Framework Code Samples: How Are They Maintained and Used by Developers?

Gabriel Menezes*, Bruno Cafeo*, Andre Hora†
* Faculty of Computing, UFMS, Brazil
menezes@aluno.ufms.br, cafeo@facom.ufms.br
† Department of Computer Science, UFMG, Brazil
andrehora@dcc.ufmg.br

Abstract*—*Background: **Modern software systems are commonly built on the top of frameworks. To accelerate the learning process of features provided by frameworks, code samples are made available to assist developers. However, we know little about how code samples are actually developed.** ***Aims:*** **In this paper, we aim to fill this gap by assessing the characteristics of framework code samples. We provide insights on how code samples are maintained and used by developers.** ***Method:*** **We analyze 233 code samples of Android and SpringBoot, and assess aspects related to their source code, evolution, popularity, and client usage.** ***Results:*** **We find that most code samples are small and simple, provide a working environment to the clients, and rely on automated build tools. They change frequently over time, for example, to adapt to new framework versions. We also detect that clients commonly fork the code samples, however, they rarely modify them.** ***Conclusions:*** **We provide a set of lessons learned and implications to creators and clients of code samples to improve maintenance and usage activities.**

I. Introduction

Modern software systems are commonly implemented with the support of frameworks, which provide feature reuse, improve productivity, and decrease costs [1]–[3]. Frameworks support the development of mobile apps, web platforms, responsive interfaces, cross-platform systems, among other. In the Java ecosystem, for example, there are more than 270,000 packages available to be used by client systems in the Maven repository.[1] In the JavaScript ecosystem the numbers are even higher: the npm repository has over 400,000 packages and reports 6 billions downloads in a single month.[2]

To facilitate and accelerate the learning process of features provided by frameworks, code samples are commonly made available to assist development efforts [4]. Code samples are often provided by world-wide software projects and organizations, such as Android,[3] Spring,[4] Google Maps,[5] Twitter,[6] Microsoft,[7] to name a few. Framework code samples may introduce the usage of basic features, as well as more advanced ones. For instance, a basic sample provided by the SpringBoot framework help newcomer developers on building RESTful web services.[8] In contrast, a more advanced code sample made available by the same framework help developers on securing web applications.[9] Due to their practicality, client developers may copy and paste code samples into their own code base, and may put them into production [4]. Thus, ideally, code samples should follow some good development practices, such as be simple, small, self-contained, easy to understand, secure, and efficient [4].

Although framework code samples are commonly available to help developers, we know little about how they are actually maintained and used by developers. In this context, some questions are still opened, such as: what is the common size of code samples? how do code samples evolve over time? what makes a code sample more popular than other? how are the code samples used by the developers? By answering these questions, we can assess common aspects of code samples, better supporting their maintenance and usage activities.

In this paper, we aim to fill this gap by assessing the characteristics of framework code samples. Specifically, we analyze 233 code samples provided by two widely popular frameworks: Android and SpringBoot. We answer four research questions related to their maintenance and usage. Particularly, we assess aspects related to their source code, evolution, popularity, and client usage:

- *RQ1 (Source Code): What are the source code characteristics of framework code samples?* We find that framework code samples are overall simple and small. We also detect that code samples rely on automated build tools and provide working environments to facilitate the task of running them.
- *RQ2 (Evolution): How do framework code samples evolve over time?* We detect that code samples are not static, but they evolve over time. Updates are often made to keep them up to date with new framework versions, and, consequently, relevant to the clients.
- *RQ3 (Popularity): Which aspects differentiate popular framework samples from ordinary ones?* By comparing

[1] https://search.maven.org/stats
[2] https://www.linux.com/news/event/Nodejs/2016/state-union-npm
[3] https://developer.android.com/samples
[4] https://spring.io/guides
[5] https://developers.google.com/maps/documentation/javascript/examples
[6] http://twitterdev.github.io
[7] https://code.msdn.microsoft.com
[8] https://spring.io/guides/gs/rest-service
[9] https://spring.io/guides/gs/securing-web

978-1-7281-2968-6/19/$31.00 ©2019 IEEE

popular and unpopular code samples, we find that the popular ones are more likely to have a higher amount of source code files. They are also more likely to change over time than the unpopular ones.
- *RQ4 (Client Usage): How are the framework code samples used by developers?* We rely on the fork metric as a proxy of code sample usage. We find that the majority of the forked code samples are inactive. However, a non-negligible ratio of the forked code samples are updated.

Based on our findings, we provide a set of implications to code sample creators and clients, particularly, to support their maintenance and usage activities.

Contributions: This paper has three major contributions:

1) To the best of our knowledge, this is the first research to assess framework code samples, which support the learning process of features provided by frameworks.
2) We provide a large empirical study on the code samples made available by Android and SpringBoot to better understand their maintenance and usage practices.
3) We provide a set of lessons learned and implications to code sample creators and clients.

Structure of the paper: Section II introduces code samples and their importance to support development nowadays. Section III presents the study design, while Section IV reports the results. Section V discusses the implications and Section VI presents the threats to validity. Finally, Section VII discusses related work and Section VIII concludes the paper.

II. Code Samples in a Nutshell

Framework code samples aim to facilitate and accelerate the learning process of features provided by frameworks. In this context, Oracle states that "*code sample is provided for educational purposes or to assist your development or administration efforts*".[10] Spring reports that "*code samples are designed to get you productive as quickly as possible*".[11]

Popular frameworks make code samples available to assist their client developers. The Android framework, for example, has more than one hundred samples on GitHub to help the creation of mobile apps. The SpringBoot framework also has dozens of samples to support the implementation of web apps. In addition to those well-known frameworks, code samples are often provided by organizations to facilitate the usage of their technologies, such as Google Maps APIs, Twitter APIs, Microsoft platforms, Apple platforms, among other.

In order to create good code samples, some guidelines are available. For example, the *Code example guidelines* provided by Mozilla [4] states general practices related to the size, understandability, simplicity, self-containment, security, and efficiency. Guidelines also exist to set up the formatting of code samples, as the one provided by Google.[12] In addition, numerous blogs on programming practices support the developers who are in charge of creating code samples.[13]

Figure 1 presents an official code sample provided by the SpringBoot framework.[14] It supports new developers on building RESTful web services. This sample is composed by only three major classes and helps the clients dealing with important SpringBoot features provided via the annotations: `@RestController`, `@RequestMapping`, `@RequestParam`, and `@SpringBootApplication`. This sample is also composed by other files (*e.g.,* xml, json, shell) to properly help the client running it.

```
public class Greeting {

    private final long id;
    private final String content;

    public Greeting(long id, String content) {
        this.id = id;
        this.content = content;
    }

    public long getId() {
        return id;
    }

    public String getContent() {
        return content;
    }
}
```

```
@RestController
public class GreetingController {

    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping("/greeting")
    public Greeting greeting(@RequestParam(value="name",
defaultValue="World") String name) {
        return new Greeting(counter.incrementAndGet(),
                            String.format(template, name));
    }
}
```

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Fig. 1. Example of code sample (SpringBoot framework).

Although simple and small, the GitHub project[15] hosting this sample has 772 stars and 1,413 forks[16], suggesting that it is indeed relevant and helpful for developers, as presented in Figure 2 (top). Interestingly, this sample is an active project: the 334 commits show that it is evolving over time. By checking its changes, we notice many of them are made to

[10] https://www.oracle.com/technetwork/indexes/samplecode
[11] https://spring.io/guides
[12] https://developers.google.com/style/code-samples
[13] *e.g.,* https://goo.gl/SzV5PL, https://goo.gl/QaA16L, https://goo.gl/ixGaqF
[14] https://spring.io/guides/gs/rest-service
[15] https://github.com/spring-guides/gs-rest-service
[16] A fork is a copy of a repository. It allows developers to change the copy without affecting the original project.

update documentation and configuration files. Changes are also performed to migrate the sample to new framework versions, keeping it up to date and ready to be used with fresh releases of SpringBoot. However, not all code samples receive the same attention from the developers: another official sample provided by SpringBoot to access data with MySQL[17] is much less popular (49 stars), grab less attention from the community (107 forks), and is less active (118 commits), as shown in Figure 2 (bottom).

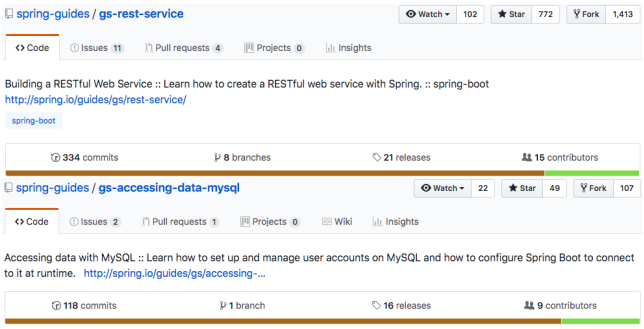

Fig. 2. Code sample statistics (SpringBoot framework).

Overall, we notice the relevance of code samples to support development, as exposed by the variety of software technologies that make them available. We also verify the concerns to create good code samples, as pointed by the many available guidelines. Finally, we notice that code samples can have distinct different levels of popularity, activity, and community engagement.

Despite their importance, to the best of our knowledge, framework code samples are understudied. We are not aware about fundamental aspects on how they are actually maintained and used by developers. By reveling these aspects, we aim to better understand the code samples and provide initial insights on their maintenance and usage practices.

III. Study Design

A. Selecting the Case Studies

In this study, we assess the code samples provided by two world wide frameworks: Android and SpringBoot.

The Android framework[18] allows the creation of Android apps for several devices, such as smartphones, smartwatches, and TVs. Android code samples are publicly available on GitHub[19] and help developers dealing with Android features, such as permissions, picture and video manipulation, background tasks, notifications, networks, multiple touch events, among many other. The SpringBoot framework[20] mostly support the development of web applications. It also provides a set of code samples publicly available on GitHub[21] to help developers creating web apps, such as dealing with RESTful web services, scheduling tasks, uploading files, validating form inputs, caching data, securing apps, among other. Considering both frameworks, in this study we analyze 233 code samples: 176 from Android and 57 from SpringBoot.

We select these two frameworks due to several reasons. *First*, they are relevant and worldwide adopted frameworks that have millions of users. *Second*, they support the creation of two distinct and important niche of apps: mobile and web. *Third*, their base of code samples are publicly available on GitHub, thus, in addition to access their source code, we can also perform evolutionary analysis. *Fourth*, they have a large base of developers, so we can better assess their usage.

Figure 3 presents the distribution of number of files, commits, and stars for the 233 code samples. On the median, the Android code samples have 47 files, 24 commits, and 95 stars. The most popular Android sample is `easypermissions` with 7,328 stars. The SpringBoot code samples have 27 files, 137 commits, and 45 stars. In this case, the most popular sample is `gs-rest-service` with 772 stars.

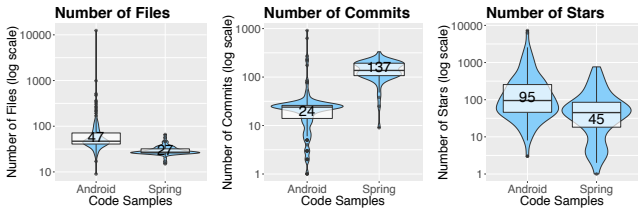

Fig. 3. Basic metrics of the Android and SpringBoot code samples.

B. Source Code Analysis (RQ1)

In Research Question 1, we assess the last version of the source code samples and extract three data: source code metrics, file extensions, and configuration files, as summarized in Figure 4.

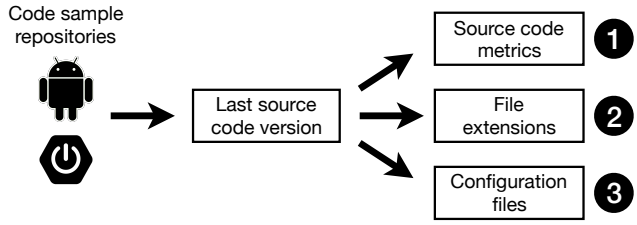

Fig. 4. Source code analysis (RQ1).

1. Source code metrics: We first assess the current state of the samples by computing source code metrics with the support of the software analysis tool Understand.[22] Particularly, we focus on four metrics: number of java files, lines of code, cyclomatic complexity, and commented code lines. Rationale: Small code

[17] https://github.com/spring-guides/gs-accessing-data-mysql
[18] https://developer.android.com
[19] https://github.com/googlesamples
[20] https://spring.io
[21] https://github.com/spring-guides
[22] https://scitools.com

with simple structures may improve code understanding and readability [5]. Code samples are not different; ideally, they should be concise and simple [4]. Code comment is important to any piece of code [6], however, it may be even more relevant to samples as they provide inline comments to help the users.

2. File extensions: We extract the file extensions found in the code samples to better understand their content in addition to source code files. Rationale: In addition to java files, we are not aware of the files that are present in the code samples. The higher presence of other files (*e.g.,* xml, json, jars etc) may indicate that a working environment is available to the clients to run the code samples. In contrast, if the files are mostly concentrated on Java, this may suggest that additional work is still needed by the clients to properly set up the environment.

3. Configuration files: In addition to the file extensions, we also compute the most common configuration files from the code samples. Particularly, we verify whether the code samples adopts automation tools to build, integrate, and manage dependency. Rationale: By relying on these automation tools, the framework code samples are following good development practices, which are commonly adopted on software projects to improve quality and productivity and reduce risks [7]–[9].

C. Evolutionary Analysis (RQ2)

In Research Question 2, we assess all the versions (*i.e.,* commits) of the code samples and extract: evolutionary metrics, file extension changes, configuration file changes, and migration delay, as presented in Figure 5.

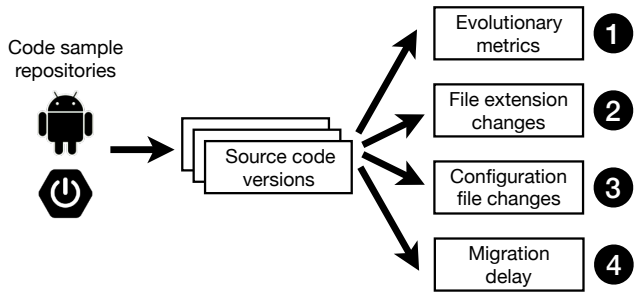

Fig. 5. Evolutionary analysis (RQ2).

1. Evolutionary metrics: We compute metrics to assess the evolution of the code samples. Specifically, we extract two evolutionary metrics: frequency of commits and lifetime. Lifetime is computed as the number of days between the first and the last project commit. Rationale: To cope with API evolution [10]–[13], ideally, the code samples should change over time. Code samples with frequent changes may indicate efforts to keep them up to date. In contrast, less active code samples may suggest they are abandoned.

2. File extension changes: We analyze the file extension changes over time to better understand how the code samples are actually maintained. Rationale: To evolve the code samples, source code and other files should be updated. However, we are not aware of which files are most relevant to keep the samples properly working.

3. Configuration file changes: We analyze the modifications in the configuration files to assess whether the automation tools are being updated. Rationale: In addition to use automation tools to build, integrate, and manage dependencies, it is important to keep them alive, otherwise, the advantages provided by these tools are not achieved.

4. Migration delay: We compute the migration delay between code samples and their frameworks. In other words, we assess how long it takes for code samples migrate to new framework versions. Rationale: As client projects, code samples are dependent of their frameworks. When these frameworks evolve and provide new versions, the code samples (as any other framework client software) should be updated, otherwise, they will be frozen on past versions, and become less attractive to their users [11], [14]–[17].

D. Popularity Analysis (RQ3)

In Research Question 3, we analyze the popularity of the studied code samples to find differences between the most and least popular. Specifically, we sort the code samples in descending order according to their popularity in number of stars. We classify as popular code samples the top 50% with the highest number of stars. Similarly, we classify as unpopular code samples the bottom 50% with the lowest number of stars. We then compare each group regarding the source code and evolutionary metrics described in RQs 1 and 2 (*e.g.,* lines of code, complexity, lifetime, etc), as summarized in Figure 6. We also analyze the statistical significance of the difference between the groups by applying the Mann-Whitney test at *p-value* = 0.05. To show the effect size of the difference between them, we compute Cliff's Delta (or *d*); we use the *effsize* package in R[23] to compute Cliff's Delta. Following previous guidelines [18], 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.

Rationale: Several previous studies have used a similar approach to find differences between popular and unpopular software artifacts, for example, by assessing the popularity of mobile apps [19] and software libraries [11], [20]. Here, we adopt a similar approach to differentiate popular and unpopular code samples, learning with the practices provided by the popular ones.

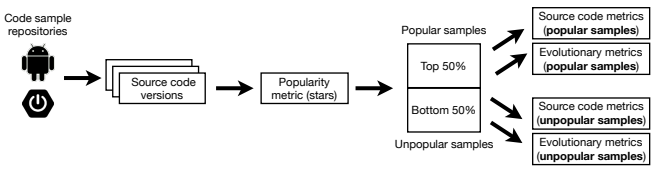

Fig. 6. Popularity analysis (RQ3).

[23]https://cran.r-project.org/web/packages/effsize

E. Client Usage Analysis (RQ4)

In our last Research Question, we focus on the client side, that is, the developers who are adopting the code samples. Particularly, we analyze all GitHub projects that forked the official code samples and compute: fork metrics and file extension changes, as summarized in Figure 7.

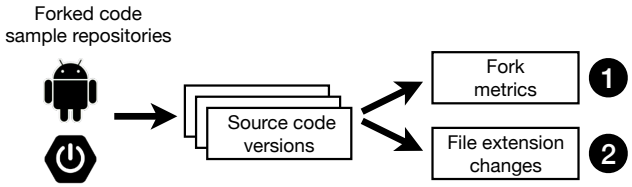

Fig. 7. Client usage analysis (RQ4).

1. Fork metrics: We compute three metrics to assess how the code samples are forked: number of forks, number of forks with commits, and number of commits in forked code samples. Rationale: Fork can be seen as a measure of popularity [21]. After forking, the client developer can update the code or simply do not perform any change. In the case the forked project is updated, this may indicate that the client developer is somehow exploring the code sample, possibly, by running and improving it.

2. File extension changes: We also analyze the file extension changes to better understand how the forked code samples are actually updated. Rationale: To evolve the forked code samples, source code and other files should be updated. However, we are not aware which files are most relevant to be explored by the clients.

IV. Results

A. Source Code (RQ1)

Source code metrics: Figure 8 presents the distribution of the source code metrics in number of java files, lines of code, cyclomatic complexity, and commented code lines in the last version of the code samples. We notice that in terms of java files, the projects are very small: on the median 9 files in the Android samples and only 4 in the SpringBoot samples. The number of lines of code per Java file is larger in Android (70.23) than in SpringBoot (25). However, the Android samples have more comments (32%) per file than the SpringBoot samples (7%). Finally, we see that the complexity is a bit higher in Android than in SpringBoot samples (1.48 vs. 1). These numbers confirm our initial impression that code samples are overall small and simple, as stated by guidelines. However, we also detect that the Android samples are larger and slightly more complex than the SpringBoot ones.

File extensions: Table I presents the file extensions found in the analyzed samples. The Android samples are dominated by xml (15%), followed by java (9.05%) and jar files (3.96%). The SpringBoot samples include mostly Java (12.49%), properties (9.75%), and jar files (8.65%). Interestingly, in addition to the java files, both samples provide a relevant proportion of xml and jar files, indicating that a working environment is also available to the clients.

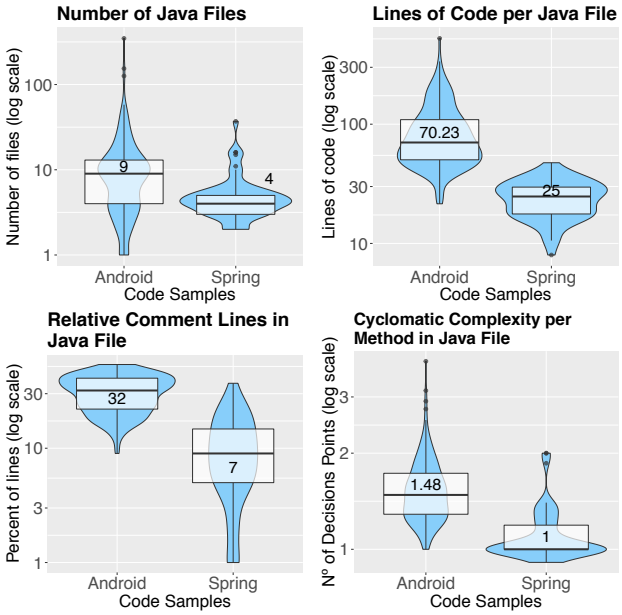

Fig. 8. Source code metrics (RQ1).

TABLE I
File extensions (RQ1).

Android			Spring		
Extensions	#	%	Extensions	#	%
xml	4,307	15.73	java	319	12.49
java	2,477	9.05	properties	249	9.75
jar	1,083	3,96	jar	221	8.65
md	572	2.09	xml	147	5.75
json	549	2,00	adoc	122	4.77
other	17,245	62.98	other	915	35.81

Configuration files: Table II complements the previous analysis by showing specific configuration files. Both projects have `build.gradle` files, which automate software build and delivery via the Gradle Build Tool. In addition, the SpringBoot samples contains `pom.xml` files, which relies on Maven and provide features equivalent to Gradle to automate the build process. The Android samples include the `manifest.xml` files, which are mandatory to Android apps and provide information that a device needs to run the app. Finally, to provide continuous integration via the Travis CI, SpringBoot samples include `travis.yml` files.

Overall, we notice that both samples include configuration files to support their clients as well as adopt automation tools to improve overall quality [7]–[9].

TABLE II
CONFIGURATION FILES (RQ1).

Android			Spring		
Files	#	%	Files	#	%
build.gradle	604	2.21	pom.xml	144	5.64
manifest.xml	397	1.45	build.gradle	118	4.62
travis.yml	2	0.01	travis.yml	56	2.19

Lesson Learned 1: Framework code samples are overall simple and small. We also find that code samples rely on tools to automate build and integration (*e.g.,* Gradle, Maven, and Travis) and provide a working environment to the users (*i.e.,* including jar, xml, properties, and other files in addition to source code).

B. Evolution (RQ2)

Evolutionary metrics: Figure 9 presents the evolutionary metrics extracted from our samples: lifetime and frequency of commits. Differently from the previous analysis, *i.e.,* RQ1, these metrics are computed taking into account the code sample changes over time. We notice that both samples are relatively aged: on the median, the Android samples have 1,474 days (4 years), while the SpringBoot ones are even older, having 1,924 days (5.2 years). Regarding the frequency of commits, the Android samples change one time each 63 days, while the SpringBoot one time each 15 days, on the median.

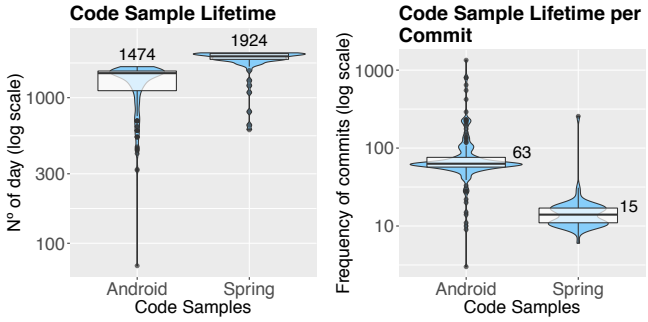

Fig. 9. Evolutionary metrics (RQ2).

File extension changes: Table III presents the changes per file extension. We clearly see that the code samples are not static: they are updated over the years. In both cases, xml files are the most changed, followed by Java, properties, and jar files. Table IV shows another view of this data: the actions performed on the files: addition, modification, or removal. While in Android samples most of the actions are to add files (53.03%), in SpringBoot the majority is to modify existing ones (85.13%). In both cases, removal of files is uncommon.

Configuration file changes: Table V presents the most changed configuration files. We notice that `build.gradle` files are the most changed in both frameworks. In Android code samples, the `manifest.xml` are usually changed, while in SpringBoot the `pom.xml` are often updated. Therefore, as most of these files are related to automation tools, we can confirm that these tools keep being updated over time.

TABLE III
FILE EXTENSION CHANGES (RQ2).

Android			Spring		
Extensions	#	%	Extensions	#	%
xml	9,075	15.67	xml	7,735	28.75
java	7,034	12.14	java	1,437	5.34
properties	1,926	3.33	properties	961	3.57
jar	1,783	3.08	jar	770	2.86
json	1,111	1.92	bat	331	1.23
other	36,988	63.86	other	15,666	58.24

TABLE IV
ACTION TYPE PER FILE (RQ2).

Android			Spring		
File action type	#	%	File action	#	%
Add	30,716	53.03	Modify	22,900	85.13
Modify	23,696	40.91	Add	3,020	11.23
Delete	3,505	6.05	Delete	980	3.64
Total	57,917	100.00	Total	26,900	100.00

TABLE V
CONFIGURATION FILE CHANGES (RQ2).

Android			Spring		
Files	#	%	Files	#	%
build.gradle	5,281	9.12	build.gradle	7,565	28.12
manifest.xml	1,076	1.86	pom.xml	7,531	28.00
travis.yml	24	0.04	travis.yml	208	0.77

Migration delay: Figure 10 presents the delay in number of days the sample take to migrate to new versions of the Android and SpringBoot frameworks. SpringBoot samples migrate much quicker than Android ones. While SpringBoot samples update in same day the new version is available (median zero days), the Android samples take 56 days to migrate, on the median.

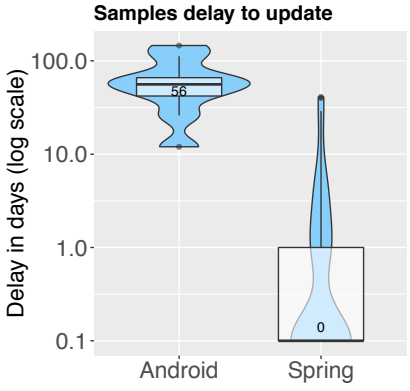

Fig. 10. Migration delay (RQ2)

Figure 11 shows the versions that the code samples are adopting. We see that the Android samples mostly rely on[24] the API level 26 (*i.e.,* Android 8.0, Oreo), 27 (*i.e.,* 8.1, Oreo), and 28 (*i.e.,* 9.0, Pie), however, many samples also rely on other API levels, which represents older versions of Android. Regarding SpringBoot, the majority of the samples are based on version 2.0.5; in this case, we found no sample relying on versions under 2.0, which represents older SpringBoot versions.

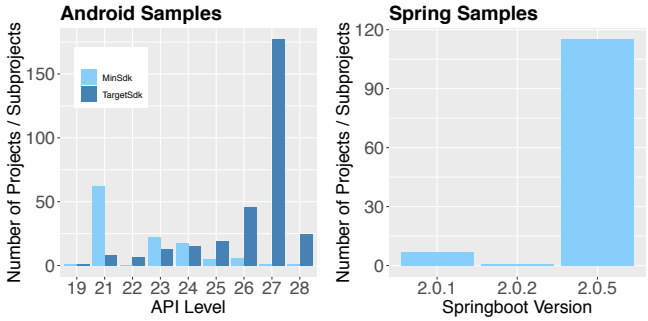

Fig. 11. Code sample versions (RQ2).

To better understand the reason the SpringBoot code samples are migrated faster than the Android ones, we investigated two scenarios. First, we hypothesize that the Android code samples are more complex than SpringBoot ones. Indeed, we have seen in RQ1 that Android code samples are slightly more complex. In addition, Figure 12 (left) presents another view of complexity and shows that the Android code samples rely more on Android APIs than the SpringBoot ones (3.7 vs. 1, on the median). Thus, it is natural that migration takes longer in Android code samples as they are more coupled to the framework. Our second hypothesis is that the developers who maintain the SpringBoot code samples are the same who maintain the SpringBoot framework itself. Figure 12 (right) shows the ratio of developers working on both code samples and framework. We notice that ratio is quite large in SpringBoot: on the median, 75% of the developers who commit code in the samples have also committed in the framework SpringBoot; in Android, this ratio is zero. Therefore, having developers working on both code samples and framework may support their maintenance by decreasing migration delay.

Lesson Learned 2: Code samples are not static, but they evolve over time. Updates are made on both source code and configuration files, for example, to keep them up to date with new framework versions. Overall, code samples are migrated quickly and often rely on recent framework versions. Moreover, having developers working on both code samples and framework may decrease the migration delay.

[24]That is, they have the TargetSdk set to a certain version.

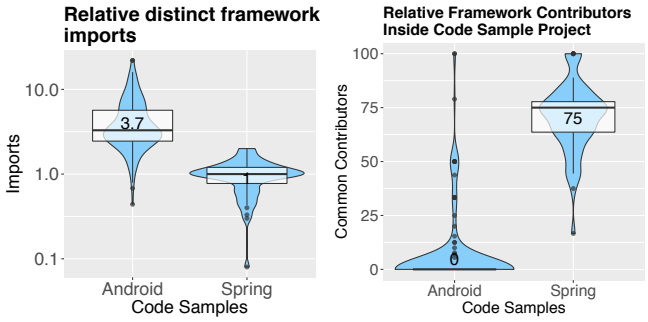

Fig. 12. Dependency to the framework in number of imports (left) and ratio of developers in both code samples and framework (right).

C. Popularity (RQ3)

Table VI presents the results for the popularity analysis. The popular and unpopular Android code samples are statistically significant different regarding the metrics *java files*, *lines of code*, and *cyclomatic complexity*, all with medium effect. The metric *frequency of commits* is also distinct, but with small effect, that is, popular Android samples have statistically significant more changes in shorter periods than the unpopular ones. In SpringBoot, we do not find any difference among the popular and unpopular code samples with respect to the investigated metrics.

TABLE VI
POPULARITY ANALYSIS (RQ3). COMPARISON BETWEEN POPULAR AND UNPOPULAR SAMPLES (POP X UNP). STATISTICALLY SIGNIFICANT DIFFERENCE WITH SMALL (S) OR MEDIUM (M) EFFECT. DIRECTION OF THE DIFFERENCE (DIR)

	Android		Spring	
Metrics	Pop x Unp	Dir	Pop x Unp	Dir
Java files	≤0.001 (M)	↑	0.15	-
Lines of Code	≤0.001 (M)	↑	0.60	-
Relative comment lines	0.57	-	0.42	-
Cyclomatic Complexity	≤0.001 (M)	↑	0.42	-
Lifetime	0.38	-	0.28	-
Frequency of commits	≤0.001 (S)	↓	0.08	-

Lesson Learned 3: Popular Android code samples have a higher amount of code files, are longer and more complex, and have more changes over time.

D. Client Usage (RQ4)

Fork metrics: We adopt the fork metric as a proxy of client usage for the code samples. We detected 25,106 forks of Android code samples and 7,025 of SpringBoot ones. Figure 13 (left) presents the distribution of the number forks per code sample. We see that Android code samples have on the median 47 forks, while the third quartile is 112. In SpringBoot code samples, the median is 71 forks and the third quartile is 137.5. The most forked code sample in Android is `android-testing` (2,409 forks), while in SpringBoot the most forked is `gs-rest-service` (1,412 forks).

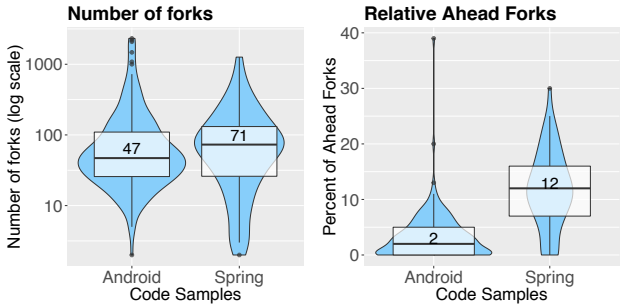

Fig. 13. Code sample forks (RQ4).

The fact that there is a fork do not necessarily mean that it changes over time. Indeed, in Android, only 3% (871 out of 25,106) forked projects are ahead of the base project, *i.e.,* they performed at least one commit; in SpringBoot this ratio is 15% (1,055 out of 7,025). Figure 13 (right) presents the distribution forked code samples with commits. On the median, only 2% of the forked Android code samples have commits; in SpringBoot, this ratio is higher: 12%. Overall, we notice that most of the forked code samples are inactive.

Figure 14 presents the frequency of commits per forked code samples; here, we only show the forks with at least one commit. In this case, 7% and 9% of the forked Android and SpringBoot code samples have 10 or more commits. In both frameworks, the majority of the forked code samples have a single commit (46% and 47%). In Android, 29% of the forked code samples have 2–3 commits, while 16% have 4–10. In SpringBoot, the ratios are equivalent: 26% have 2–3 commits while 16% have 4–10.

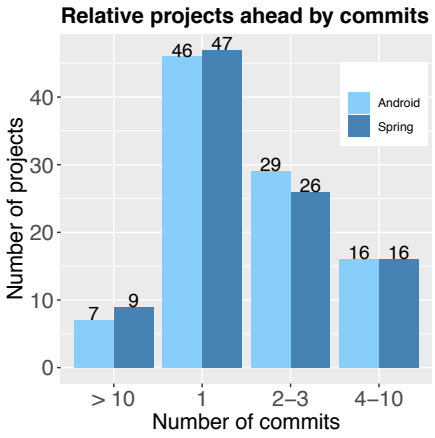

Fig. 14. Commits in forked code samples (RQ4).

File extension changes in forked code samples: Table VII shows the file extension changes in the forked code samples. The notice that the developers change mostly xml, json, java, and jar files. Table VIII shows the actions performed on the files: addition, modification, or removal. While in Android samples most of the actions are to add files (56.97%), in SpringBoot the majority is to modify existing ones (43.59%).

TABLE VII
FILE EXTENSION CHANGES IN FORKED CODE SAMPLES (RQ4).

Android			Spring		
Extensions	#	%	Extensions	#	%
xml	24,022	17.39	java	4,525	34.56
json	8,530	6.17	xml	1,128	8.61
java	8,298	6.01	jar	983	7.51
jar	3,784	2.74	properties	709	5.41
txt	1,264	0.91	yml	398	3.04
other	92,253	66.78	other	5,352	40.87

TABLE VIII
ACTION TYPE PER FILE IN THE FORKED CODE SAMPLES (RQ4).

Android			Spring		
File Action	#	%	File Action	#	%
Add	78,706	56.97	Modify	5,708	43.59
Delete	42,971	31.10	Add	4,640	35.43
Modify	16,474	11.92	Delete	2,747	20.98
Total	138,151	100.00	Total	13,095	100.00

Lesson Learned 4: The majority of the forked code samples are inactive. However, a non-negligible percentage of the forked code samples are updated and evolve over time. The changes are mostly concentrated in xml and java files.

V. IMPLICATIONS

Based on our findings, we provide a set of implications to framework code sample creators and clients in order to support their maintenance and usage practices:

✓Code samples should be simple and small to facilitate their reuse, as stated by good development practices [4]. Indeed, the majority of the code samples provided by Android and SpringBoot follow this rule. However, this is not strict: we find that the code samples with more java files are more likely to be popular in Android.

✓Code samples should provide working environments to ease their usage. Indeed, most of the Android and SpringBoot code samples are formed by source code and many other configuration files necessary to properly run them. Automated build and integration tools may also support both the creators and clients, improving their quality and reducing risks [7]–[9].

✓Code samples are not frozen projects, but they should be updated over time. Changes are commonly performed to follow recent framework versions, otherwise the code samples become out of date and less attractive to the clients [11], [14]–[17]. Indeed, this practice is often performed by Android and SpringBoot code samples, but much faster in the latter. We also find that the code samples that are changed frequently are more likely to be popular in Android.

✓Code samples may benefit from scenarios where their developers also contribute to the framework itself. For example, we found that migration delay may decrease in cases in which the overlap of developers is higher between code samples

and framework. We recognize, however, that this phenomenon should be more explored in further research.

✓The majority of the forked code samples are inactive, however, a non-negligible percentage are updated by their clients as a way to explore and learn them. Thus, we recommend this cycle (fork-change-learn) to the clients kick start in a code sample.

VI. Threats to Validity

This section discusses the study limitations based on the four categories of validity threats described by Wohlin et al. [22]. Each category has a set of possible threats to the validity of an experiment. We identified these possible threats to our study within each category, which are discussed in the following with the measures we took to reduce each risk.

Conclusion validity: It concerns the relationship between the treatment and the outcome. In this work, potential threats arise from *violated assumptions of statistical tests*: the statistical tests used to support our conclusions may have been inappropriately chosen. To mitigate this threat wherever possible, we used statistical tests obeying the characteristics of our data. More specifically, we used non-parametric tests, which do not make any assumption on the underlying data distribution regarding variances and types.

Internal validity: It is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables. One important threat to the internal validity is related to the *ambiguity about the direction of causal influence*: specifically in RQ3, aspects from code samples may be a key to their popularity. On the other hand, the popularity of a code sample may influence code sample aspects measured in our study as code comments and cyclomatic complexity. To ameliorate this threat, we analyze the history of the code samples in order to avoid considering aspects arisen due to the increase in popularity over time.

Construct validity: It refers to the degree to which inferences can legitimately be made from the operationalizations in your study to the theoretical constructs on which those operationalizations were based. We detected a possible threat related to the *restricted generalizability across constructs*: Java might present specific source code characteristics when compared to other programming languages and affects RQ1. This risk cannot be avoided since we analyzed only source code implemented in Java. However, we argue that Java is an important programming language and comprises a large number of code samples in GitHub repository.

External validity: Threats associated with external validity concern the degree to which the findings can be generalised to the wider classes of subjects from which the experimental work has drawn a sample. We identified a risk related to *the interaction between selection and treatment*: the use of code samples provided by two frameworks might present specific aspects when compared to other frameworks. This risk cannot be avoided because our focus are the two frameworks presented in Section III. However, we argue that they are relevant and worldwide adopted frameworks that have millions of end-users. Therefore, we believe the results extracted can be a first step towards the generalization of the results.

VII. Related Work

Frameworks are used to support development, providing source code reuse, improving productivity, and decreasing costs [1]–[3]. Often there is a steep learning curve involved when developers adopt frameworks. Development based on code samples provides the benefits of code reuse, efficient development, and code quality [23]. Moreover, with the popularity and relevance of the Question and Answer (Q&A) sites as Stack Overflow, some studies propose approaches and tools to search and/or retrieve source code samples as well as explore properties of those samples.

Context-based code samples. Software engineering tools are bringing sophisticated search power into the development environment by extending the browsing and searching capabilities [23]–[27]. For instance, Holmes and Murphy [24] proposed a technique that recommends source code examples from a repository by matching structures of given code. XSnippet [27] provides a context-sensitive code assistant framework that provides sample source code snippets for developers. In general, these tools help locate samples of code, demonstrate the use of frameworks and fasten development by exploring the syntactic context provided mainly by the IDE to recommend code samples more relevant to developers (as in Strathcona [24]). However, the samples provided by these systems are highly dependent of a particular development context, whereas code samples typically are complete projects that were made to facilitate and accelerate the learning process of features provided by frameworks. Therefore, it is expected that the types of code samples explored in this paper present different characteristics when compared to samples automatically generated by tools.

Mining API usage examples. Complementing the tools aforementioned, many studies confirmed the the significance of API usage examples, mainly in the context of framework APIs, and proposed approaches to mine API usage examples from open code repositories and search engines [28]–[33]. Most of these work retrieve the so-called code snippets to support API learning, whereas our work focus on complete projects of framework code samples. In addition, our work is not focused on proposing an approach to mine code samples, but analyze characteristics of these code samples.

Assessing Q&A code snippets. Nasehi et al. [34] focused on finding the characteristics of a good example on Stack Overflow. They adopted an approach based on high/low voted answers, number of code blocks used, the conciseness of the code, the presence of links to other resources, the presence of alternate solutions, and code comments. Yang et al. [35] assessed the usability of code snippets across four languages: C#, Java, JavaScript, and Python. The analysis was based on the standard steps of parsing, compiling and running the source code, which indicates the effort that would be required

for developers to use the snippet as-is. Finally, there are studies analyzing the adoption of code snippets [36]–[38]. For instance, Roy and Cordy [36] analyzed code snippet clones in open source systems. They found that on average 15% of the files in the C systems, 46% of the files in the Java systems and 29% of files in the C# systems are associated with exact (block-level) clones. Similar to our work, these studies focus on analyzing properties of code snippets and their adoption on real projects. However, our work targets entire code sample projects instead of code snippets.

VIII. Conclusion

To the best of our knowledge, this is the first research to assess framework code samples. We proposed a large scale empirical study to better understand how these code samples are maintained and used by developers. By assessing 233 code samples provided by the worldwide frameworks Android and SpringBoot, we investigated aspects related to their source code, evolution, popularity, and client usage. We reiterate the most interesting implications to support maintenance and usage of code samples:

- Code samples should be simple and small to facilitate their reuse, as stated by guidelines and followed by the majority of the code samples of Android and SpringBoot.
- Code samples should provide working environments to ease their usage and rely on automated build and integration tool to improve quality.
- Code samples are not static and should evolve over time. Updates are commonly performed to follow recent framework versions, otherwise the code samples become out of date and less relevant to the clients.
- Code samples may benefit from scenarios where their developers also contribute to the framework itself. In this case, migration delay may be decreased.
- Clients of code samples may explore them via the cycle fork-change-learn. Indeed, a strong minority of the client developers do rely on this cycle when using code samples.

As future work, we plan to extend this research by assessing the code samples provided by other frameworks and written in other programming languages (*e.g.,* Google Maps and Twitter APIs). We also plan to analyze other metrics relevant to the samples, such as security and readability. Finally, we plan to perform a survey with the creators and the clients of the code samples to better understand, from their point of view, major code sample limitations and benefits.

Acknowledgements

This research is supported by CAPES and CNPq.

References

[1] S. Moser and O. Nierstrasz, "The effect of object-oriented frameworks on developer productivity," *Computer*, vol. 29, no. 9, 1996.

[2] D. Konstantopoulos, J. Marien, M. Pinkerton, and E. Braude, "Best principles in the design of shared software," in *International Computer Software and Applications Conference*, 2009, pp. 287–292.

[3] S. Raemaekers, A. van Deursen, and J. Visser, "Measuring software library stability through historical version analysis," in *International Conference on Software Maintenance*, 2012, pp. 378–387.

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

[5] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.

[6] T. C. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: The state of the practice," *IEEE Software*, no. 6, pp. 35–39, 2003.

[7] P. Duvall, S. M. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, ser. Addison-Wesley Signature Series. Addison-Wesley, 2007.

[8] M. Meyer, "Continuous integration and its tools," *IEEE Software*, vol. 31, no. 3, pp. 14–16, May 2014.

[9] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and Productivity Outcomes Relating to Continuous Integration in GitHub," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 805–816.

[10] D. Dig and R. Johnson, "How do APIs evolve? A story of refactoring," *Journal of software maintenance and evolution: Research and Practice*, vol. 18, no. 2, pp. 83–107, 2006.

[11] L. Xavier, A. Brito, A. Hora, and M. T. Valente, "Historical and impact analysis of API breaking changes: A large scale study," in *International Conference on Software Analysis, Evolution and Reengineering*, 2017, pp. 138–147.

[12] R. G. Kula, A. Ouni, D. M. German, and K. Inoue, "An empirical study on the impact of refactoring activities on evolving client-used apis," *Information and Software Technology*, vol. 93, pp. 186–199, 2018.

[13] A. Hora, D. Silva, R. Robbes, and M. T. Valente, "Assessing the threat of untracked changes in software evolution," in *40th International Conference on Software Engineering (ICSE)*, 2018, pp. 1102–1113.

[14] T. McDonnell, B. Ray, and M. Kim, "An empirical study of API stability and adoption in the Android ecosystem," in *International Conference on Software Maintenance*, 2013, pp. 70–79.

[15] R. Robbes, M. Lungu, and D. Röthlisberger, "How do developers react to API deprecation? the case of a Smalltalk ecosystem," in *International Symposium on the Foundations of Software Engineering*, 2012.

[16] A. Hora, R. Robbes, M. T. Valente, N. Anquetil, A. Etien, and S. Ducasse, "How do developers react to API evolution? a large-scale empirical study," *Software Quality Journal*, vol. 26, no. 1, pp. 161–191, 2018.

[17] R. G. Kula, D. M. German, A. Ouni, T. Ishio, and K. Inoue, "Do developers update their library dependencies?" *Empirical Software Engineering*, vol. 23, no. 1, pp. 384–417, 2018.

[18] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and cohensd for evaluating group differences on the nsse and other surveys," in *Florida Association of Institutional Research*, 2006, pp. 1–33.

[19] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan, "What are the characteristics of high-rated apps? a case study on free Android applications," in *International Conference on Software Maintenance and Evolution*, 2014, pp. 301–310.

[20] G. Brito, A. Hora, M. T. Valente, and R. Robbes, "On the use of replacement messages in API deprecation: An empirical study," *Journal of Systems and Software*, vol. 137, pp. 306–321, 2018.

[21] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of GitHub repositories," in *International Conference on Software Maintenance and Evolution*, 2016, pp. 334–344.

[22] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[23] R. Sindhgatta, "Using an information retrieval system to retrieve source code samples," in *International Conference on Software Engineering*, 2006, pp. 905–908.

[24] R. Holmes and G. C. Murphy, "Using structural context to recommend source code examples," in *International Conference on Software Engineering*, 2005, pp. 117–125.

[25] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: Helping to navigate the api jungle," in *Conference on Programming Language Design and Implementation*, 2005, pp. 48–61.

[26] D. Poshyvanyk and A. M. and, "Jiriss - an eclipse plug-in for source code exploration," in *International Conference on Program Comprehension*, 2006, pp. 252–255.

[27] N. Sahavechaphan and K. Claypool, "Xsnippet: Mining for sample code," in *Conference on Object-oriented Programming Systems, Languages, and Applications*, 2006, pp. 413–430.
[28] Z. Zhu, Y. Zou, B. Xie, Y. Jin, Z. Lin, and L. Zhang, "Mining api usage examples from test code," in *International Conference on Software Maintenance and Evolution*, 2014, pp. 301–310.
[29] J. E. Montandon, H. Borges, D. Felix, and M. T. Valente, "Documenting apis with examples: Lessons learned with the apiminer platform," in *Working Conference on Reverse Engineering*, 2013, pp. 401–408.
[30] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, and A. Marcus, "How Can I Use this Method?" in *International Conference on Software Engineering*, 2015, pp. 880–890.
[31] R. P. L. Buse and W. Weimer, "Synthesizing api usage examples," in *International Conference on Software Engineering*, 2012, pp. 782–792.
[32] I. Keivanloo, J. Rilling, and Y. Zou, "Spotting working code examples," in *International Conference on Software Engineering*, 2014, pp. 664–675.
[33] H. Niu, I. Keivanloo, and Y. Zou, "Learning to rank code examples for code search engines," *Empirical Software Engineering*, vol. 22, no. 1, pp. 259–291, Feb. 2017.
[34] J. Sillito, F. Maurer, S. M. Nasehi, and C. Burns, "What Makes a Good Code Example?: A Study of Programming Q&A in StackOverflow," in *International Conference on Software Maintenance*, 2012, pp. 25–34.
[35] D. Yang, A. Hussain, and C. V. Lopes, "From Query to Usable Code: An Analysis of Stack Overflow Code Snippets," in *International Conference on Mining Software Repositories*, 2016, pp. 391–402.
[36] C. K. Roy and J. R. Cordy, "Near-miss function clones in open source software: An empirical study," *Journal of Software: Evolution and Process*, vol. 22, no. 3, pp. 165–189, 2010.
[37] L. Heinemann, F. Deissenboeck, M. Gleirscher, B. Hummel, and M. Irlbeck, "On the Extent and Nature of Software Reuse in Open Source Java Projects," in *International Conference on Top Productivity Through Software Reuse*, 2011, pp. 207–222.
[38] D. Yang, P. Martins, V. Saini, and C. Lopes, "Stack Overflow in Github: Any Snippets There?" in *International Conference on Mining Software Repositories*, 2017, pp. 280–290.