The Journal of Systems & Software 178 (2021) 110971

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Characterizing top ranked code examples in Google[☆]

Andre Hora

Department of Computer Science, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brazil

ARTICLE INFO

Article history:
Received 11 June 2020
Received in revised form 25 March 2021
Accepted 30 March 2021
Available online 9 April 2021

Keywords:
Code examples
API usage
Software quality
Code search
Google search engine

ABSTRACT

Developers often look for code examples on the web to improve learning and accelerate development. Google indexes millions of pages with code examples: pages with better content are likely to be top ranked. In practice, many factors may influence the rank: page reputation, content quality, etc. Consequently, the most relevant information on the page, *i.e.*, the code example, may be overshadowed by the search engine. Thus, a better understanding of how Google would rank code examples in isolation may provide the basis to detect its strengths and limitations on dealing with such content. In this paper, we assess how the Google search engine ranks code examples. We build a website with 1,000 examples and submit it to Google. After being fully indexed, we query and analyze the returned examples. We find that pages with multiple code examples are more likely to top ranked by Google. Overall, single code examples that are higher ranked are larger, however, they are not necessarily more readable and reusable. We predict top ranked examples with a good level of confidence, but generic factors have more importance than code quality ones. Based on our results, we provide insights for researchers and practitioners.

1. Introduction

A code example is a snippet of reusable source code that illustrates how a programming problem can be solved (Keivanloo et al., 2014; Menezes et al., 2019). Code examples improve learning (Holmes et al., 2009; Robillard and Deline, 2011), support reuse (Holmes and Walker, 2012; Philip et al., 2012; Yang et al., 2017), and accelerate development (Spring Code Guides, 2021; Vincent, 2018). In practice, developers often rely on web search engines, such as Google, to find code examples (Gu et al., 2016; Hora, 2021b; Kim et al., 2010; Niu et al., 2017; Raghothaman et al., 2016; Sim et al., 2011, 2013; Stolee et al., 2014). Previous studies report that developers may spend up to 20% of their time looking for code examples on the web (Brandt et al., 2009; Niu et al., 2017; Philip et al., 2012). For instance, a popular programming website, W3Schools (W3Schools, 2021), has over 3 billions pageviews per year.[1] Therefore, accessing good code examples is essential to software development in the current days (Nasehi et al., 2012).

The Google search engine indexes millions of webpages that include code examples (Treude and Aniche, 2018). Naturally, pages with better content are likely to be top ranked, grabbing more attention and click from the users (Google General Guidelines, 2021; How Search Algorithms Work, 2021; Treude and Aniche, 2018). In practice, many factors may influence the rank: page reputation, page domain, content quality, to name a few (Furnell and Evans, 2007; Google General Guidelines, 2021; Hannak et al., 2013; Kliman-Silver et al., 2015). However, these factors are inherently hard to enumerate and assess as the search engines do not reveal which particular ones they rely on when determining a website ranking (Furnell and Evans, 2007). For instance, prior work reports that there are over 200 different factors used by Google to calculate a page's rank (Furnell and Evans, 2007). This way, the literature has examined several facets of web search engines to better understand how they work, to improve content discovery, or even to assess their fairness. For instance, techniques are proposed to audit black-box algorithms (Diakopoulos, 2014; Sandvig et al., 2014), empirical studies are performed to assess how personalization of web search may affect the results (Hannak et al., 2013; Kliman-Silver et al., 2015), to identify factors related to highly ranked webpages (Furnell and Evans, 2007), to assess partisanship of search results (Diakopoulos et al., 2018; Hu et al., 2019), and to analyze search snippets (Cutrell and Guan, 2007; Kaisser et al., 2008).

In practice, code example webpages are composed not only by code, but they are mixed with other elements, such as code explanations (see Fig. 1). Indeed, code examples are often enriched with textual description: as Google is a general web search engine, natural language is a solution to bypass the lack of expression inherent of programming language (Chen and Zhou, 2018; Gu et al., 2018; Hu et al., 2018; Nasehi et al., 2012; Yao et al.,

☆ Editor: Gabriele Bavota.
E-mail address: andrehora@dcc.ufmg.br.
URL: http://dcc.ufmg.br/~andrehora.

1 https://www.w3schools.com/about/default.asp.

https://doi.org/10.1016/j.jss.2021.110971
0164-1212/© 2021 Elsevier Inc. All rights reserved.

2019). However, these factors may introduce a side effect: we are left unsure about the quality of the code examples themselves. For instance, a webpage with a poor code example could be top ranked due to its good textual description (we present concrete examples in Section 2). This way, it is important to understand how Google would rank code examples in isolation, *i.e.,* without any other page elements. In this case, we could query and assess the characteristics of top/bottom ranked code examples and verify their quality aspects, for instance, whether good coding practices (Buse and Weimer, 2009; Martin, 2009; Moreno et al., 2015; Nasehi et al., 2012; Scalabrino et al., 2016, 2018) are found in higher ranked ones. This may provide the basis to detect the possible strengths and limitations of the search engine in dealing with code. While previous studies propose dedicated code search engines (Bajracharya et al., 2006; Codota, 2021; Kim et al., 2010; krugle, 2021; McMillan et al., 2012; SearchCode, 2021) and techniques to rank code examples (Buse and Weimer, 2012; Gu et al., 2018; Hora, 2021a; Keivanloo et al., 2014; Moreno et al., 2015), to the best of our knowledge, no study assesses how Google – the *de facto* web search engine (Search Engine Market Share Worldwide, 2021) – deals with such content.

In this paper, we perform an empirical study to assess how the Google search engine ranks code examples. We analyze the characteristics of the top and bottom ranked code examples in return to code search queries. We focus on code examples that describe the usage of APIs, which are often the target of code search (Buse and Weimer, 2012; Parnin et al., 2012; Sadowski et al., 2015). For this purpose, we perform the following steps. *First*, we select 100 API methods from popular libraries and frameworks. *Second*, we collect from programming websites 1000 code examples about the selected APIs, including didactic and real software examples. *Third*, we build a website, host the code examples on webpages, and submit this website to the Google search engine. *Lastly*, after being fully indexed by Google, we query for APIs and assess the returned code examples in this controlled environment. Specifically, we investigate: (i) the rank of webpages with single and multiple code examples; (ii) the rank of webpages with didactic and real software code examples; (iii) the characteristics of top/bottom ranked code examples, such as their size, readability, reusability, and query similarity; and (iv) whether top ranked code examples can be predicted. We then propose the following research questions:

- *RQ1 (single vs. multiple): How are single and multiple code examples ranked?* We find that webpages with multiple code examples are more likely to be top ranked by Google than webpages with single examples. 82% of the webpages with multiple code examples are top ranked.
- *RQ2 (didactic vs. real software): How are didactic and real software code examples ranked?* Code examples created for didactic purposes are more likely to be higher ranked than code examples originated from real software systems. However, this is likely to happen because they have more API references and tokens density, not because they have better quality.
- *RQ3 (top vs. bottom): What are the characteristics of top ranked code examples?* Overall, top ranked code examples are larger and have more API references. We find that readable and reusable code examples are not necessarily top ranked.
- *RQ4 (prediction and importance): To what extent can we predict that a code example will be top ranked? What are the most important characteristics?* We can predict top ranked code examples with a good level of confidence (in the best case, precision: 79%, recall: 70%, and AUC: 89%). Generic factors (*e.g.,* term frequency and size) are more important than code quality factors (*e.g.,* reusability).

Based on our results, we provide insights to drive future research on code search. Moreover, we provide insights to improve the user experience of code example webpages, which is a practice encouraged by Google to benefit users and facilitate content discovery (Search Engine Optimization (SEO) Starter Guide, 2021).

Contributions. This study has three major contributions: (i) we provide the first empirical study to assess how the Google search engine ranks single/multiple and didactic/real software code examples (Sections 4.1 and 4.2); (ii) we study factors associated to top/bottom ranked code examples and investigate whether these factors can predict rank positions (Sections 4.3 and 4.4); and (iii) we provide guidelines to improve code example webpages and present insights to code search researchers (Section 5).

Structure of the paper. Section 2 presents a motivating example. Section 3 describes our study design. Section 4 presents our results and Section 5 discusses them. Section 6 states the threats to validity. Finally, Section 7 presents the related work and Section 8 concludes the paper.

2. Motivating examples

Developers often look for code examples on the web (Gu et al., 2016; Sim et al., 2011, 2013; Stolee et al., 2014). They are commonly interested in how to use APIs provided by libraries and frameworks (Buse and Weimer, 2012; Parnin et al., 2012; Sadowski et al., 2015). Typically, a code search query consists of API tokens, *i.e.,* class and method names (Niu et al., 2017). For example, if a developer desires to retrieve code examples about an API, for instance, `File.mkdirs`,[2] he might simply query for "*File mkdirs*". Fig. 1 presents the *first* result for this query on Google, a code example webpage provided by the website tutorialspoint.[3]

In addition to the code example, there are other elements in the webpage, such as API description and related APIs. According to good coding practices (Martin, 2009; Nasehi et al., 2012), the code example itself could be slightly better: variable names could be more descriptive (*e.g.,* `file` instead of `f` and `isDirectoryCreated` instead of `bool`), code comments could be more detailed (*e.g.,* "print if the directory is created" instead of "print"), and code could be more self explained (*e.g.,* by handling the case the file already exists).

Another programming website, JavaTutorialHQ,[4] presents a slightly richer code example of the same API, as shown in Fig. 2: it has intention revealing variable names, descriptive comments, and clearer code.[5] This webpage, however, is ranked as *third* in our search. Indeed, Google search results depend on several factors, such as page reputation, content, domain, location, query, etc. (Furnell and Evans, 2007; Google General Guidelines, 2021; Hannak et al., 2013; Kliman-Silver et al., 2015), thus, to better understand how the code examples themselves are ranked, they should be hosted and queried under the same conditions.

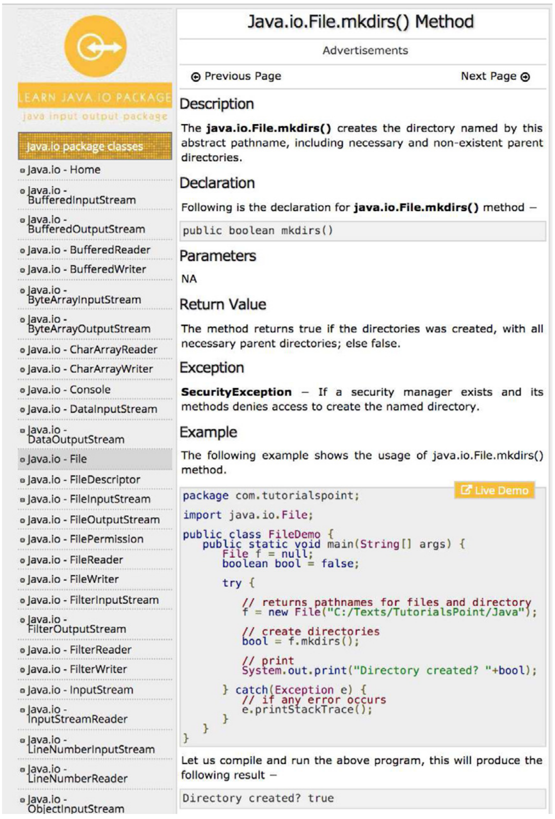
Example 1. As a proof of concept, we performed a simple experiment: we analyzed how Google would rank code examples when they are under equivalent conditions (*i.e.,* without code explanations and hosted on the same domain). We collected 10 code examples of `File.mkdirs` from popular programming websites (including the two examples aforementioned) and hosted them

2 https://docs.oracle.com/javase/8/docs/api/java/io/File.html.

3 https://www.tutorialspoint.com/java/io/file_mkdirs.htm.

4 http://javatutorialhq.com/java/io/file-class-tutorial/mkdirs-method-example.

5 For simplicity, we only show code example itself in Fig. 2, and not the full webpage.


LEARN JAVA.IO PACKAGE
java input output package
Java.io package classes
Java.io - Home
Java.io - BufferedInputStream
Java.io - BufferedOutputStream
Java.io - BufferedReader
Java.io - BufferedWriter
Java.io - ByteArrayInputStream
Java.io - ByteArrayOutputStream
Java.io - CharArrayReader
Java.io - CharArrayWriter
Java.io - Console
Java.io - DataInputStream
Java.io - DataOutputStream
Java.io - File
Java.io - FileDescriptor
Java.io - FileInputStream
Java.io - FileOutputStream
Java.io - FilePermission
Java.io - FileReader
Java.io - FileWriter
Java.io - FilterInputStream
Java.io - FilterOutputStream
Java.io - FilterReader
Java.io - FilterWriter
Java.io - InputStream
Java.io - InputStreamReader
Java.io - LineNumberInputStream
Java.io - LineNumberReader
Java.io - ObjectInputStream
Advertisements
Previous Page
Next Page
Live Demo

Java.io.File.mkdirs() Method

Description

The **java.io.File.mkdirs()** creates the directory named by this abstract pathname, including necessary and non-existent parent directories.

Declaration

Following is the declaration for **java.io.File.mkdirs()** method –

```
public boolean mkdirs()
```

Parameters

NA

Return Value

The method returns true if the directories was created, with all necessary parent directories; else false.

Exception

SecurityException – If a security manager exists and its methods denies access to create the named directory.

Example

The following example shows the usage of java.io.File.mkdirs() method.

```
package com.tutorialspoint;

import java.io.File;

public class FileDemo {
   public static void main(String[] args) {
      File f = null;
      boolean bool = false;

      try {

         // returns pathnames for files and directory
         f = new File("C:/Texts/TutorialsPoint/Java");

         // create directories
         bool = f.mkdirs();

         // print
         System.out.print("Directory created? "+bool);

      } catch(Exception e) {
         // if any error occurs
         e.printStackTrace();
      }
   }
}
```

Let us compile and run the above program, this will produce the following result –

```
Directory created? true
```

Fig. 1. Code example webpage (Tutorialspoint).

```
public class FileMkdirsExample {

   public static void main(String[] args) {

      // initialize File object
      File file = new File("C:\\javatutorialhq\\test_folder\\input\\test1");
      // check if the pathname already exists
      // if not create it
      if (!file.exists()) {
         // create the full path name
         boolean result = file.mkdirs();
         if (result) {
            System.out.println("Successfully created " + file.getAbsolutePath());
         } else {
            System.out.println("Failed creating " + file.getAbsolutePath());
         }
      } else {
         System.out.println("Pathname already exists");
      }
   }
}
```

Fig. 2. Code example (JavaTutorialHQ).

on 10 distinct webpages. After being indexed by Google, we performed the same previous query "*File mkdirs*", but now restricted to this website.[6] The result was the opposite of the first query: the richer code example (*i.e.*, Fig. 2) was now top ranked and the poorer code example (*i.e.*, Fig. 1) was bottom ranked. While other factors may have contributed to better rank the poorer code example in the first query, the richer code example was favored in the second query, when the conditions were equivalent.

Example 2. To further assess the previous result, we performed a larger analysis, with 1000 code examples and 100 API methods. That is, for each API method, we collected 10 code examples, kept track of the code example that was originally top ranked by Google, hosted each on a single webpage on our website, and made them indexed by Google. Then, for each API, we performed one query restricted to our website (100 queries in total) and verified the rank position of the code examples. The result was also different: only about 20% of the code examples originally top ranked by Google were ranked as first in our controlled environment when *only* the code examples were available.

Problem and proposed solution. This initial analysis provides evidence that code examples may be obfuscated by other factors. That is, *under equivalent conditions*, the ordering of the returned code examples may be completely distinct from when they are hosted on their native websites. In this study, we propose to host code examples on a website and make them indexed by Google, so we can focus only on the code examples, without any external (*e.g.*, page reputation, domain, users' location) nor internal (*e.g.*, code explanations) interference. Particularly, these code examples should have distinct characteristics (*e.g.*, size, readability, origin, etc.), so we can better profile them. Then, we can rely on the Google search engine, query, and assess the returned code examples. This may provide the basis to better understand how Google "sees" the major content of programming websites (*i.e.*, the code examples themselves), which is the sole factor that varies among our code example webpages and that the website owner can directly control, change, and improve.

3. Study design

Fig. 3 presents an overview of the proposed approach to assess how the Google search engine ranks code examples. It includes five major steps: (1) selecting APIs, (2) collecting code examples, (3) indexing code examples, (4) querying code examples, and (5) assessing query results. We detail each step in the following subsections. Our results are publicly available.[7]

3.1. Selecting APIs

In this study, we focus on code examples that describe the usage of APIs, which are often the target of code search on the web (Buse and Weimer, 2012; Parnin et al., 2012; Sadowski et al., 2015). Therefore, we start by selecting important libraries and frameworks, from where the code examples will be extracted. We select Google Guava, Java SE, Apache Commons, and Spring Framework (Table 1) due to two reasons. First, these systems are relevant libraries and frameworks that have a large community of users. Second, due to their popularity, many programming websites provide code examples about their APIs. Having programming websites with code examples about these libraries and frameworks is important because we need to manually collect code examples about their APIs (as explained in the following subsections). Thus, if we had selected less popular systems, the manual task of finding code examples would not be feasible.

The selected systems can be described as follows. Guava is a library that includes core features to handle collections and utilities for concurrency, IO, hashing, among others.[8] Java SE provides basic functionalities that are used by almost all Java applications, such as collections, date and time facilities, among others.[9] Apache Commons provides a large set of reusable components.[10] Lastly, the Spring Framework supports the creation of

[6] Restricted queries can be done with the "`site:`" command provided by Google.

[7] https://github.com/andrehora/jss-code-examples.

[8] https://github.com/google/guava.

[9] https://docs.oracle.com/javase/8/docs.

[10] https://commons.apache.org.

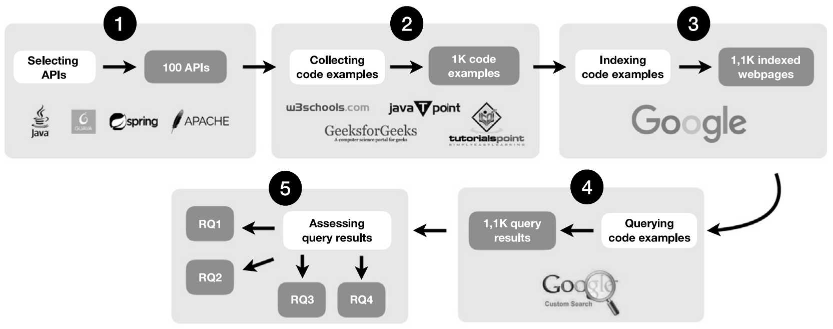


Fig. 3. Overview of the proposed approach.

Table 1
Selected APIs.

System	API	
	Class name	Method name
Google Guava	Preconditions	checkArgument, checkElementIndex, checkNotNull, checkPositionIndex, checkState
	Ints	asList, concat, saturatedCast, toArray, tryParse,
	Lists	asList, charactersOf, newArrayList, partition, reverse
	Sets	cartesianProduct, difference, intersection, powerSet, union
	Files	append, createTempDir, readLines, toByteArray, write
Java SE	BufferedReader	close, read, readLine, reset, skip
	File	exists, getAbsolutePath, isDirectory, listFiles, mkdirs
	ArrayList	add, contains, get, isEmpty, size
	HashMap	containsKey, containsValue, isEmpty, put, putAll
	Scanner	hasNextLine, match, nextLine, useDelimiter, useLocale
Apache commons	Base64	decode, decodeBase64, encodeBase64, encodeBase64URLSafeString, isArrayByteBase64
	CSVFormat	parse, withCommentMarker, withDelimiter, withHeader, withQuote
	FilenameUtils	concat, getExtension, isExtension, normalize, separatorsToUnix
	IOUtils	closeQuietly, copy, readLines, toByteArray, toInputStream
	StringUtils	isBlank, join, split, substringBefore, trimToNull
Spring framework	SpringApplication	addListeners, exit, run, setAdditionalProfiles, setBannerModemidrule
	ApplicationContext	containsBean, getBean, publishEventmidrule, getAutowireCapableBeanFactory, getBeanDefinitionNames
	BindingResult	getFieldErrors, getGlobalErrors, hasErrors, hasFieldErrors, rejectmidrule
	RestTemplate	exchange, getForEntity, getForObject, postForObject, setMessageConvertersmidrule
	ModelAndView	addAllObjects, addObject, getModel, hasView, setViewName

enterprise applications in many scenarios and architectures, such as web apps, microservices, and cloud.[11]

For each system, we select five classes, and, for each class, we select five methods, that is, 25 methods per system and 100 methods in total, as presented in Table 1. We rely on API popularity (Lima and Hora, 2020) to select these APIs in order to maximize the chance we find code examples on the web. That is, the more popular an API is, the more examples we are likely to find on the web. Specifically, we adopt the rankings at class and method levels[12] proposed by the programming website ProgramCreek (ProgramCreek, 2021) as a proxy of API popularity and as a solution to objectively select the 25 API methods per system.

3.2. *Collecting code examples*

The next step in our experimental design is to collect code examples for the 100 selected API methods. We look for code examples on the web by querying on Google (on a private browser session) the API full name followed by the word "*example*". The word "*example*" is used to maximize the chance we find proper API examples, and not API documentation (Treude and Aniche, 2018). For instance, for the Guava API `Ints.concat`, we query for "*com.google.common.collect. Ints.concat example*". Thus, for each API, we manually collect 10 code examples returned by the query. We take special care to collect five didactic (*i.e.*, created for didactic purposes) and five real software examples (*i.e.*, originated from real software systems), thus, totaling 1000 examples, *i.e.*, 500 didactic and 500 real software. The collected code examples come from several programming websites, such as ProgramCreek (ProgramCreek, 2021), Stack Overflow (StackOverflow, 2021), and Baeldung (Baeldung, 2021) among many other that are often top ranked by the Google search engine (Treude and Aniche, 2018).

3.2.1. *Differentiating between didactic and real software examples*

Didactic examples typically present only the information needed to understand the API and do not include external context (Buse and Weimer, 2012), as illustrated in Fig. 4a (code extracted from geeksforgeeks.org). In contrast, real software code examples are originated from software systems, as illustrated in Fig. 4b (code extracted from codota.com). Real software code examples often contain extra statements and references to other APIs than the target one, being more complex, and difficult to understand and reuse (Buse and Weimer, 2012).

We perform a manual classification of the code examples as didactic or real software ones. Specifically, we rely on their definitions to classify the examples. For example, didactic code examples tend to be simpler and easier to understand than real software ones, which often contain extra statements and references to other APIs. As presented in Table 2, the didactic examples come from 113 programming websites, such as Stack Overflow,

11 https://spring.io.
12 https://www.programcreek.com/java-api-examples/index.php?action=index, https://www.programcreek.com/java-api-examples/index.php?action=top_java_methods.

(a) Didactic code example

```
public static void main(String[] args) {

     // Creating 2 Integer arrays
    int[] arr1 = { 1, 2, 3, 4, 5 };
    int[] arr2 = { 6, 2, 7, 0, 8 };

    // Using Ints.concat() method to combine
    // elements from both arrays into a single array
    int[] res = Ints.concat(arr1, arr2);

    // Displaying the single combined array
    System.out.println("Combined Array: " + Arrays.toString(res));
}
```

(b) Real software code example

```
public static IComplexNDArray repeat(IComplexNDArray n, int num) {
     List<IComplexNDArray> list = new ArrayList<>();
     for (int i = 0; i < num; i++)
         list.add(n.dup());
     IComplexNDArray ret = Nd4j.createComplex(list,
         Ints.concat(new int[] {num}, n.shape()));
     logCreationIfNecessary(ret);
     return ret;
}
```

Fig. 4. Code examples for the Guava API `Ints.concat`.

Table 2
Origin of the code examples (top-10 websites).

Pos	Didactic		Real software	
	Website	#	Website	#
1	stackoverflow.com	48	programcreek.com	407
2	baeldung.com	45	codota.com	33
3	tutorialspoint.com	36	javatips.net	25
4	alvinalexander.com	31	searchcode.com	12
5	geeksforgeeks.org	29	programtalk.com	8
6	javatutorialhq.com	17	javased.com	7
7	techiedelight.com	14	useof.org	3
8	howtodoinjava.com	13	zgrepcode.com	2
9	logicbig.com	12	javadocexamples.com	1
10	commons.apache.org	12	github.com	1
Distinct websites		113		10

Baeldung, Tutorialspoint, and GeeksforGeeks. For instance, Baeldung, Tutorialspoint, and GeeksforGeeks contain programming tutorials, articles, and guides, which typically include educational code examples and explanations on how to use certain APIs.[13] On the other hand, the real software examples come from only 10 programming websites, including ProgramCreek, Codota, JavaTips, and SearchCode. Those websites are hubs of code examples that are (automatically) mined from external software repositories like GitHub, Bitbucket, and GitLab.[14] The websites with real software code examples typically present dozens or hundreds of examples for the same API since they are automatically extracted from other sources.

To validate our manual classification, we randomly selected 280 out of the 1000 code examples in our dataset (95% confidence level and 5% confidence interval). Then, we invited eight software developers with distinct experience levels to classify those examples in didactic or real software. Each developer rated 35 code examples as didactic or real software ($8 \times 35 = 280$). Next, we compared their classification with ours. Fig. 5 presents the results of this analysis by developer and experience. Overall, the agreement between the developers' classification and ours is 80% (224 out of 280); Cohen's Kappa is 0.60, leading to close to substantial strength of agreement. Developers 1–5 have five or more years of experience, whereas developers 6–8 have up to five years of experience. Notice that, the agreement did not change significantly according to their experience. This way, we believe there is a natural subjective in the classification of the code examples as didactic or real software, however, the cases in which there is agreement are significantly more frequent.

3.2.2. Characterizing the code examples

We assess the characteristics of the code examples with respect to their size, documentation, readability, reusability, API references, and query similarity.

Size. We compute the size of the code examples in terms of (i) number of lines of code, (ii) number of tokens, and (iii) density of tokens (*i.e.,* number of tokens/number of lines of code). In lines of code, we count all lines in the code, including comments and blank lines. Small code may improve its understanding (Martin, 2009). Moreover, code examples should be concise and simple (Nasehi et al., 2012; Vincent, 2018).

Documentation. We compute the number and the ratio of comments in a code example by considering the inline comments (*i.e.,* // in Java). In contrast to multiline comments, inline comments are typically mixed with code and are more likely to include relevant explanations about the studied code snippets. Code comments are important to any piece of code (Lethbridge et al., 2003), however, they are even more relevant to code examples as they provide help to the developers understanding the API usage (Nasehi et al., 2012).

Readability. It is a human judgment of how easy a text is to understand (Buse and Weimer, 2009; Moreno et al., 2015). We rely on the metric proposed by Scalabrino et al. (2016, 2018) to evaluate the readability of a code example, which have a higher accuracy score when evaluated against other state-of-the-art models. This metric uses textual properties of source code that aid in characterizing its readability, including comments and identifiers consistency, narrow meaning identifiers, and textual coherence. We used the authors' implementation of the metric.[15] Given a code example, the readability metric produces values between 0 (low readability) and 1 (high readability). According to this metric, the didactic code example presented in Fig. 4a presents much higher readability (0.82) than the real software example shown in Fig. 4b (0.33), which seems quite reasonable.

Reusability. This metric assesses the facility to reuse a given code example. We adopt a modified version of the original metric of Moreno et al. (2015) to evaluate reusability:

$$Reuse = \begin{cases} \frac{\#\text{library object types}}{\#\text{object types}} & \text{if } \#object\ types > 0 \\ 1.0 & \text{otherwise} \end{cases} \quad (1)$$

where *#object types* is the total number of different object types used by the code example, and *#library object types* is the number of object types used by the code example and belonging to the target library (*i.e.,* Guava, Spring, or Apache Commons) plus the Java native object types (which can be reused by any code example). The reuse metric varies from 0 to 1: 0 means that all object types in the code example are custom or external objects (low reusability), while 1 indicates that all object types in the code example belong to the target library or that no object types are present in the code example (high reusability) (Moreno et al., 2015). The rationale is that reusing a code example that uses custom/external object types requires importing those objects into

[13] Didactic code: https://www.baeldung.com/convert-input-stream-to-a-file.

[14] Real software code: https://www.programcreek.com/java-api-examples/?class=com.google.common.collect.Lists&method=asList.

[15] Tool available at: https://dibt.unimol.it/report/readability.

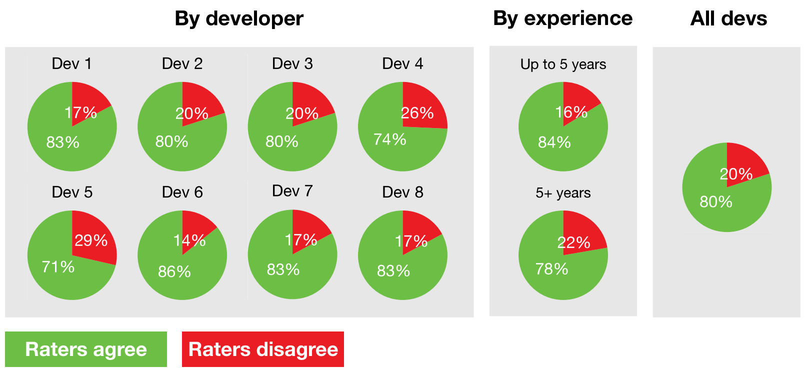


Fig. 5. Results of the survey with eight developers about the classification in didactic and real software code examples.

the production code, which represents an extra work (Moreno et al., 2015).

API references. We compute the number of API references (*i.e.*, references to the target API class and method) in the code examples. We also compute the ratio of API references (*i.e.*, number of API references/number of tokens), which measures how frequently a term occurs in a document (aka, Term Frequency Manning et al., 2008). Ideally, when looking for code examples on the web, developers may expect to see references to the desired API on the code.

Query similarity. We measure query similarity to the code examples with two metrics: cosine similarity[16] and soft cosine similarity.[17] Cosine similarity (Salton et al., 1975) is largely adopted in information retrieval and code search literature (Gu et al., 2018; Keivanloo et al., 2014; Niu et al., 2017) to compare the similarity between two documents (or two vectors) independently of their size. The soft cosine similarity (Sidorov et al., 2014) is a soft similarity measure: it assumes that similarity between features is known (*e.g.*, from a synonym dictionary).[18] For example, the words "play" and "game" are different but semantically related (Sidorov et al., 2014), thus in the soft cosine similarity, they are not considered completely distinct as in the traditional cosine similarity.

3.2.3. *Measuring the code examples*

Table 3 presents the median values of the metrics for all, didactic, and real software code examples. It also presents the statistical significance of the difference between the metric values of the didactic and real examples by applying the Mann–Whitney test at *alpha value* $= 0.05$. To show the effect-size of the difference between them, we compute Cliff's Delta (d). Note that the didactic and real software code examples are different with at least a small effect for all metrics unless tokens density. As expected, the didactic code examples are more readable (0.66 vs. 0.39) and reusable (1.0 vs. 0.78) than the real software ones; the difference is statistically significant with a large effect. Moreover, the median value of comments is zero for all, didactic, and real software code examples, but their third quartiles are 1, 3, and 0, while the mean values are 1, 1.59, and 0.54, respectively. Thus, as expected, the didactic code examples are more likely to have comments (the difference is also statistically significant with a small effect).

Table 3
Metric values of the collected code examples. *effect-size*: negligible for $d < 0.147$, small for $d < 0.33$, medium for $d < 0.474$, and large otherwise (Romano et al., 2006).

Metric	All	Didactic	Real software	p-value	Effect-size
Lines of code	13.0	11.0	15.00	<0.01	Small
Tokens	49.0	40.0	55.00	<0.01	Small
Tokens density	3.7	3.7	3.67	0.45	Negligible
Comments	0.0	0.0	0.0	<0.01	Small
Comments ratio	0.0	0.0	0.0	<0.01	Small
Readability	0.54	0.66	0.39	<0.01	Large
Reusability	1.0	1.0	0.78	<0.01	Large
API references	3.0	4.0	3.0	0.00	Small
API ratio	0.08	0.11	0.06	<0.01	Medium
Cosine similarity	0.26	0.32	0.19	<0.01	Medium
Soft cosine simi.	0.27	0.35	0.21	<0.01	Medium

3.3. *Indexing code examples*

After collecting and measuring the code examples, we move to the third step of our experimental design. Our goal here is to make each code example indexed by the Google search engine, so we can query and assess them afterward. Notice that this is not a trivial task; the Google search engine has many quality restrictions (How Search Algorithms Work, 2021): "*The indexing of your content by Google is determined by system algorithms that take into account user demand and quality checks*" (Introduction to Indexing, 2021).

For this purpose, we created a website and hosted each code example on a single webpage, totaling 1000 webpages with code examples. We embedded the code examples in `<pre>` tags so we have preformatted text, without any syntax highlighting. Also, the code examples are inside `<code>` tags to indicate that the text is a fragment of computer code. Using `<pre><code>` is the standard in HTML to represent multiple lines of code on webpages, as suggested by Mozilla: "*To represent multiple lines of code, wrap the <code> element within a <pre> element. The <code> element by itself only represents a single phrase of code or line of code*".[19] Indeed, this combination `<pre><code>` is adopted by programming websites like Stack Overflow to show code content.

In order to have the webpages indexed, we followed good practices suggested by Google (Introduction to Indexing, 2021), for instance, we provided useful content, we managed our website via the Search Console,[20] we submitted the pages via a

[16] We relied on the Python lib sklearn, method cosine_similarity.
[17] We relied on the Python lib gensim, method softcossim.
[18] Adopted dataset: https://github.com/RaRe-Technologies/gensim-data/tree/fasttext-wiki-news-subwords-300.

[19] https://developer.mozilla.org/en-US/docs/Web/HTML/Element/code.
[20] https://support.google.com/webmasters/answer/9128668?hl=en.

sitemap, we made the webpages mobile responsive,[21] we performed (and passed) mobile-friendly tests,[22] and we created human intelligible URLs.[23] By following these procedures, our website was fully indexed by Google in two weeks — this information is provided by the Google Search Console. Being fully indexed means that the 1000 webpages can appear in Google search results.

In addition to the 1000 webpages with single examples, we also submitted to Google, for each API, one webpage including the 10 previously selected API code examples (for a total of 100 additional webpages). These webpages with multiple examples are intended to support answering RQ1, which addresses the ranking of single and multiple code examples. The 100 webpages with multiple examples were also indexed by Google. Thus, in total, we have 1100 (1000 + 100) indexed webpages. That is, each API has 11 webpages with code examples: 10 webpages with single code examples and one with multiple examples.

3.4. Querying code examples

After being fully indexed by Google, we are able to query the code examples. To automate this task, we created a Google Custom Search Engine (Creating a Programmable Search Engine, 2021) for our website. It allows us to programmatically query the content of our website via the official Google Search API (Google Search API, 2021). By relying on the Google Search API, we ensure that all queries are subjected to the same query environment, avoiding being subjected to manual queries in the browser.

Having set up the search environment, we need to perform the queries to retrieve the API code examples. As stated in Section 2, typically, a code search query consists of API tokens, *i.e.,* class and method names. For example, developers may query for "*File mkdirs*" to find code examples of `File.mkdirs`. Thus, our queries followed the format "<class-name> <method-name>", as also previously adopted by the code search literature (Niu et al., 2017).

Finally, with the support of the Google Search API, we performed 1500 queries in a period of 15 days in June 2019; each day we run 100 queries, that is, one query for each API. We performed this analysis for 15 days because it was the necessary period for the returned results to become stable. *The results reported in this paper refer to the 15th day*. Moreover, we replicated this analysis 16 months later (in November 2020) to assess whether results have significantly changed over time (*e.g.,* due to possible major changes in Google search algorithms). We restricted each query to the context of its own API in order to facilitate the analysis of returned results.[24] This way, each query may return at most 11 ranked results: 10 webpages with single code examples and one webpage with multiple code examples. In RQ1, we consider all webpages in the analysis, while in RQs 2, 3, and 4 we only consider the webpages with single examples.

3.5. Assessing query results

Next, we describe how we answer each research question.

[21] https://developers.google.com/search/mobile-sites.

[22] https://search.google.com/test/mobile-friendly.

[23] https://support.google.com/webmasters/answer/76329?hl=en.

[24] Restricted queries can be done with the "`site:`" command provided by Google.

3.5.1. Single vs. multiple code examples (RQ1)

Ideally, one may expect that webpages with multiples examples should be top ranked due to the following reasons. *First*, when looking for code examples, it is intuitively better to see a webpage with multiple code examples rather than only one. Indeed, most programming websites present several code examples per page. *Second*, in practice, developers often inspect multiple results of different usages to learn from Gu et al. (2018) and Raghothaman et al. (2016), thus, having a webpage with several examples facilitates this task.

We rely on three metrics to assess whether webpages with multiple examples are top ranked and measure their performance: FRank, SuccessRate@k, and Mean Reciprocal Rank (MRR). These metrics are typically adopted in information retrieval and code search literature (Gu et al., 2018; Keivanloo et al., 2014; Lv et al., 2015; Raghothaman et al., 2016; Ye et al., 2014).

FRank is the rank position of the hit in the query results (Raghothaman et al., 2016). A smaller value means lower inspection effort for finding the desired result (Gu et al., 2018). We recall that the hit is the webpage with multiple code examples of an API, which would be the best answer for a developer looking for code examples.

SuccessRate@k measures the percentage of queries for which more than one correct result could exist in the top k ranked results (Gu et al., 2018; Keivanloo et al., 2014; Ye et al., 2014). In this analysis, we consider the webpage with multiple code examples as the only correct result for a query. We measure it as follows:

$$SuccessRate@k = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\delta(FRank_q \le k) \tag{2}$$

where Q is a set of queries, $\delta()$ is a function that returns 1 if the input is true and 0 otherwise. A better code search engine should help developers to discover the needed code example by inspecting fewer returned results. In this sense, the higher the metric value, the better the code search performance (Gu et al., 2018). We evaluate SuccessRate@k when k value is 1, 2, and 3. Commonly, k is evaluated with 1, 5, and 10, as these are typical sizes that users would inspect (Gu et al., 2018). However, in our study, each query may return at most 11 results, thus, to be rigorous, we only assess the top 3 search results (*i.e., k* values from 1 to 3).

MRR is the average of the reciprocal ranks of results of a set of queries Q (Gu et al., 2018; Lv et al., 2015). The reciprocal rank of a query is the inverse of the rank of the first hit result (Grechanik et al., 2010; Gu et al., 2018). The higher the MRR value, the better the code search performance (Gu et al., 2018). We measure it as follows:

$$MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{FRank_q} \tag{3}$$

Importantly: In the following research questions, our goal is to assess the code examples **individually**. Thus, in RQs 2, 3, and 4, we focus on the 1000 code example webpages with single examples and exclude the 100 webpages with multiple examples from the analysis.

3.5.2. Didactic vs. real software code examples (RQ2)

In this research question, we compute the ranks of the didactic and real software code examples. We analyze the statistical significance of the difference between the metric values of the didactic and real examples by applying the Mann–Whitney test at *alpha value* = 0.05. To show the effect-size of the difference between them, we compute Cliff's Delta (d). We interpret the effect-size values as negligible for $d < 0.147$, small for $d < 0.33$, medium for $d < 0.474$, and large otherwise (Romano et al., 2006).

3.5.3. Top ranked vs. bottom ranked code examples (RQ3)

We compare the top and the bottom ranked code examples in three scenarios: Top 1 vs. Bottom 1, Top 3 vs. Bottom 3, and Top 5 vs. Bottom 5. The first is more strict as it contrasts the very top with the very bottom returned code example in each query. The last considers all code examples and compares the top half with the bottom half code examples. We apply the Mann–Whitney test and Cliff's Delta effect-size to verify the significance of the difference.

3.5.4. Prediction of the code examples and importance of the factors (RQ4)

In our last research question, we assess the importance of multiple metrics to distinguish the top and the bottom ranked code examples. Specifically, we rely on the Random Forest binary classifier (Liaw et al., 2002) to predict whether a code example will be top or bottom ranked according to the 12 metrics described in the previous subsections (*i.e.,* lines of code, comments, readability, origin, etc.). We select Random Forest due to several advantages, such as being robust to noise and outliers (Tian et al., 2014). Moreover, the Random Forest classification power has successfully been used to automate many Software Engineering tasks (Dias et al., 2015; Peters et al., 2013; Tian et al., 2014). We use 10 times 10-fold cross-validation to evaluate the model effectiveness. We train and test our classifier in the three datasets of RQ3: Top 1 vs. Bottom 1 (200 code examples), Top 3 vs. Bottom 3 (600 code examples), and Top 5 vs. Bottom 5 (1000 code examples); each dataset has half of top ranked and half of bottom ranked, thus, we run the Random Forest binary classifier 30 times, 10 on each dataset to predict top and bottom ranked ones. Following the 10-fold cross-validation, for each dataset, we train a subset of examples and then evaluate on another subset.

Before running the classifier, we perform a correlation analysis on the 12 selected metrics. Correlated metrics may bias the results (Jiarpakdee et al., 2019; Niu et al., 2017), so it is important to detect and remove them. For this purpose, we verify a correlation with a Spearman test on all metrics. The values of Spearman coefficient ranges from −1 to 1. Values close to −1 or 1 represent a high correlation, while values close to 0 indicates no correlation. After, finding correlated metrics, we select one representative, as typically performed by the literature (Niu et al., 2017).

Finally, to assess the classifier's effectiveness, we compute precision, recall, and AUC (area under the curve), which are commonly adopted metrics in classification problems (Dias et al., 2015; Kim et al., 2008). Precision and recall measure the correctness and completeness, respectively, of the classifier in predicting whether a code example will be top ranked. AUC is a commonly used measure to judge predictions in binary classification problems: it refers to the area under the Receiver Operating Characteristic (ROC) curve. According to the literature, AUC $\geq$ 70% is considered reasonably good (Lessmann et al., 2008; Thung et al., 2012; Tian et al., 2014). Lastly, to assess the most important factors, we report the importance of the metrics according to the Gini Importance (Jiarpakdee et al., 2019).

4. Results

4.1. RQ1: How are single and multiple code examples ranked?

Fig. 6 presents the performance of the Google search engine for the metric FRank: 82% of the webpages with multiple code examples (*i.e.,* the hits) are top ranked. That is, webpages with multiple code examples are more likely to be top ranked by the Google search engine than webpages with single examples.

Moreover, this ratio tend to be constant over time, as presented in Fig. 7. We find no major difference during the first 15 days of analysis in June 2019 (black bars): it started with 79% on the first day and increased/decreased by 1% in the following days until remain constant at 82%. Besides, to verify whether those results are stable over a long period, we report the same analysis 16 months later (in November 2020, blue bars). We notice that the FRank had a small growth, from 82% to 88%.

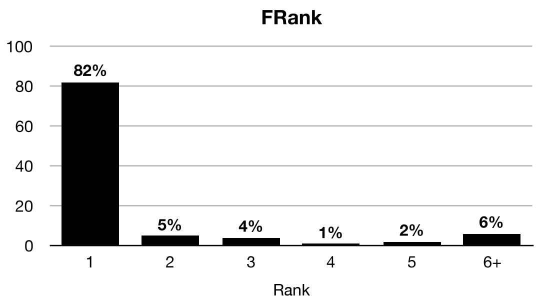


Fig. 6. Rank of the first webpage with multiple examples (FRank).

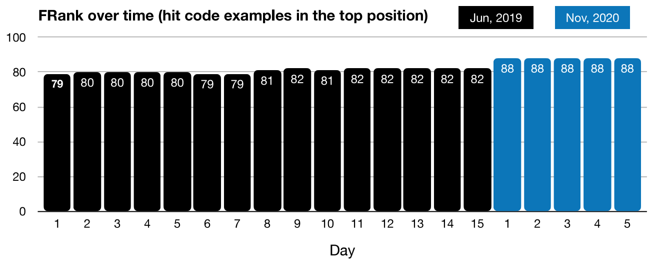


Fig. 7. FRank over time. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 4
Overall accuracy. SR: Success Rate. MRR: Mean Reciprocal Rank.

Date	SR@1	SR@2	SR@3	MRR
Jun, 2019	82%	87%	91%	87%
Nov, 2020	88%	92%	95%	93%

Table 4 presents the performance according to the metrics SuccessRate@k and Mean Reciprocal Rank (MRR) in the two analyzed periods (last day of each period). The columns SuccessRate@1-3 present the results for SuccessRate@k when *k* is 1, 2, and 3. The results suggest that the overall performance of the Google search engine is high to find the webpage with multiple code examples. For example, SuccessRate@3 value is 91%: this means that for 91% of the queries, the webpages with multiple examples can be found within the top 3 returned code examples. Finally, the MRR at 87% also represents a high number, reinforcing that the hits are very often ranked in the first positions. Notice that in November 2020, the metric values have increased, meaning that the hits are even more consolidated in the top positions 16 months later.

Summary of RQ1: Webpages with multiple code examples are more likely to be top ranked by Google than webpages with single examples. 82% of the webpages with multiple code examples are top ranked, while 91% are among the top-3 returned results. Over time, webpages with multiple code examples tend to be even more consolidate in top positions.

As detailed in our Study Design, from now on, we only consider the webpages with single examples, that is, we do not consider the web pages with multiple code examples in the following three analyses (RQ2, RQ3, and RQ4).

Table 5
Didactic vs. real software code examples. *D*: didactic. *RS*: real software. *pv*: *p*-value. *es*: effect-size. *N*: negligible, *S*: small, *M*: medium, and *L*: large.

Metric	Rank 1				Rank 10			
	D	RS	pv	es	D	RS	pv	es
Lines of code	17.0	18.0	0.38	N	6.5	13.5	<0.01	L
Tokens	66.0	72.0	0.50	N	20.0	46.0	<0.01	L
Tokens density	**4.0**	**4.0**	0.87	**N**	**3.33**	**3.52**	0.79	**N**
Comments	2.0	0.0	<0.01	L	0.0	0.0	0.63	N
Comments ratio	0.15	0.0	<0.01	L	0.0	0.0	0.38	N
Readability	0.70	0.26	<0.01	L	0.67	0.43	<0.01	L
Reusability	1.0	0.7	<0.01	L	1.00	0.75	<0.01	M
API references	**7.0**	**7.0**	0.62	**N**	**2.0**	**2.0**	0.54	**N**
API ratio	0.11	0.12	0.33	N	0.12	0.06	<0.01	L
Cosine similarity	0.37	0.38	0.36	N	0.27	0.18	<0.01	L
Soft cosine simi.	0.43	0.41	0.28	N	0.33	0.19	<0.01	M

4.2. RQ2: How are didactic and real software code examples ranked?

In this research question, we assess the origin of the code examples. Fig. 8 summarizes the results of 100 queries and their 1000 returned code examples. Each stacked column shows one position of the rank with the ratio of code examples. Overall, the didactic code examples are more concentrated on higher positions. For instance, position 1 has 67% of didactic and 33% of real software code examples. Positions 2 and 3 also have more didactic code examples than real software (60% vs. 40% and 53% vs. 47%, respectively). From positions 4 to 10, we notice a higher proportion of code examples coming from real software. For example, position 10 includes 60% of real software and 40% of didactic.

Table 5 presents a deeper analysis of the results shown in the previous figure. We assess the metrics of the didactic and real software code examples in the ranking position 1 and 10. For each metric, we present the median values, the *p-values* for Mann–Whitney, and the effect-size for Cliff's Delta. For instance, on the median, the didactic code examples in rank 1 have 17 lines of code, while the real software ones have 18 (the difference is only negligible). In rank 10, the didactic code examples are smaller than the real software ones (6.5 lines vs. 13.5, the difference is statistically significant with a large effect). As lines of code, most metrics change their proportions when we compare ranks 1 and 10, so we cannot derive any concrete conclusion of their prevalence. However, two metrics remain the same and are independent of the ranking positions: tokens density and API references (presented in bold in Table 5). Tokens density has values 4.0/4.0 and 3.33/3.52 for the didactic and real software examples in ranks 1 and 10, respectively, both with negligible differences. Similarly, the number of API references is equal independently of ranking positions (7.0/7.0 APIs in rank 1; 2.0/2.0 APIs in rank 10).

Interestingly, Table 3 in Section 3 shows that overall the didactic and real software code examples are equal only in 1 out of 11 metrics (*i.e.,* tokens density, with a negligible difference). In contrast, as presented in Table 5, the top ranked didactic and real software examples are equal in 7 out of 11 metrics (*i.e.,* their differences are negligible in rank 1). That is, independently of being didactic or real software, top ranked code examples are similar in terms of size (3 metrics), API references (2 metrics), and query similarity (2 metrics).

Summary of RQ2: Overall, code examples created for didactic purposes are more likely to be top ranked by the Google search engine than code examples originated from real software systems. However, this is likely to happen simply because they have more API references and tokens density (*i.e.,* not because didactic code examples are more readable or reusable).

Table 6
Top 1 vs. bottom 1 ranked code examples.

Metric	Top 1 vs. Bottom 1				
	Median top	Median bottom	p-value	Effect-size	
Lines of code	17.5	9.0	<0.01	Large	███
Tokens	69.0	33.0	<0.01	Large	███
Tokens density	4.04	3.50	<0.01	Medium	██░
Comments	1.0	0.0	<0.01	Medium	██░
Comments ratio	0.10	0.0	<0.01	Medium	██░
Readability	0.57	0.53	0.17	Negligible	░░░
Reusability	1.0	1.0	0.09	Negligible	░░░
API references	7.0	2.0	<0.01	Large	███
API ratio	0.11	0.08	<0.01	Small	█░░
Cosine similarity	0.38	0.08	<0.01	Large	███
Soft cosine simi.	0.42	0.21	<0.01	Medium	██░

Table 7
Top 3 vs. bottom 3 ranked code examples.

Metric	Top 3 vs. Bottom 3				
	Median top	Median bottom	p-value	Effect-size	
Lines of code	16.0	10.0	<0.01	Medium	██░
Tokens	59.0	35.0	<0.01	Medium	██░
Tokens density	3.94	3.62	<0.01	Small	█░░
Comments	0.0	0.0	<0.01	Small	█░░
Comments ratio	0.00	0.00	<0.01	Small	█░░
Readability	0.55	0.54	0.18	Negligible	░░░
Reusability	1.0	1.0	0.12	Negligible	░░░
API references	5.0	3.0	<0.01	Large	███
API ratio	0.10	0.09	0.02	Negligible	░░░
Cosine similarity	0.32	0.09	<0.01	Small	█░░
Soft cosine simi.	0.34	0.24	<0.01	Small	█░░

4.3. RQ3: What are the characteristics of top ranked code examples?

Tables 6, 7, and 8 compare the characteristics of the top and the bottom ranked code examples. It shows the median metric values of the top and the bottom code examples, the *p-value* for the Mann–Whitney test, and the *effect-size* for the Cliff's Delta.

Overall results. We find many differences between the top and the bottom ranked code examples: independently of the comparison, top ranked code examples are statistically larger and have more API references. However, the disparity is broader in the comparison of Top 1 vs. Bottom 1, *i.e.,* when we contrast the first and the last returned code example of each query (Table 6). In this case, there are four metrics in which the differences are statistically significant with large effect (lines of code, tokens, API references, and cosine similarity), four metrics with medium effect (tokens density, comments, comments ratio, and soft similarity), one with small effect (API ratio), and two negligible (readability and reusability). The comparison of Top 3 vs. Bottom 3 shows a statistically significant difference with at least a small effect in 8 metrics (Table 7). Finally, the comparison of Top 5 vs. Bottom 5 still presents some differences, but mostly with a small effect (Table 8). Next, we further discuss each metric category.

Size. Top ranked code examples are larger in the number of lines of code and tokens in the three comparisons. In the first comparison, the top ranked code examples have, on the median, 17.5 lines of code and 69 tokens, while bottom ranked are much smaller, with 9 lines and 33 tokens. In the other two comparisons, the size differences are lower, although still statistically significant. Fig. 9 breaks the code examples in two groups: small (LOC $\leq$ 10) and

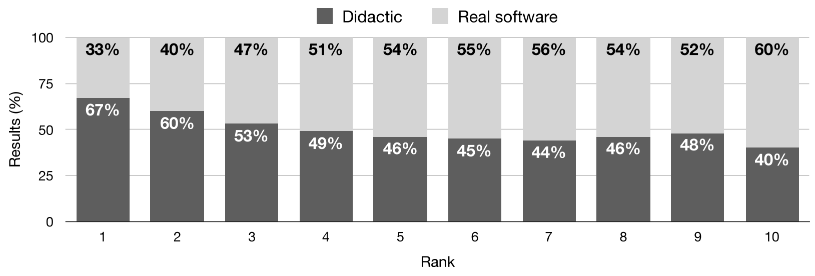


Fig. 8. Summary of the rank (didactic vs. real software).

Table 8
Top 5 vs. bottom 5 ranked code examples.

Metric	Top 5 vs. Bottom 5				
	Median top	Median bottom	p-value	Effect-size	
Lines of code	15.0	12.0	<0.01	Small	▮▯
Tokens	58.0	40.0	<0.01	Small	▮▯
Tokens density	3.82	3.62	<0.01	Negligible	▯
Comments	0.0	0.0	<0.01	Small	▮▯
Comments ratio	0.0	0.0	<0.01	Small	▮▯
Readability	0.53	0.54	0.61	Negligible	▯
Reusability	1.0	1.0	0.23	Negligible	▯
API references	5.0	3.0	<0.01	Medium	▮▮▯
API ratio	0.09	0.07	0.01	Negligible	▯
Cosine similarity	0.30	0.07	<0.01	Small	▮▯
Soft cosine simi.	0.31	0.23	<0.01	Small	▮▯

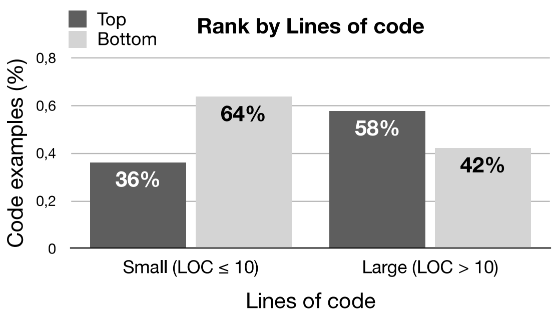


Fig. 9. Rank by lines of code. Dataset: Top 5 vs. Bottom 5.

large (LOC > 10). It makes it clear that small code examples are more likely to be bottom ranked, while large ones happen more in the top.

Documentation. We find more code comments in the top ranked code examples than on the bottom ones. Both documentation metrics (*i.e.*, comments and comments ratio) are statistically significant with at least a small effect in the three comparisons. However, their median values are equal to zero in two comparisons (Top 3 vs. Bottom 3 and Top 5 vs. Bottom 5).

Readability and reusability. We detect no difference between the top and the bottom ranked code examples regarding readability and reusability metrics: 6 out of 6 comparisons have negligible effect. In the comparison Top 1 vs. Bottom 1, despite the top ranked examples have almost double the size of the bottom ranked ones (17.5 vs. 9 LOC), they have equivalent readability and reusability. Thus, overall, top and bottom code examples are equally reusable and readable.

API references. The top ranked code examples have more references to the target API than the bottom ones in the three comparisons, all statistically significant with at least medium effect. For instance, on the median, the top 1 code examples have 7 API references, while the bottom 1 code examples only have 2. However, when we look at the ratio of API references, the differences are negligible in two cases and small in one. That is, the absolute number of API references is more important than the frequency to differentiate the top and bottom ranked code examples.

Query similarity. Lastly, we find that the top ranked code examples are more similar to their queries than the bottom ones in all comparisons for both cosine similarity and soft cosine similarity. This is not surprising because query similarity is an important factor in information retrieval to rank the documents (Gu et al., 2018; Niu et al., 2017). Moreover, the soft cosine similarity values are higher than the cosine similarity. For instance, on the median, the cosine similarity values for the top 1 and bottom 1 are 0.38 and 0.08, respectively, while the soft cosine similarity values are 0.42 and 0.21. This occurs because the soft cosine similarity is a more flexible similarity measure that detects semantically related words (Sidorov et al., 2014).

Summary of RQ3: Overall, top ranked code examples by Google are larger, have more comments and API references, and are more similar to the input query. Readable and reusable code examples are not necessarily top ranked.

4.4. RQ4: To what extent can we predict that a code example will be top ranked? What are the most important characteristics?

In this final RQ, we investigate whether we can predict top ranked code examples as well as the most important factors on the prediction. Before running the Random Forest classifier, we detect the correlated metrics. Fig. 10 presents the Spearman correlation structure of our 12 metrics. We find four groups of correlated metrics: (1) is_didactic and reusability, (2) comments and comments ratio, (3) lines of code and tokens, and (4) API references, soft cosine similarity, API ratio, and cosine similarity. To represent each group we select (1) reusability, (2) comments, (3) lines of code, and (4) API references. Those metrics are selected because they are easier to compute/understand or to judge code quality. After the selection, we have 6 uncorrelated metrics (reusability, comments, lines of code, API references, tokens density, and readability), which are used in our classifier.

Table 9 presents the effectiveness of the Random Forest classifier to predict top and bottom ranked code examples. We report three configurations: Top 1 vs. Bottom 1, Top 3 vs. Bottom 3, and Top 5 vs. Bottom 5. The three classifiers produce AUC $\geq$ 70%, which is considered reasonably good (Lessmann et al., 2008; Thung et al., 2012; Tian et al., 2014). The prediction is less effective in the Top 5 vs. Bottom 5 configuration, which has precision: 65%, recall: 60%, and AUC: 71%. On the other hand, the prediction is more effective to distinguish the top 1 and the bottom 1 code

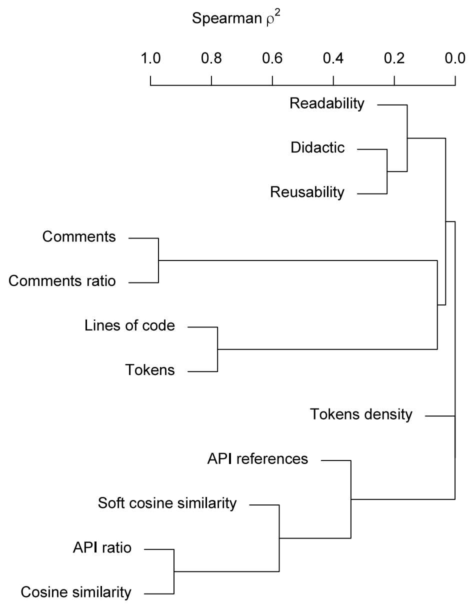


Fig. 10. Correlation structure of the 12 adopted metrics.

Table 9
Effectiveness of the Random Forest classifier.

Measure	Top 1 vs. Bottom 1	Top 3 vs. Bottom 3	Top 5 vs. Bottom 5
Precision	0.79	0.72	0.65
Recall	0.70	0.66	0.60
AUC	0.89	0.81	0.71

examples. In this case, precision is 79%, recall is 70%, and AUC is 89%.

Fig. 11 presents the distribution of the importance according to the Gini Importance (Jiarpakdee et al., 2019). Overall, the three most important metrics are API references, tokens density, and lines of code, which are metrics related to term frequency and size. The least important metrics are comments and reusability. Interestingly, the most important metrics are general measures (*i.e.*, they can measure the size and term frequency of any textual document), while the least important ones are directly related to code (*i.e.*, they only make sense when applied to source code). In general webpages, factors as webpage size[25] and term frequency[26] (*i.e.*, the matching between webpage content and search query) are also among the most important to characterize the relevance of webpages and related to top ranked ones. Therefore, it seems Google treats code as text and applies general ranking heuristics for code. *We reinforce that those results refer to our controlled environment and do not necessarily apply to Google in practice, which is subjected to many other factors not covered by our research.*

Summary of RQ4: Summary of RQ4: We can predict top ranked code examples with a good confidence level: in the best scenario, precision is 79%, recall is 70%, and AUC is 89%. The most important metrics are generic measures (*e.g.*, term frequency and size) and not associated with code quality.

[25] https://backlinko.com/google-ranking-factors, https://www.bluecorona.com/blog/google-ranking-factors-2018.
[26] https://www.google.com/search/howsearchworks/algorithms.

5. Discussion and implications

Based on our results, we provide implications for practitioners and researchers. First, we present guidelines that can be applied by practitioners to improve code example webpages of programming websites. Then, we present insights to code search researchers.

5.1. For practitioners

Google encourages the modification of websites to improve user experience and performance in search results: "*You should build a website to benefit your users, and any optimization should be geared toward making the user experience better. One of those users is a search engine, which helps other users discover your content.*" (Search Engine Optimization (SEO) Starter Guide, 2021). This way, we provide guidelines that can be applied by practitioners to improve programming websites to both users and search engines.

Present multiple code examples. Developers often inspect multiple results of different usages to learn from Gu et al. (2018) and Raghothaman et al. (2016), thus, having webpages with several examples can facilitate this task. We find that Google prioritizes webpages with multiple code examples rather than with single examples. Specifically, in this study, the webpages with 10 code examples were detected to be higher ranked. *Thus, practitioners in charge of maintaining programming websites should strive to produce webpages with multiple code examples (10 in our experiment). In addition to providing a better user experience (Gu et al., 2018; Raghothaman et al., 2016), this practice can improve performance in search results.*

Prefer to show didactic code examples. Didactic code examples are written for educational purposes and focus on the information needed to understand the API (Buse and Weimer, 2012). We find that didactic code examples are more likely to be higher ranked by the Google search engine. *This way, we advise that programming websites should focus on presenting didactic code examples. They are naturally easier to understand (Buse and Weimer, 2012) and have more chances to be top ranked. This practice is already followed by some popular programming websites* (*e.g.*, Baeldung, 2021; GeeksforGeeks, 2021; Tutorialspoint, 2021).

Avoid micro and uncommented code examples. Code examples that are larger and more commented have better performance than smaller and less commented ones (with a statistically significant difference). For instance, code examples with less than 10 lines are more likely to be lower ranked (see Fig. 9). *Thus, to improve user experience and facilitate content discovery, programming websites should avoid very small and uncommented code examples.*

5.2. For researchers

Novel metrics relevant to search engines and software engineering. We provide empirical evidence that readable (Buse and Weimer, 2009; Scalabrino et al., 2016, 2018), and reusable (Moreno et al., 2015) code examples are not necessarily top ranked by Google (at least not in our controlled environment). For instance, the adopted readability metric is effective in predicting developers' readability judgments (Scalabrino et al., 2016, 2018), however, the most readable code examples according to this metric are not higher ranked. The same is true for reusability: higher and lower ranked code examples are equally reusable. We recognize, however, that there are other code quality metrics such as code understandability (Scalabrino et al., 2017), security (Fischer et al., 2017; Meng et al., 2018), and API usage

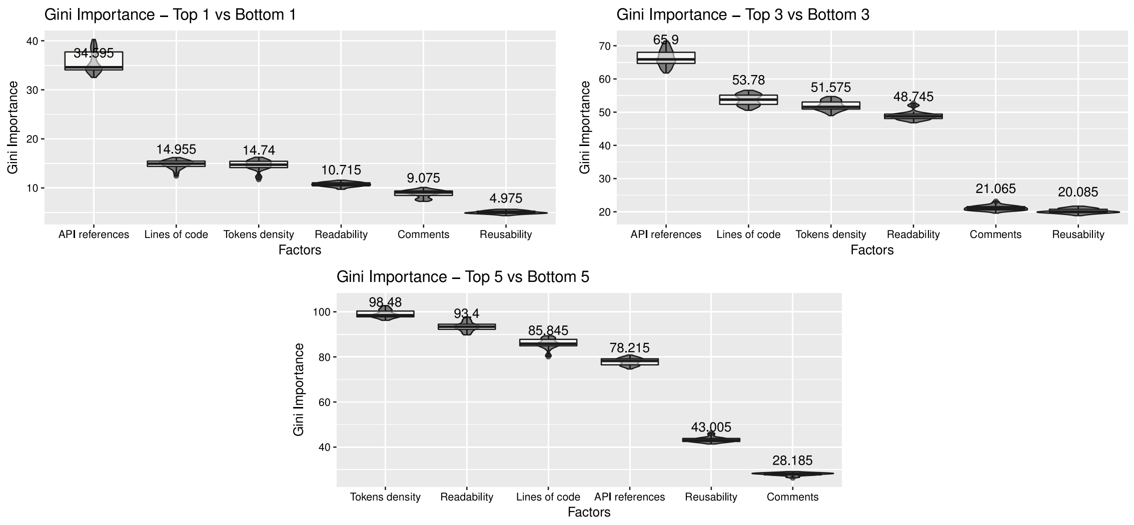


Fig. 11. Most important factors as measured by Gini Importance.

correctness (Zhou and Walker, 2016; Zhang et al., 2018) that are not evaluated in this paper and can be further investigated. Furthermore, we find that generic measures (as term frequency and size) are more important than code quality ones to rank code examples. Indeed, the code search literature shows that best ranked code examples may be associated with several factors, for instance: query similarity and term frequency (Niu et al., 2017); readability and reusability (Moreno et al., 2015); and popularity and similarity (Keivanloo et al., 2014). That is, so far, there is no standard solution. We contribute to this research field by showing that, when code is indexed in isolation, generic factors prevail in Google, as one would expect since it is a general web search engine. *Thus, our results may pave the way for the proposal of novel metrics that can assess code quality but also be friendly to search engines.*

Reusing online code examples. Code examples illustrate how programming problems can be solved (Keivanloo et al., 2014; Menezes et al., 2019). In practice, developers are likely to reuse such code snippets into production software (Fischer et al., 2017; Yang et al., 2017). However, recent studies on Stack Overflow show that posted code examples can be insecure and contain usage violations (Fischer et al., 2017; Meng et al., 2018; Ragkhitwetsagul et al., 2019; Zhang et al., 2018), hence, they should be reused with care. Our findings on Google search are aligned with the ones on Stack Overflow code examples (Fischer et al., 2017; Meng et al., 2018; Ragkhitwetsagul et al., 2019; Zhang et al., 2018) in the sense that both platforms have not considered code-related metrics, which raises some concerns about reusing online code examples. *When building the next generation of code search engines or API usage example mining techniques, researchers should also consider those quality metrics.*

Spotting the best code examples from code repositories. In the previous subsection, we advised using didactic code examples and avoiding the ones extracted from real software projects. Indeed, this is done by some programming websites (Baeldung, 2021; GeeksforGeeks, 2021; Tutorialspoint, 2021), but only for certain APIs, such as the most popular ones. That is, manually creating didactic code examples for all existing APIs is a task that may not scale. Thus, a solution to automate the selection of code examples is to mine and rank them from software repositories (Buse and Weimer, 2012; Keivanloo et al., 2014; Moreno et al., 2015; Hora, 2021a). *Based on our results, we recommend that mining solutions to spot examples should focus on finding code that is closer to didactic ones, as those are easier to understand and reuse, and are more likely to be top ranked by the studied search engine.*

6. Threats to validity

Generalization of the results. We assess the Google search engine and code examples implemented in Java. Google dominates the web, with more than 92% of the search market share (Search Engine Market Share Worldwide, 2021), while Java is among the most popular language nowadays. Despite these observations, our findings – as usual in empirical software engineering – cannot be directly generalized to other search engines (*e.g.*, bing, Yahoo!, Baidu) nor to code examples written in other languages.

Association is not causation. We study factors associated with top and bottom ranked code examples. We note, however, that association does not imply causation (Couto et al., 2014). More advanced statistical analysis, *e.g.*, causal analysis (Retherford and Choe, 2011), can be used to extend our study.

Format of the query. In this study, we adopt the format "<class-name> <method-name>" when we are querying code examples in the indexed webpages. We rely on this format because it is also adopted in the code search literature (Niu et al., 2017). Moreover, this format is the smallest one possible and the one that can be consistently applied to distinct the code examples, *i.e.*, each API has its own search query. However, in practice, developers are free to type in natural language and our results do not apply to those cases; more research should be performed in that direction.

Adopted metrics. We compute 12 metrics from the code examples, related to size, documentation, readability, reusability, frequency, similarity, and origin. These metrics provide general and quality assessments. They are also commonly adopted in the code search literature (Buse and Weimer, 2012; Gu et al., 2018; Keivanloo et al., 2014; Moreno et al., 2015). We recall, however, that readability does not necessarily mean understandability (Scalabrino et al., 2017). All in all, even if other measures could be adopted, we are aligned to previous code search studies.

High reusability values. The reusability metric produces overall high values, as presented in Table 3. We find that 546 code examples have a reusability score of 1.0. However, the majority of those are didactic examples: 70% (382 out of 546) are didactic, whereas 30% (164 out of 546) are real software code examples. This is expected because the didactic code examples are created for educational purposes and ideally should be easy to copy, paste, and run, as confirmed by their high reusability scores. On the other hand, as also expected, the reusability is lower for the real software code examples. Moreover, it is important to recall that we do not have the `import` statements in the code examples (they are rarely presented in their original websites), thus, our solution to compute reusability relies only on the used class names.

Influence of other factors. Several factors may influence the Google search result: reputation, domain, location, query language, usability, content quality, etc. (Furnell and Evans, 2007; Google General Guidelines, 2021; Hannak et al., 2013; Kliman-Silver et al., 2015). Some of these factors are out of control of the webpage owner, for instance, website reputation. We overcome this threat by hosting all code examples on the same website, so they have an equivalent reputation and are under the same domain. Moreover, other factors such as location and query language are also controlled, since all queries are performed with the Google Search API (Google Search API, 2021) and subjected to the same query environment. Our webpages are also mobile responsive, thus, they all have equivalent usability. In this study, we focus on the content, which is the factor that varies among our code example webpages and that the website owner can directly control, change, and improve.

Method and HTML file name. Many method names are simple and can appear as part of other method names, identifiers, or comments in the source code, *e.g.,* `Ints concat` (Guava), `BufferedReader close/read/skip` (Java SE), `ArrayList add/get/size` (Java SE), among others. Thus, further studies should be performed to assess whether simple and common method names can affect the ranking. Moreover, as our HTML file names include the method full qualified names, studies should be performed to assess whether the HTML file names can affect the ranking. However, it is important to recall that all webpages in our experiment are subjected to the very same naming convention. This way, even if the HTML file's name has some effect on the search results, all the search results would be similarly affected.

Change in Google algorithms. Google tends to change its algorithms frequently.[27] Thus, it is possible that running our experiments again may not lead to the very same results. To assess this threat, we performed two analyses in RQ1. First, in June 2019, we initially run the queries for 15 consecutive days until the FRank remains constant; this happened in the last five days when the FRank was 82% (see Fig. 7). Second, 16 months later, in November 2020, we run the very same queries during five consecutive days; in this case, the FRank was already constant at 88% and did not change during the analyzed period. This way, we observe that the FRank has grown up from 82% to 88%, meaning that the webpages with multiple code examples (*i.e.,* the hits) were consolidated in top positions. Notice that those changes are expected due to the evolution of Google algorithms and changes in our page indexes. In the analyzed time window (16 months), however, those changes did not largely affect our results.

7. Related work

Several commercial code search tools exist in the market. Nowadays, it is possible to navigate in code search tools, such as SearchCode (SearchCode, 2021), ProgramCreek (ProgramCreek, 2021), and Krugle (krugle, 2021). Code is also easy to find in online version control platforms, such as GitHub. Over time, other code search tools were discontinued, such as codase (Codase, 2021) and OpenHub Code (OpenHub, 2021) (previously known as Koders and ohloh). The Google Code Search is perhaps the most known case of a code search engine that was discontinued to the public (Google Code Search, 2021). Several code search engines and ranking solutions are proposed by the research community (Bajracharya et al., 2006; Brandt et al., 2010; Buse and Weimer, 2012; Grechanik et al., 2010; Gu et al., 2018; Hora, 2021a; Hora and Valente, 2015; Keivanloo et al., 2014; Kim et al., 2010; McMillan et al., 2012; Moreno et al., 2015; Niu et al., 2017; Stylos and Myers, 2006). Sourcerer (Bajracharya et al., 2006), for example, indexes millions of lines of Java code and maintains a public database with this data. Gu et al. (2018) propose an approach that uses deep learning to rank code examples, in which semantically related words can be recognized. Moreno et al. (2015) present Muse, an approach for mining and ranking code examples. Keivanloo et al. (2014) provide a low complexity approach to detect code examples that can be adopted by code search engines. Niu et al. (2017) present a code search approach based on machine learning techniques. Buse and Weimer (2012) propose a technique for mining and synthesizing representative readable documentation of APIs. None of these studies, however, analyze the Google search engine.

In a related line, Montandon et al. (2013) provide APIMiner, a tool that instruments the API documentation with concrete usage examples. Kim et al. (2013) improve documentation by generating rich API documents with code examples, while Zhu et al. (2014) provide an approach to mine API usage examples from test code. Previous studies also assess how developers search for code, particularly, analyzing usage logs (*e.g.,* Bajracharya and Lopes, 2009, 2012; Sadowski et al., 2015). For instance, Sadowski et al. (2015) present how the developers search for code through a case study at Google, while Bajracharya and Lopes (2009, 2012) provide a topic modeling analysis of usage logs of the Koders search engine. Among other findings, they detect that API queries are commonly performed by developers.

Regarding code search on the web, Rahman et al. (2018) assessed the search queries of 310 developers and built a model to detect code and non-code related queries. Xia et al. (2017) analyzed the search queries from 60 developers to understand what developers search for on the web. Bansal et al. (2019) proposed a classifier for distinguishing software engineering-related search queries from other queries and defined the taxonomy of query intents. While these studies focus on the search queries, our research focuses on the search results.

Recently, some studies also pointed out that code examples mined from the web should be used with caution. For example, even though Stack Overflow is known to have a great number of code examples that are reviewed by the community (Parnin et al., 2012), they may suffer from API misuse (Zhang et al., 2018), outdated code and license violations (Ragkhitwetsagul et al., 2019), and security issues (Fischer et al., 2017; Meng et al., 2018; Ragkhitwetsagul et al., 2019).

Overall, the literature agrees on two points: code examples are typically adopted to better understand APIs (Buse and Weimer, 2012; Parnin et al., 2012; Sadowski et al., 2015) and developers often rely on Google to detect code examples (Gu et al., 2016; Hora, 2021b; Kim et al., 2010; Niu et al., 2017; Raghothaman et al., 2016; Sim et al., 2011, 2013; Stolee et al., 2014). We contribute to this research line by studying API code examples and assessing how they are ranked by Google.

8. Conclusion

Code examples are often provided by programming websites to support software development. However, due to many factors found in webpages of programming websites (*e.g.,* code explanations), code examples can be overshadowed by search engines. Thus, it is important to understand how code examples would be

[27] *e.g.,* https://blog.google/products/search/search-language-understanding-bert.

ranked in isolation, *i.e.,* without any other page elements. In this case, we could query and assess the characteristics of top/bottom ranked code examples and verify their quality aspects. In this study, we assessed how the Google search engine ranks code examples in isolation. For this purpose, we designed and built a controlled environment to query and assess code examples with distinct characteristics, by focusing on code examples that describe API usage.

Our empirical analysis showed that: (i) webpages with multiple code examples are more likely to be top ranked by Google than webpages with single examples: 82% of the webpages with multiple code examples are top ranked; (ii) overall, top ranked code examples by Google are larger, have more API references, and are more similar to the input query, while readable and reusable code examples are not necessarily top ranked; and (iii) top ranked code examples can be predicted with a good level of confidence: in the best scenario, precision is 79% is recall is 70%, and AUC is 89%; however, the most important metrics are generic measures and not associated with code quality. Finally, based on our results, we provided insights for practitioners and code search researchers. For instance, we proposed guidelines that can be applied by practitioners to improve programming websites. We also provided insights for researchers on the proposal of novel metrics that are relevant to search engines and software engineering as well as on the reuse of online code examples.

As future work, we plan to extend this research to cover more metrics (*e.g.,* security) and code examples written in other popular programming languages (*e.g.,* Python and JavaScript). We also plan to explore other queries formats in our evaluation, that is, instead of using "<class-name> <method-name>", we can also use variations as "<class-name>" and "<method-name>" or many other suggested by the literature (Hora, 2021b). Another interesting research direction is to assess other internal factors such as code explanations and HTML file names provided by programming websites and verify how they affect the ranking. We should keep in mind that external factors were not investigated in this study, thus, a combination of those factors with internal ones can also be employed in a future research direction. Finally, we plan to replicate this study to other web search engines, such as Yahoo!, Bing, and Baidu.

CRediT authorship contribution statement

Andre Hora: Conceptualization, Methodology, Data curation, Investigation, Software, Validation, Visualization, Writing - original draft, Writing - review and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by CAPES and CNPq.

References

2021. Baeldung. https://www.baeldung.com.
Bajracharya, S., Lopes, C., 2009. Mining search topics from a code search engine usage log. In: International Working Conference on Mining Software Repositories. pp. 111–120.
Bajracharya, S.K., Lopes, C.V., 2012. Analyzing and mining a code search engine usage log. Empir. Softw. Eng. 17 (4–5), 424–466.
Bajracharya, S., Ngo, T., Linstead, E., Dou, Y., Rigor, P., Baldi, P., Lopes, C., 2006. Sourcerer: a search engine for open source code supporting structure-based search. In: Symposium on Object-Oriented Programming Systems, Languages, and Applications. pp. 681–682.
Bansal, C., Zimmermann, T., Awadallah, A.H., Nagappan, N., 2019. The usage of web search for software engineering. arXiv preprint arXiv:1912.09519.
Brandt, J., Dontcheva, M., Weskamp, M., Klemmer, S.R., 2010. Example-centric programming: Integrating web search into the development environment. In: Conference on Human Factors in Computing Systems. pp. 513–522.
Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M., Klemmer, S.R., 2009. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In: Conference on Human Factors in Computing Systems. pp. 1589–1598.
Buse, R.P., Weimer, W.R., 2009. Learning a metric for code readability. IEEE Trans. Softw. Eng. 36 (4), 546–558.
Buse, R.P., Weimer, W., 2012. Synthesizing API usage examples. In: International Conference on Software Engineering. pp. 782–792.
Chen, Q., Zhou, M., 2018. A neural framework for retrieval and summarization of source code. In: International Conference on Automated Software Engineering. pp. 826–831.
2021. Codase. http://www.codase.com.
2021. Codota. https://www.codota.com.
Couto, C., Pires, P., Valente, M.T., Bigonha, R., Anquetil, N., 2014. Predicting software defects with causality tests. J. Syst. Softw. 93, 24–41.
2021. Creating a programmable search engine. https://developers.google.com/custom-search/docs/tutorial/creatingcse.
Cutrell, E., Guan, Z., 2007. What are you looking for?: an eye-tracking study of information usage in web search. In: Conference on Human Factors in Computing Systems. pp. 407–416.
Diakopoulos, N., 2014. Algorithmic accountability reporting: On the investigation of black boxes.
Diakopoulos, N., Trielli, D., Stark, J., Mussenden, S., 2018. I vote for? How search informs our choice of candidate. In: Moore, M., Tambini, D. (Eds.), Digital Dominance: The Power of Google, Amazon, Facebook, and Apple. 22.
Dias, M., Bacchelli, A., Gousios, G., Cassou, D., Ducasse, S., 2015. Untangling fine-grained code changes. In: International Conference on Software Analysis, Evolution and Reengineering. pp. 341–350.
Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., Fahl, S., 2017. Stack overflow considered harmful? the impact of copy&paste on android application security. In: Symposium on Security and Privacy. pp. 121–136.
Furnell, S., Evans, M.P., 2007. Analysing google rankings through search engine optimization data. Internet Res..
Google Search API, 2021. https://www.geeksforgeeks.org.
2021. Google code search google blog shutting down code search. https://googleblog.blogspot.com/2011/10/fall-sweep.html.
2021. Google general guidelines. https://static.googleusercontent.com/media/www.google.com/en//insidesearch/howsearchworks/assets/searchqualityevaluatorguidelines.pdf.
2021. Google search api. https://developers.google.com/custom-search/v1/overview.
Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., Cumby, C., 2010. Exemplar: Executable examples archive. In: International Conference on Software Engineering. pp. 259–262.
Grechanik, M., Fu, C., Xie, Q., McMillan, C., Poshyvanyk, D., Cumby, C., 2010. A search engine for finding highly relevant applications. In: International Conference on Software Engineering. pp. 475–484.
Gu, X., Zhang, H., Kim, S., 2018. Deep code search. In: International Conference on Software Engineering. pp. 933–944.
Gu, X., Zhang, H., Zhang, D., Kim, S., 2016. Deep API learning. In: International Symposium on Foundations of Software Engineering. pp. 631–642.
Hannak, A., Sapiezynski, P., Molavi Kakhki, A., Krishnamurthy, B., Lazer, D., Mislove, A., Wilson, C., 2013. Measuring personalization of web search. In: International Conference on World Wide Web. pp. 527–538.
Holmes, R., Cottrell, R., Walker, R.J., Denzinger, J., 2009. The end-to-end use of source code examples: An exploratory study. In: International Conference on Software Maintenance. pp. 555–558.
Holmes, R., Walker, R.J., 2012. Systematizing pragmatic software reuse. ACM Trans. Softw. Eng. Methodol. 21 (4), 20.
Hora, A., 2021a. Apisonar: Mining API usage examples. Softw. - Pract. Exp. 51 (2), 319–352, http://apisonar.com.
Hora, A., 2021b. Googling for software development: What developers search for and what they find. In: International Conference on Mining Software Repositories. pp. 1–12.
Hora, A., Valente, M.T., 2015. Apiwave: Keeping track of API popularity and migration. In: International Conference on Software Maintenance and Evolution. pp. 321–323.
2021. How search algorithms work. https://www.google.com/search/howsearchworks/algorithms.
Hu, D., Jiang, S., E Robertson, R., Wilson, C., 2019. Auditing the partisanship of google search snippets. In: The World Wide Web Conference. pp. 693–704.
Hu, X., Li, G., Xia, X., Lo, D., Jin, Z., 2018. Deep code comment generation. In: International Conference on Program Comprehension. pp. 200–210.
2021. Introduction to indexing. https://developers.google.com/search/docs/guides/intro-indexing.

Jiarpakdee, J., Tantithamthavorn, C., Hassan, A.E., 2019. The impact of correlated metrics on the interpretation of defect models. IEEE Trans. Softw. Eng..

Kaisser, M., Hearst, M.A., Lowe, J.B., 2008. Improving search results quality by customizing summary lengths. In: Annual Meeting of the Association for Computational Linguistics. pp. 701–709.

Keivanloo, I., Rilling, J., Zou, Y., 2014. Spotting working code examples. In: International Conference on Software Engineering. pp. 664–675.

Kim, J., Lee, S., Hwang, S.-w., Kim, S., 2010. Towards an intelligent code search engine. In: Conference on Artificial Intelligence.

Kim, J., Lee, S., Hwang, S.-W., Kim, S., 2013. Enriching documents with examples: A corpus mining approach. Trans. Inf. Syst. 31 (1), 1.

Kim, S., Whitehead Jr, E.J., Zhang, Y., 2008. Classifying software changes: Clean or buggy?. IEEE Trans. Softw. Eng. 34 (2).

Kliman-Silver, C., Hannak, A., Lazer, D., Wilson, C., Mislove, A., 2015. Location, location, location: The impact of geolocation on web search personalization. In: Internet Measurement Conference. pp. 121–127.

2021. Krugle. http://opensearch.krugle.org.

Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. Trans. Softw. Eng. 34 (4).

Lethbridge, T.C., Singer, J., Forward, A., 2003. How software engineers use documentation: The state of the practice. IEEE Softw. (6), 35–39.

Liaw, A., Wiener, M., et al., 2002. Classification and regression by randomforest. R News 2 (3), 18–22.

Lima, C., Hora, A., 2020. What are the characteristics of popular apis? A large scale study on java, android, and 165 libraries. Softw. Qual. J. 28, 425–458.

Lv, F., Zhang, H., Lou, J.-g., Wang, S., Zhang, D., Zhao, J., 2015. Codehow: Effective code search based on API understanding and extended boolean model (e). In: International Conference on Automated Software Engineering (ASE). pp. 260–270.

Manning, C.D., Raghavan, P., Schütze, H., 2008. Scoring, term weighting and the vector space model. Introd. Inf. Retr. 100, 2–4.

Martin, R.C., 2009. Clean Code: A Handbook of Agile Software Craftsmanship. Pearson Education.

McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C., Xie, Q., 2012. Exemplar: A source code search engine for finding highly relevant applications. Trans. Softw. Eng. 38 (5), 1069–1087.

Menezes, G., Cafeo, B., Hora, A., 2019. Framework code samples: How are they maintained and used by developers?. In: International Symposium on Empirical Software Engineering and Measurement. pp. 1–11.

Meng, N., Nagy, S., Yao, D., Zhuang, W., Argoty, G.A., 2018. Secure coding practices in java: Challenges and vulnerabilities. In: International Conference on Software Engineering. pp. 372–383.

Montandon, J.E., Borges, H., Felix, D., Valente, M.T., 2013. Documenting apis with examples: Lessons learned with the apiminer platform. In: Working Conference on Reverse Engineering. pp. 401–408.

Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A., 2015. How can i use this method?. In: International Conference on Software Engineering. pp. 880–890.

Nasehi, S.M., Sillito, J., Maurer, F., Burns, C., 2012. What makes a good code example?: A study of programming q&a in stackoverflow. In: International Conference on Software Maintenance. pp. 25–34.

Niu, H., Keivanloo, I., Zou, Y., 2017. Learning to rank code examples for code search engines. Empir. Softw. Eng. 22 (1), 259–291.

2021. Openhub. https://code.openhub.net.

Parnin, C., Treude, C., Grammel, L., Storey, M.-A., 2012. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. Tech. Rep., Georgia Institute of Technology.

Peters, F., Menzies, T., Marcus, A., 2013. Better cross company defect prediction. In: Working Conference on Mining Software Repositories.

Philip, K., Umarji, M., Agarwala, M., Sim, S.E., Gallardo-Valencia, R., Lopes, C.V., Ratanotayanon, S., 2012. Software reuse through methodical component reuse and amethodical snippet remixing. In: Conference on Computer Supported Cooperative Work. pp. 1361–1370.

2021. Programcreek. https://www.programcreek.com.

Raghothaman, M., Wei, Y., Hamadi, Y., 2016. Swim: Synthesizing what i mean-code search and idiomatic snippet synthesis. In: International Conference on Software Engineering. pp. 357–367.

Ragkhitwetsagul, C., Krinke, J., Paixao, M., Bianco, G., Oliveto, R., 2019. Toxic code snippets on stack overflow. IEEE Trans. Softw. Eng..

Rahman, M.M., Barson, J., Paul, S., Kayani, J., Lois, F.A., Quezada, S.F., Parnin, C., Stolee, K.T., Ray, B., 2018. Evaluating how developers use general-purpose web-search for code retrieval. In: International Conference on Mining Software Repositories. pp. 465–475.

Retherford, R.D., Choe, M.K., 2011. Statistical Models for Causal Analysis. John Wiley & Sons.

Robillard, M.P., Deline, R., 2011. A field study of API learning obstacles. Empir. Softw. Eng. 16 (6), 703–732.

Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J., 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen?sd for evaluating group differences on the NSSE and other surveys. In: Annual Meeting of the Florida Association of Institutional Research.

Sadowski, C., Stolee, K.T., Elbaum, S., 2015. How developers search for code: a case study. In: European Software Engineering Conference and the Symposium on the Foundations of Software Engineering. pp. 191–201.

Salton, G., Wong, A., Yang, C.-S., 1975. A vector space model for automatic indexing. Commun. ACM 18 (11), 613–620.

Sandvig, C., Hamilton, K., Karahalios, K., Langbort, C., 2014. Auditing algorithms: Research methods for detecting discrimination on internet platforms. Data Discrim.: Convert. Crit. Concerns Prod. Inq. 22.

Scalabrino, S., Bavota, G., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D., Oliveto, R., 2017. Automatically assessing code understandability: How far are we?. In: International Conference on Automated Software Engineering. pp. 417–427.

Scalabrino, S., Linares-Vásquez, M., Oliveto, R., Poshyvanyk, D., 2018. A comprehensive model for code readability. J. Softw.: Evol. Process 30 (6), e1958.

Scalabrino, S., Linares-Vasquez, M., Poshyvanyk, D., Oliveto, R., 2016. Improving code readability models with textual features. In: International Conference on Program Comprehension. pp. 1–10.

2021. Search engine market share worldwide. https://gs.statcounter.com/search-engine-market-share.

2021. Search engine optimization (seo) starter guide. https://support.google.com/webmasters/answer/7451184?hl=en.

2021. Searchcode. https://searchcode.com.

Sidorov, G., Gelbukh, A., Gómez-Adorno, H., Pinto, D., 2014. Soft similarity and soft cosine measure: Similarity of features in vector space model. Comput. Sist. 18 (3), 491–504.

Sim, S.E., Agarwala, M., Umarji, M., 2013. A controlled experiment on the process used by developers during internet-scale code search. In: Finding Source Code on the Web for Remix and Reuse. Springer, pp. 53–77.

Sim, S.E., Umarji, M., Ratanotayanon, S., Lopes, C.V., 2011. How well do search engines support code retrieval on the web?. ACM Trans. Softw. Eng. Methodol. 21 (1), 4.

2021. Spring code guides. https://spring.io/guides.

2021. Stackoverflow. https://stackoverflow.com/.

Stolee, K.T., Elbaum, S., Dobos, D., 2014. Solving the search for source code. ACM Trans. Softw. Eng. Methodol. 23 (3), 26.

Stylos, J., Myers, B.A., 2006. Mica: A web-search tool for finding API components and examples. In: Visual Languages and Human-Centric Computing. pp. 195–202.

Thung, F., Lo, D., Jiang, L., 2012. Automatic defect categorization. In: Working Conference on Reverse Engineering.

Tian, Y., Nagappan, M., Lo, D., Hassan, A.E., 2014. What are the characteristics of high-rated apps? A case study on free android applications. In: International Conference on Software Maintenance and Evolution.

Treude, C., Aniche, M., 2018. Where does google find API documentation?. In: International Workshop on API Usage and Evolution. pp. 19–22.

2021. Tutorialspoint. https://www.tutorialspoint.com.

Vincent, D., 2018. Code example guidelines. URL https://developer.mozilla.org/en-US/docs/MDN/Contribute/Guidelines/Code_guidelines.

2021. W3schools. https://www.w3schools.com.

Xia, X., Bao, L., Lo, D., Kochhar, P.S., Hassan, A.E., Xing, Z., 2017. What do developers search for on the web?. Empir. Softw. Eng. 22 (6), 3149–3185.

Yang, D., Martins, P., Saini, V., Lopes, C., 2017. Stack overflow in github: any snippets there?. In: International Conference on Mining Software Repositories. pp. 280–290.

Yao, Z., Peddamail, J.R., Sun, H., 2019. Coacor: Code annotation for code retrieval with reinforcement learning. In: The World Wide Web Conference. pp. 2203–2214.

Ye, X., Bunescu, R., Liu, C., 2014. Learning to rank relevant files for bug reports using domain knowledge. In: International Symposium on Foundations of Software Engineering. pp. 689–699.

Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., Kim, M., 2018. Are code examples on an online q&a forum reliable?: a study of api misuse on stack overflow. In: International Conference on Software Engineering. pp. 886–896.

Zhou, J., Walker, R.J., 2016. API Deprecation: a retrospective analysis and detection method for code examples on the web. In: International Symposium on Foundations of Software Engineering. pp. 266–277.

Zhu, Z., Zou, Y., Xie, B., Jin, Y., Lin, Z., Zhang, L., 2014. Mining API usage examples from test code. In: International Conference on Software Maintenance and Evolution. pp. 301–310.

Andre Hora is an Assistant Professor in the Computer Science Department at the Federal University of Minas Gerais (UFMG), Brazil. His research interests include software evolution, software repository mining, and empirical software engineering. He earned his Ph.D. in Computer Science from the University of Lille, France. Webpage: www.dcc.ufmg.br/~andrehora.