;
   OutputStream tmpOut = null;
   // Get the BluetoothSocket input and
   //output streams
   try {
      tmpIn = socket.getInputStream();
      tmpOut = socket.getOutputStream();
   } catch (IOException e) {
      Log.e(TAG, "temp sockets not created", e);
   }
   mmInStream = tmpIn;
   mmOutStream = tmpOut;
   mState = STATE_CONNECTED;
}
```

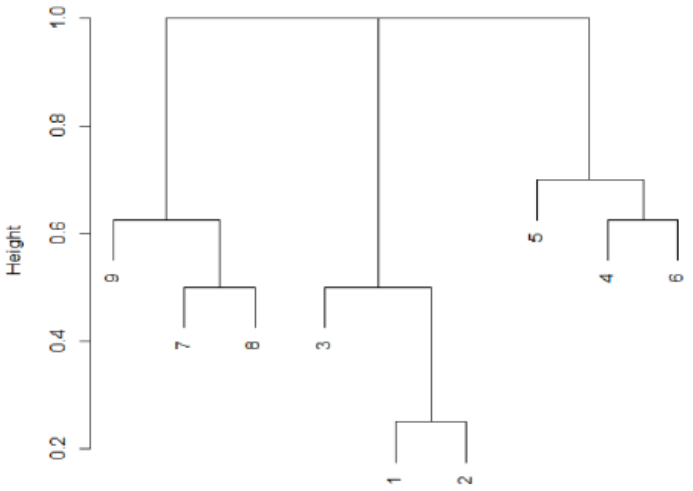
(e) Method B2 from Project 2

Figure 4

A real example of a API Usage-based Method Clone Structure taken from Bluetooth chat projects. Highlighting shows common API usages

Method ID	API Call IDs
1	1 2 3 4
2	1 2 3
3	7 8 1 2 3
4	11 12 13 24 25
5	26 27 11 28 12 29 13
6	31 11 32 12 13 33
7	8 35 9 10
8	8 9 10 15 16
9	41 42 8 43 9 10

(a) Example method & API Call IDs



(b) Dendrogram obtained by clustering methods 1 - 9

Method ID	Clone Group ID
1	1
2	1
3	1
4	2
5	2
6	2
7	3
8	3
9	3

(c) Resulting clone group for each method

Figure 5

Step 1: Cluster methods by API usage similarity. After this step, each method in our repository has a clone group ID.

Project ID	Clone Group IDs
1	1 2 3 11 19
2	1 9 2 4 3
3	5 6 15 18 19
4	21 5 22 6
5	26 1 2 3

(a) Example clone group IDs recorded for each project

Clone Structure ID	Clone Group IDs	Support
C1	5 6	2
C2	1 2 3	3

(b) Resulting method Clone Structures across Projects

Figure 6

Step2: Mining frequent patterns of method clones across projects

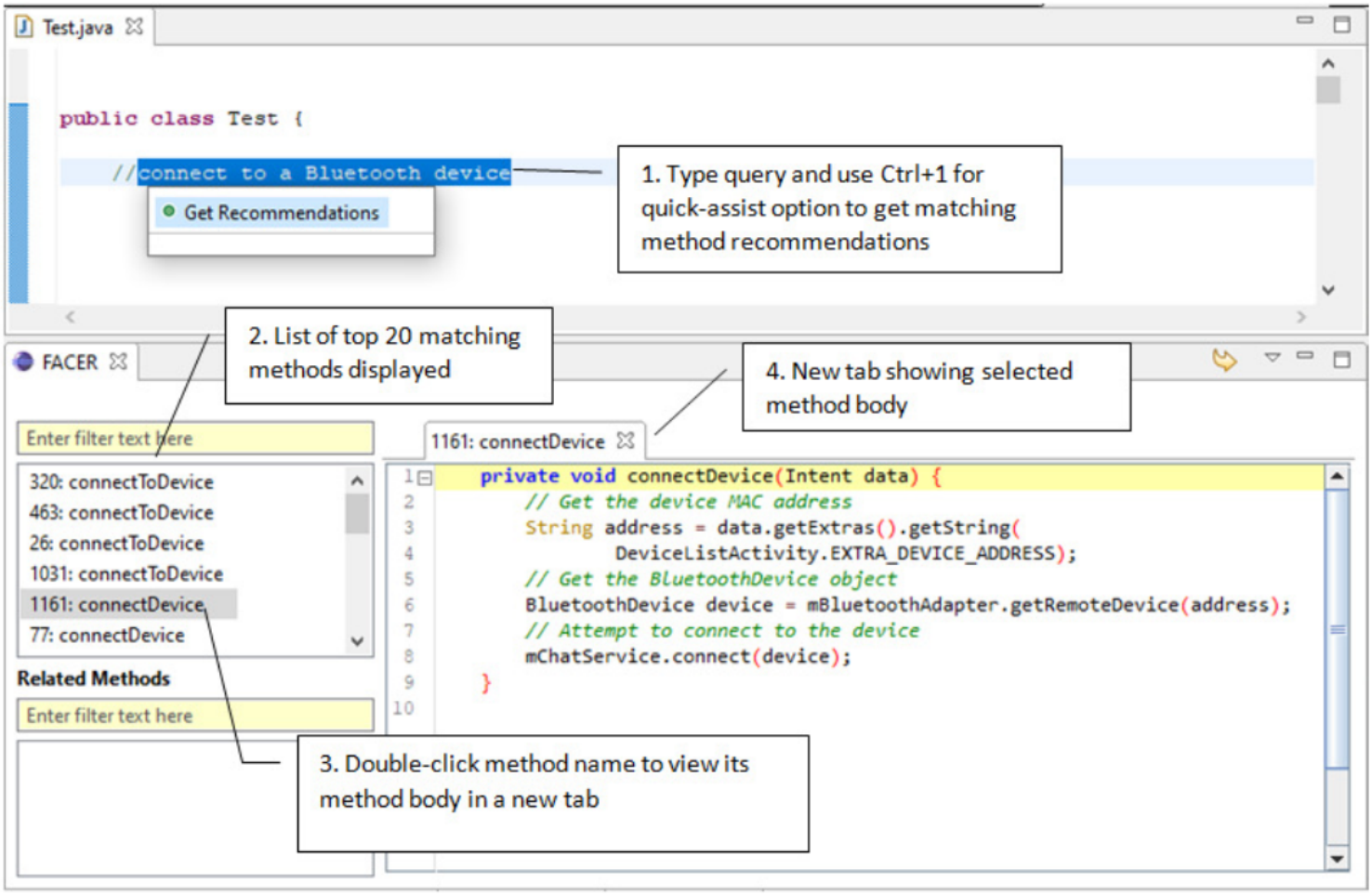


Figure 7

Stage 1: Method Search

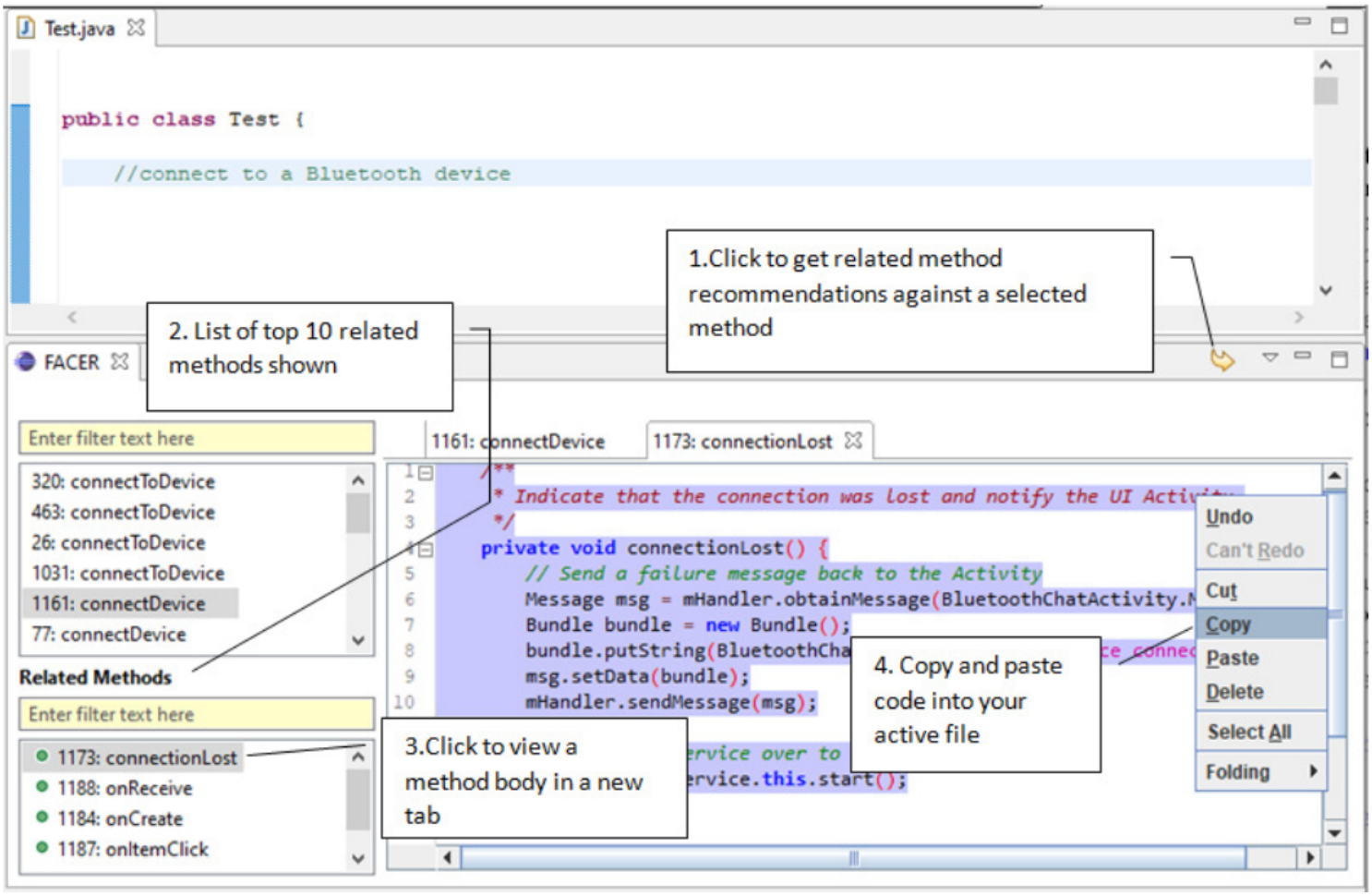


Figure 8

Stage 2: Related Method Recommendations

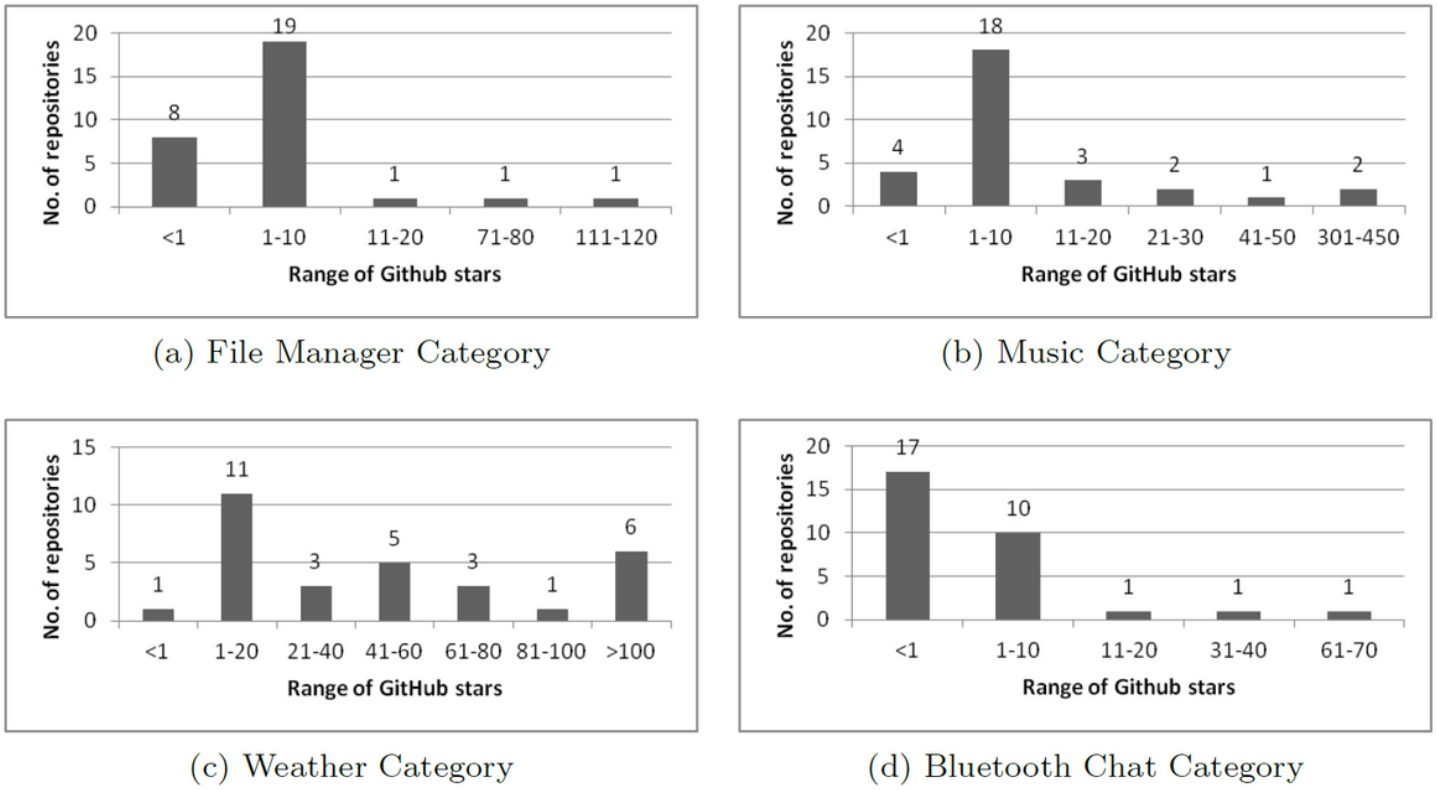


Figure 9

The number of GitHub repositories from the four categories across different ranges of the number of stars

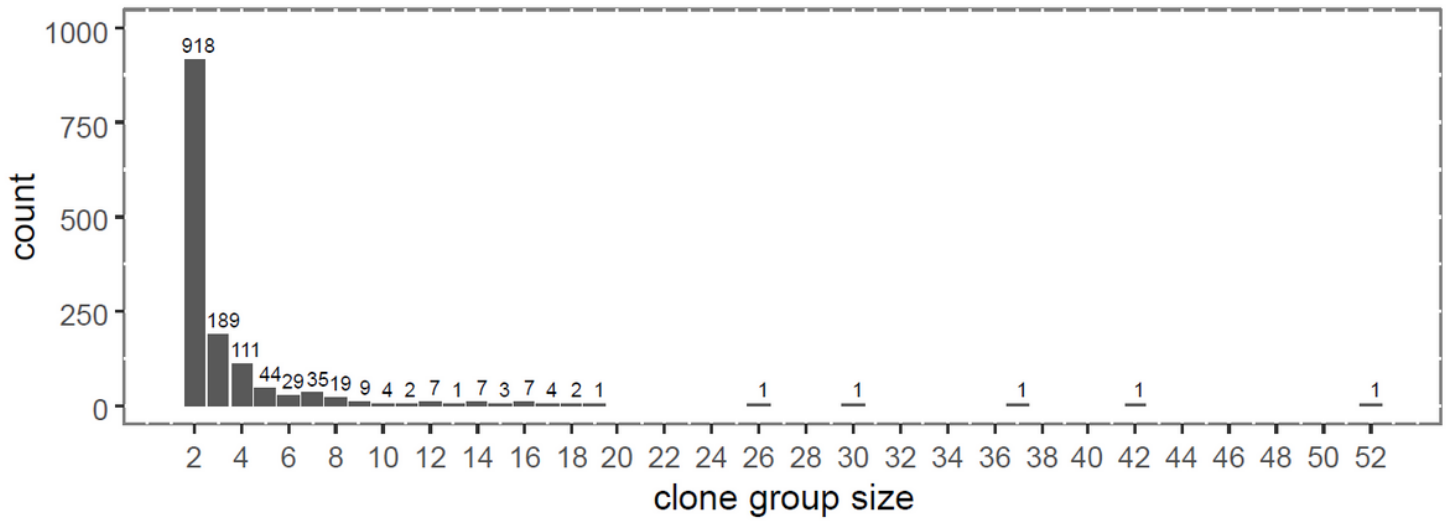
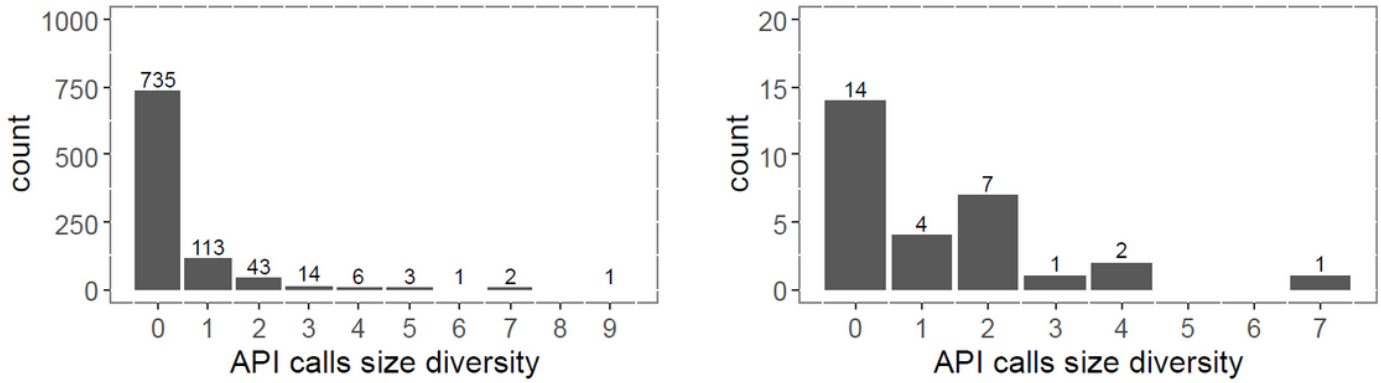


Figure 10

Frequencies of clone groups of varying sizes with similarity threshold α = 0.5



(a) Frequency of API call size diversity for clone groups of size 2

(b) Frequency of API call size diversity for clone groups of size 6

Figure 11

Example API call size diversity for clone groups of size 2 and 6

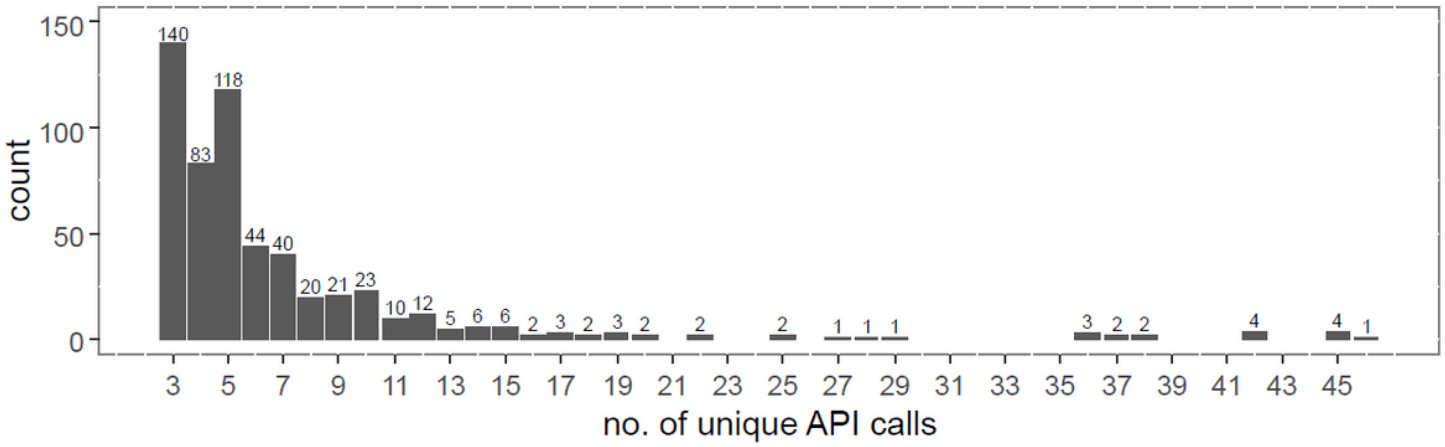


Figure 12

Distribution of API call size for all the methods from our sampled clone groups in Table 4

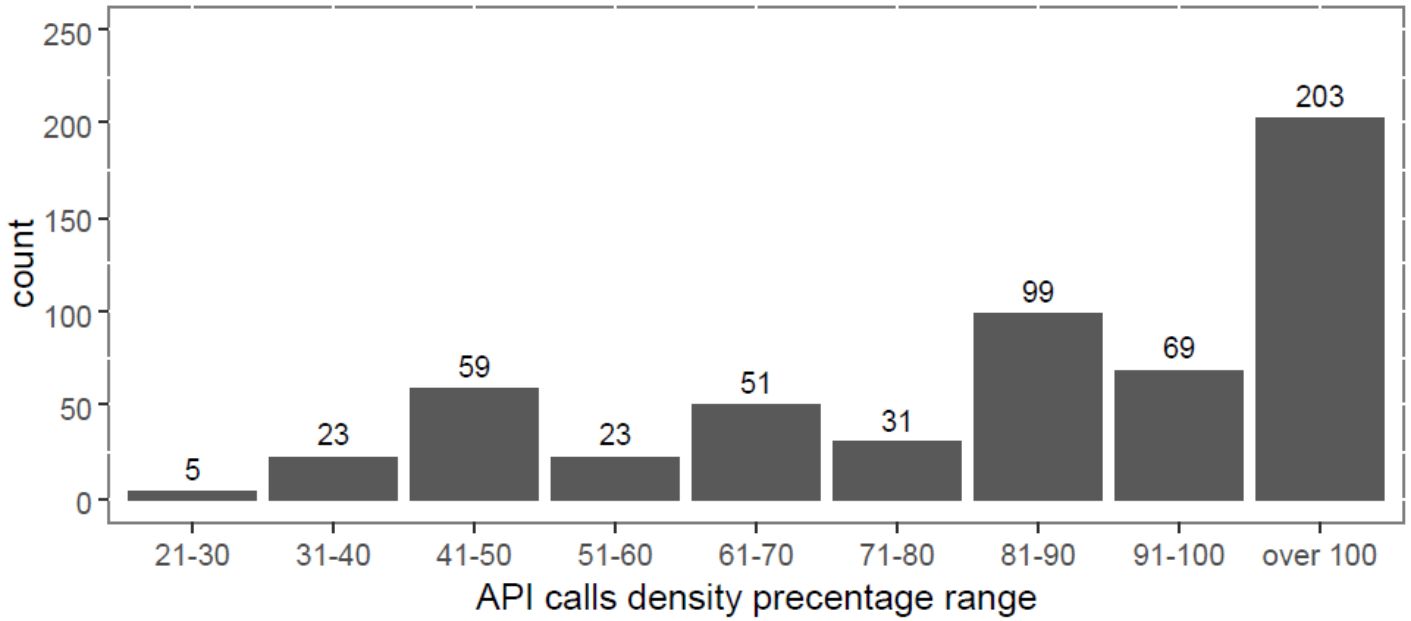


Figure 13

Method distribution from sampled clone groups based on API call density

```java
public boolean isNetworkAvailableAndConnected() {
        ConnectivityManager connectivityManager
                = (ConnectivityManager)
                    mContext.getSystemService(Context.CONNECTIVITY_SERVICE);

        NetworkInfo networkInfo = connectivityManager.getActiveNetworkInfo();
        return networkInfo != null && networkInfo.isConnected();
    }
```

(a) Clone Group 1 Method 1

```java
public boolean isNetworkAvailable() {
        ConnectivityManager connectivityManager
                = (ConnectivityManager)
                    mContext.getSystemService(Context.CONNECTIVITY_SERVICE);
        NetworkInfo activeNetworkInfo = connectivityManager.getActiveNetworkInfo();
        return activeNetworkInfo != null && activeNetworkInfo.isConnected();
    }
```

(b) Clone Group 1 Method 2

```java
/**
     * Indicate that the connection was lost and notify the UI Activity.
     */
    private void connectionLost() {
        // Send a failure message back to the Activity
        Message msg = mHandler.obtainMessage(LanylActivity.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString(LanylActivity.TOAST, "?????????");
        msg.setData(bundle);
        mHandler.sendMessage(msg);

        // Start the service over to restart listening mode
        LanylService.this.start();
    }
```

(c) Clone Group 2 Method 1

```java
private void connectionLost() {
        Message msg = handler.obtainMessage(MainActivity.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString("toast", "Conexion perdida con el dispositivo");
        msg.setData(bundle);
        handler.sendMessage(msg);

        // Start the service over to restart listening mode
        ChatController.this.start();
    }
```

(d) Clone Group 2 Method 2

```java
private void connectionFailed() {
        Message msg = handler.obtainMessage(MainActivity.MESSAGE_TOAST);
        Bundle bundle = new Bundle();
        bundle.putString("toast", "Unable to connect device");
        msg.setData(bundle);
        handler.sendMessage(msg);

        // Start the service over to restart listening mode
        ChatController.this.start();
    }
```

(e) Clone Group 2 Method 3

Figure 14

Examples of evaluated clone groups. Figures 14a-14b show two methods from a clone group of size = 10. Figures 14c-14e show three methods from a clone group of size = 37

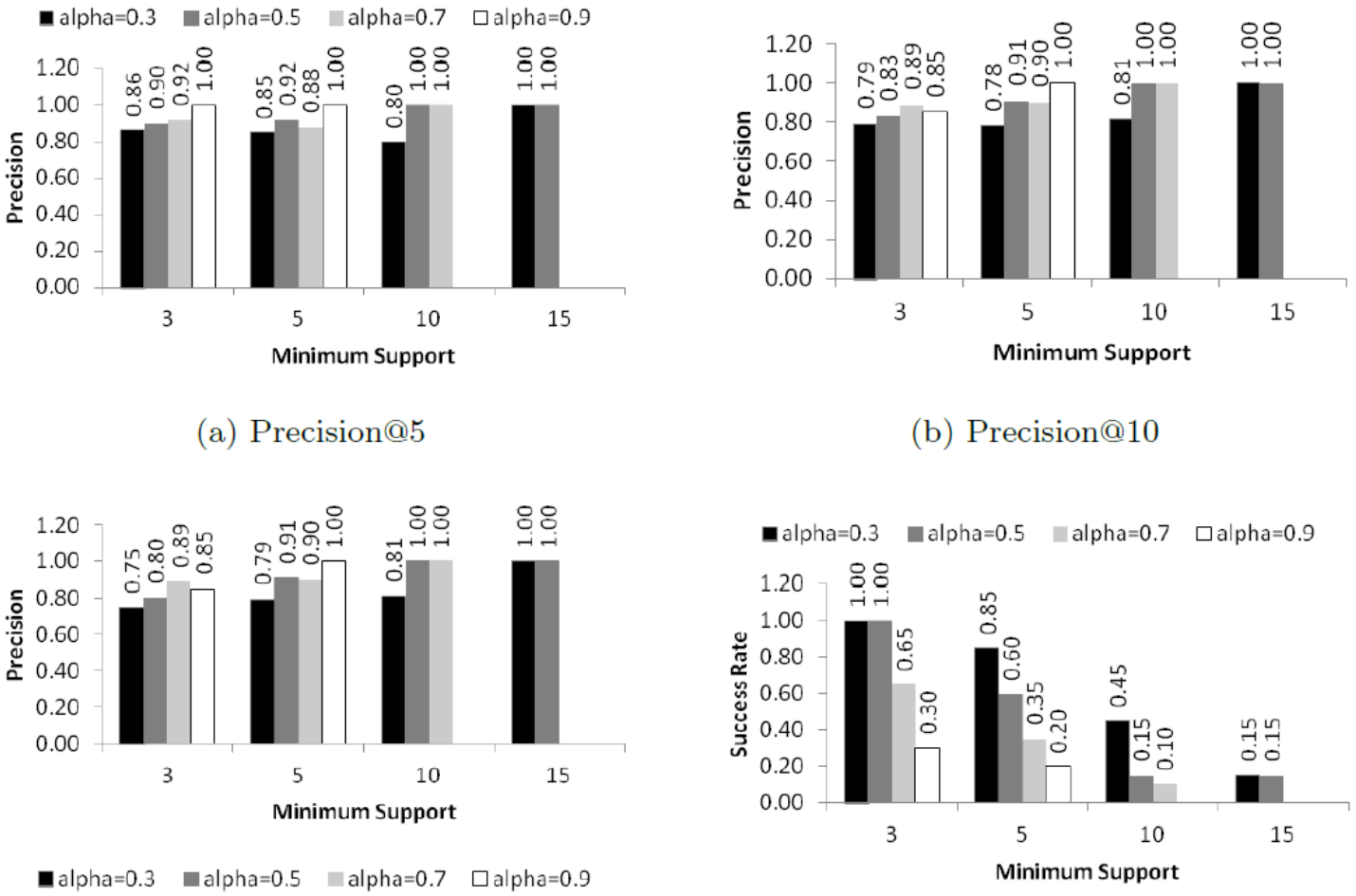


Figure 15

Precision and success rate of recommendations across varying similarity threshold (alpha) and minimum support(beta)

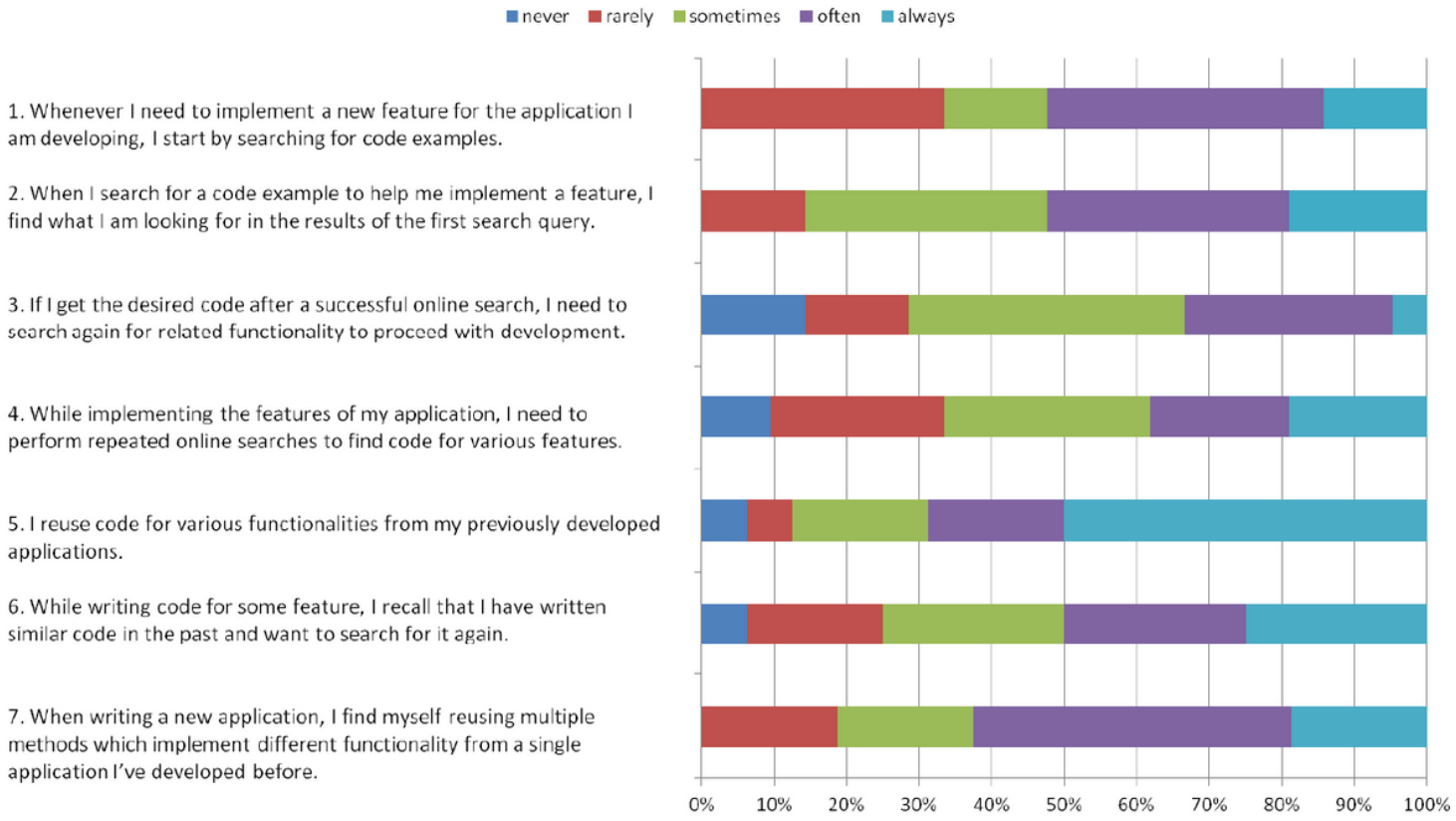


Figure 16

Analysing developer's code search and reuse practices

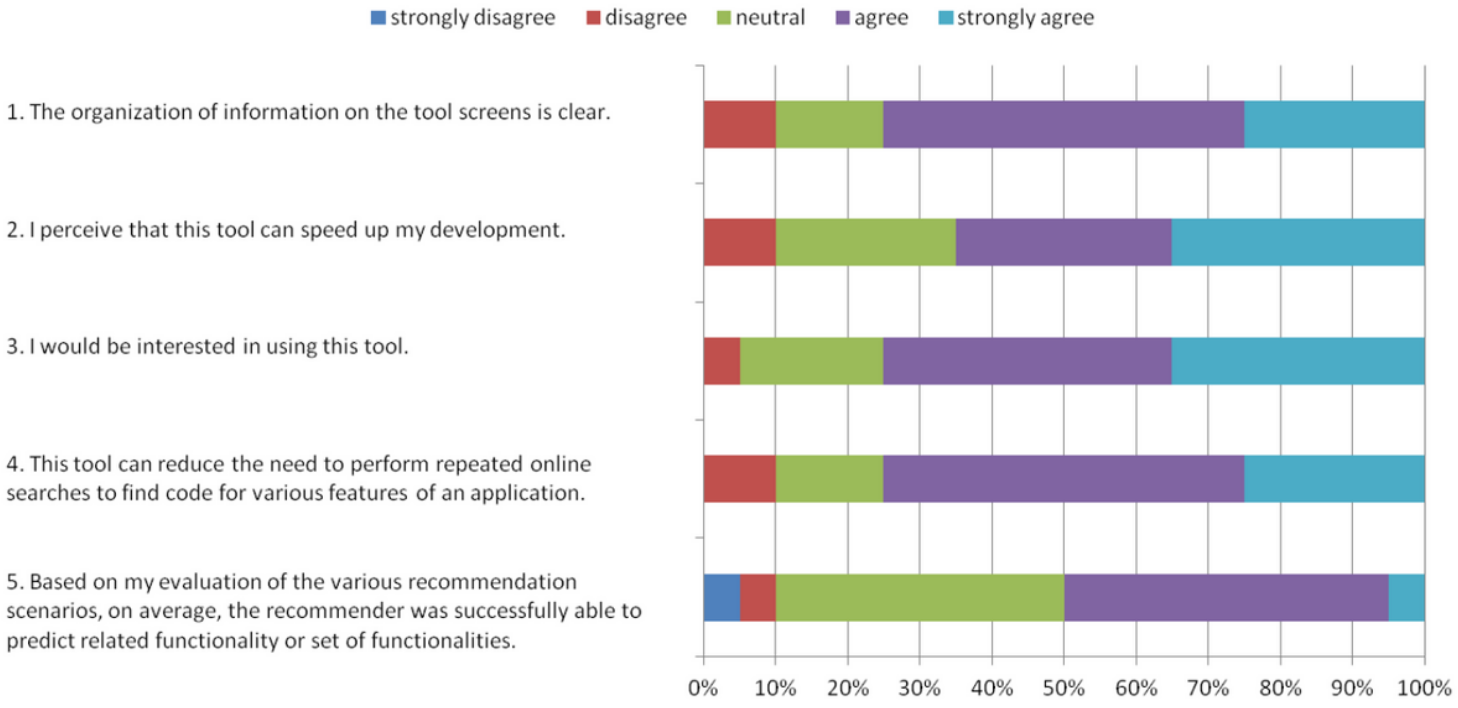


Figure 17

Analysing developer's feedback on FACER

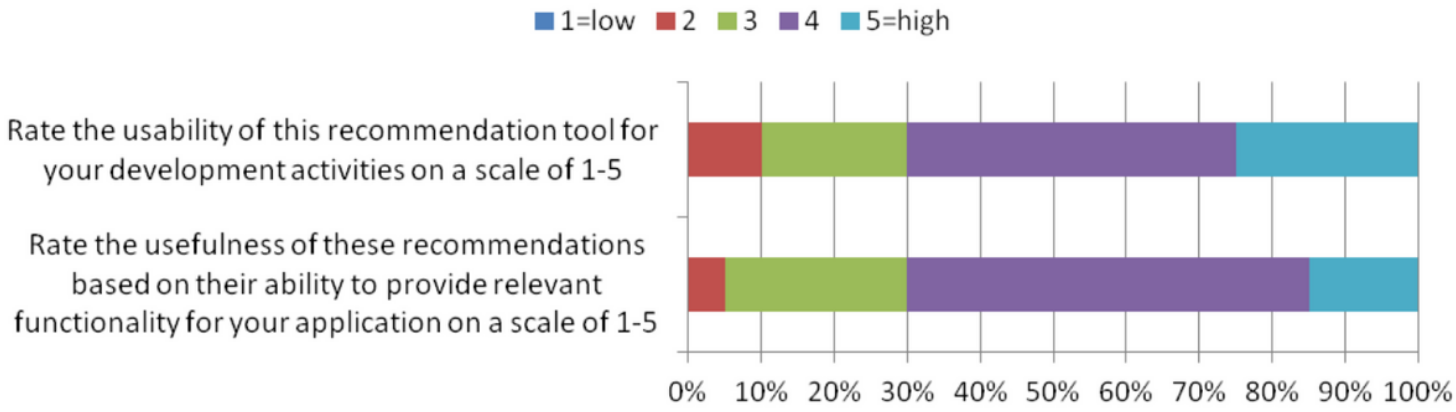


Figure 18

Professional developer's ratings on the usefulness and usability of FACER

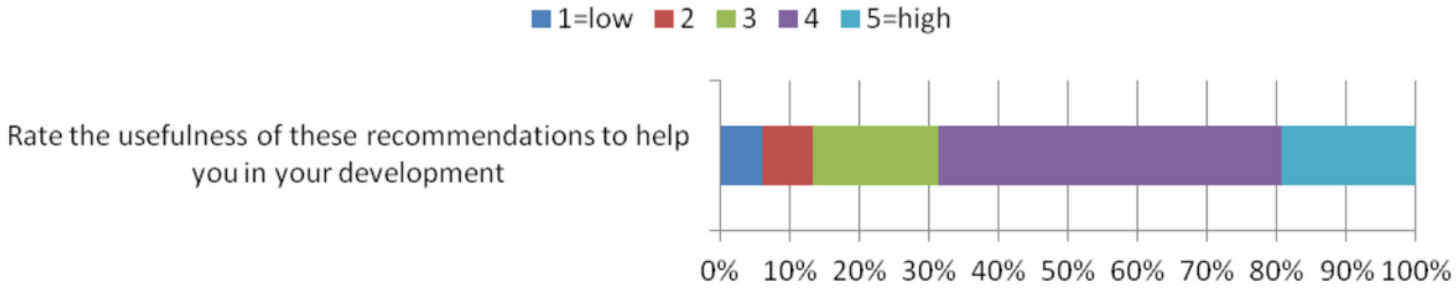


Figure 19

Student developer's ratings on the usefulness of FACER