How Are Framework Code Samples Maintained and Used by Developers? The Case of Android and Spring Boot

Gabriel Menezes[a], Bruno Cafeo[a], Andre Hora[b]

[a]*Faculty of Computing, UFMS, Brazil*
[b]*Department of Computer Science, UFMG, Brazil*

Abstract

Modern software systems are commonly built on 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 developed and consumed. In this paper, we aim to fill this gap by assessing characteristics of framework code samples. We provide insights into how code samples are maintained and used by developers. We analyze over 230 code samples provided by Android and Spring, and assess aspects related to their code, evolution, popularity, and usage. We find that most code samples are small and simple, provide a working environment, and rely on automated build tools. They frequently change, for example, to adapt to new framework versions. We also detect that clients commonly fork the code samples, however, they rarely modify them. To further understand the problems faced by developers, we analyze 614 Stack Overflow questions about the code samples and 269 issues from code sample repositories. We find that developers face problems when trying to modify the code samples and the most common issue is related to improvement. Finally, we propose implications to creators and code sample clients to improve maintenance and usage activities.

Keywords: Code Samples, Android, Spring Boot, Mining Software Repositories, Software Maintenance, Frameworks

1. Introduction

Modern software systems are commonly built with the support of frameworks, which provide feature reuse, improve productivity, and decrease costs [25, 14, 30]. Frameworks support the development of mobile apps, web platforms, responsive interfaces, cross-platform systems, among others. 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 num-

[1]`https://search.maven.org/stats`

bers are even higher: the npm repository has over 400,000 packages and reports 6 billion 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 [40, 31]. 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 Spring Boot framework helps newcomer developers on building RESTful web services.[8] In contrast, a more advanced code sample made available by the same framework help developers in securing web applications.[9] Due to their practicality, client developers may copy and paste code samples into their own codebase and may put them into production [40]. Thus, ideally, code samples should follow some good development practices, such as be simple, small, self-contained, easy to understand, secure, and efficient [40].

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 others? 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 over 230 code samples provided by two widely popular frameworks: Android and Spring Boot. We answer four research questions related to their maintenance and usage. We then 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 in comparison to conventional projects?* We find that framework code samples are overall simpler and smaller than conventional projects. 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 in comparison to conventional projects?* We detect that code samples are not static, but they evolve like any software system. Updates are often made to keep them up to date with new framework versions and, consequently,

[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`

relevant to the clients.

- *RQ3 (Popularity): Which aspects differentiate popular framework samples from ordinary ones?* By comparing 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 in comparison to conventional projects?* 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 is updated.

In the previous research questions, we focus on better understating, from a quantitative perspective, several aspects of the code samples, including their source code, evolution, and usage. In the second part of the paper, we propose a qualitative study to understand the most common problems and challenges faced by developers when using the code samples. For this purpose, we manually classify 614 Stack Overflow posts about code samples and 269 issues from code sample repositories. We find that developers usually face problems when trying to perform a code sample modification, for example, extending some functionality or performing minor changes. Moreover, we find that developers commonly suggest improvements to code samples via GitHub issues.

Paper extension: This study is an extension of our previous conference paper [21]. We extend this study by proposing three novel analyses: (1) a comparative analysis between code samples and conventional projects to better understand and improve previous findings; (2) a qualitative analysis to better understand why code samples end up in Stack Overflow; and (3) a qualitative study exploring GitHub issues that cause changes on code sample repositories.

Contributions: This paper has three major contributions:

1. We provide an empirical study on the code samples made available by Android and Spring Boot to understand their maintenance and usage practices.
2. We provide a qualitative analysis based on Stack Overflow posts and GitHub issues to reveal the most common problems and needs faced by developers that use code samples.
3. We provide a set of lessons learned and implications to code sample creators and clients.

Structure of the paper: Section 2 introduces code samples and their importance to support development nowadays. Section 3 presents the study design, while Section 4 reports the results of our research questions. Section 5 presents the qualitative study. Section 6 discusses the implications, and Section 7 presents the threats to validity. Finally, Section 8 discusses related work, and Section 9 concludes the paper.

2. 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 code samples on GitHub to help the creation of mobile apps. The Spring Boot also has dozens of code samples to support the implementation of web apps. In addition to those well-known frameworks, organizations often provide code samples to facilitate the usage of their technologies, such as Google Maps APIs, Twitter APIs, Microsoft platforms, and Apple platforms.

To create good code samples, some guidelines are available. For example, the *Code example guidelines* provided by Mozilla [40] states general practices related to size, understandability, simplicity, self-containment, security, and efficiency. Guidelines also exist to set up the formatting of code samples, like 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 example provided by the Spring Boot framework.[14] It supports new developers in building RESTful web services. This code sample is composed by only three major classes and helps the clients dealing with important Spring Boot features provided via the annotations: `@RestController`, `@RequestMapping`, `@RequestParam`, and `@SpringBootApplication`. This sample is also composed of other files (*e.g.,* xml, json, shell) to help the client running it properly.

Although simple and small, the GitHub project[15] hosting this sample has over 950 stars and 1,8K forks, suggesting that it is indeed relevant and helpful for developers. Interestingly, this sample is an active project: the 350 commits show that it evolves over time. By checking its changes, we notice many of them are made to 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 Spring Boot. However, not all code samples receive the same attention from the developers: another official sample provided by Spring Boot to access data with MySQL[16] is much less popular (80 stars), grab less attention from the community (150 forks), and is less active (140 commits).

[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] `https://github.com/spring-guides/gs-accessing-data-mysql`

```java
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;
    }
}
```

```java
@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));
    }
}
```

```java
@SpringBootApplication
public class Application {

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

Figure 1: Example of code sample (Spring Boot framework).

Overall, we notice the relevance of code samples to support development, as exposed by various 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 distinctly different popularity, activity, and community engagement levels.

Despite their importance, framework code samples are understudied. We are not aware of the fundamental aspects of how they are maintained and used by developers. Also, we are not yet aware of common problems and needs faced by their users. By revealing these aspects, we aim to understand the code samples and provide insights into their maintenance and usage practices.

3. Study Design

3.1. Selecting the Case Studies

This study assesses the code samples provided by two largely adopted frameworks: Android and Spring Boot. The Android Framework[17] allows the creation of Android apps for several devices, such as smartphones, smartwatches, and TVs. Android code samples are publicly available on GitHub[18] and help developers dealing with Android features, such as permissions, pictures, and video manipulation, background tasks, notifications, networks, multiple touch events, among many others. The Spring Boot Framework[19] mostly supports the development of web applications. It also provides a set of code samples publicly available on GitHub[20] 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 others. In this study, we analyze 233 code samples: 176 Android and 57 Spring Boot.

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 niches 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 and common problems.

Figure 2 presents the distribution of the 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, while the Spring Boot ones have 27 files, 137 commits, and 45 stars.

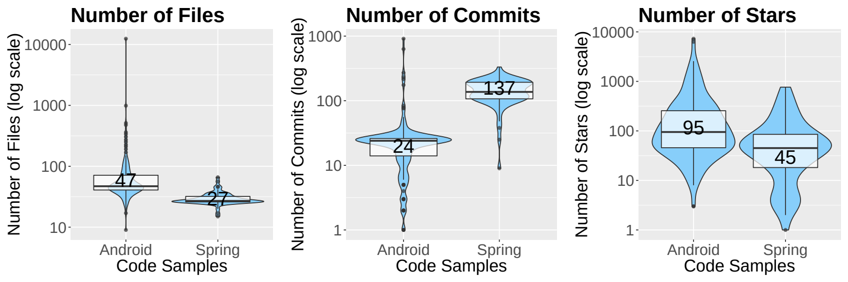


Figure 2: Basic metrics of the Android and Spring Boot code samples.

Our dataset is publicly available at: `https://git.io/J825h`.

[17]`https://developer.android.com/guide/platform`
[18]`https://github.com/googlesamples`
[19]`https://spring.io`
[20]`https://github.com/spring-guides`

3.2. Conventional Projects Comparison

It is important to highlight that, we assess conventional projects to compare them with code samples in some results of the research questions. To do so, we randomly select 233 out of the top 5,000 projects sorted by the number of stars. We select 176 Android and 57 Spring Boot conventional projects to follow the same proportion of code samples. More specifically, our goal is to define a benchmark for comparing some characteristics of the code samples to conclude source code size, evolution, and client usage. For that, we extract the same metrics used in code samples for three out of four research questions (RQ1, RQ2, and RQ4). We run a statistical significance test to compare the results of the extracted metrics between conventional projects and code samples. We apply the Mann-Whitney test at *alpha value* = 0.05, and compute Cliff's Delta to compute the effect size of the difference.

3.3. Source Code Analysis (RQ1)

In Research Question 1, we assess the last version of the source projects (code samples and conventional projects) and extract three data: source code metrics, file extensions, and configuration files, as summarized in Figure 3.

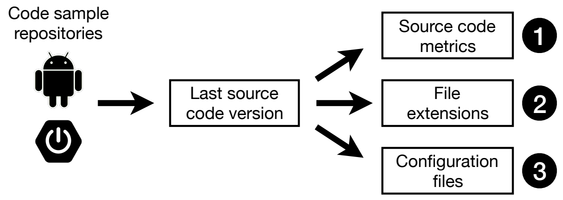


Figure 3: Source code analysis (RQ1).

1. Source code metrics: We first assess the current state of the projects by computing source code metrics with the support of the software analysis tool Understand.[21] We focus on four metrics: number of java files, lines of code, cyclomatic complexity, and commented code lines. Rationale: Small code with simple structures may improve code understanding and readability [18]. Code samples are not different; ideally, they should be concise and simple [40]. This means that code samples need to be simpler than conventional projects. In this way, it is necessary to extract source code metrics for code samples and conventional projects. Moreover, code comment is important to any piece of code [16]. However, it may be even more relevant to samples than to conventional projects, as they provide inline comments to help the users.

2. File extensions: We extract the file extensions found in the projects 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.

[21] `https://scitools.com`

A similar presence of other files (*e.g.,* xml, json, jars, etc.) in code samples may indicate that a working environment is available to the clients to run the code samples as in conventional ones. 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. Besides that, we can observe which file extensions are present under framework influence.

3. Configuration files: In addition to the file extensions, we also compute the most common configuration files from the projects. Particularly, we verify whether the code samples adopt automation tools to build, integrate, and manage dependency Rationale: With a similar result between code samples and conventional projects, we could conclude that the framework code samples are following good development practices when they rely on these automation tools, which are commonly adopted on software projects to improve quality and productivity and reduce risks [8, 22, 39].

3.4. Evolutionary Analysis (RQ2)

In this research question, we assess all the versions (*i.e.,* commits) of the projects and extract: evolutionary metrics, file extension changes, configuration file changes, and migration delay, as presented in Figure 4.

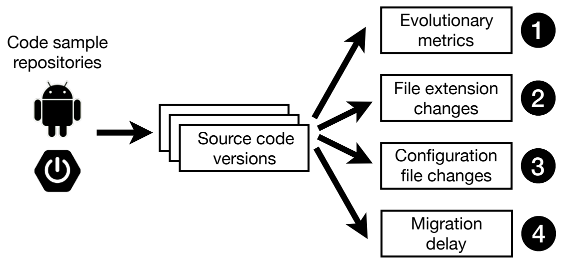


Figure 4: Evolutionary analysis (RQ2).

1. Evolutionary metrics: We compute metrics to assess the evolution of the projects. 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 updates and bug fixes, ideally, the code samples should change over time as conventional projects also do. 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. In this way, results of conventional projects can be used as a threshold to guide these results.

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, organizations should update source code and other files. However, we are not aware of which files are most relevant to keep the samples properly working. To better understand that, we need to know how conventional projects of these frameworks are maintained.

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 projects and their frameworks. In other words, we assess how long it takes for code samples to migrate to new framework versions. Rationale: As client projects, code samples are dependent on their frameworks. When these frameworks evolve and provide new versions, the code samples (as any other framework client project) should be updated. Otherwise, they will be frozen on past versions and become less attractive to their users [12, 15, 19, 42]. In addition, it is important to find out how long conventional projects take to keep up to date, and compare to code samples.

3.5. 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 the 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 5. We also analyze the statistical significance of the difference between the groups 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 (or d); we use the *effsize* package in R[22] to compute Cliff's Delta. Following previous guidelines [32], 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 [37], GitHub projects [2], and libraries [4]. Here, we adopt a similar approach to differentiate popular and unpopular code samples, learning with the practices provided by popular ones.

3.6. Client Usage Analysis (RQ4)

In our last research question, we focus on the client-side, that is, the developers who are using the projects. Particularly, we analyze all GitHub projects

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

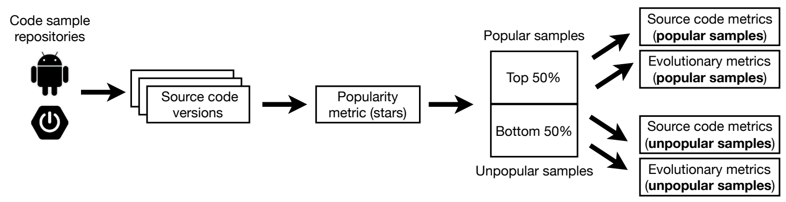

Code sample repositories
Source code versions
Popularity metric (stars)
Popular samples
Top 50%
Bottom 50%
Unpopular samples
Source code metrics (popular samples)
Evolutionary metrics (popular samples)
Source code metrics (unpopular samples)
Evolutionary metrics (unpopular samples)

Figure 5: Popularity analysis (RQ3).

that forked the projects and compute: fork metrics and file extension changes, as summarized in Figure 6.

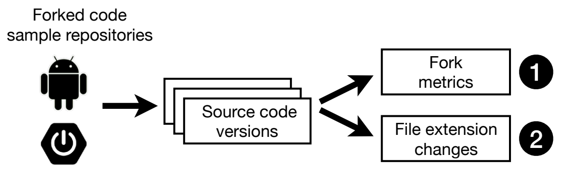


Figure 6: Client usage analysis (RQ4).

1. Fork metrics: We compute three metrics to assess how the projects are forked: number of forks, number of forks with commits, and number of commits in forked projects. Rationale: Fork can be seen as a measure of popularity [2]. After forking, the client developer can update the code or simply do not perform any change. If 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 understand better how the forked code samples are actually updated. Rationale: To evolve the forked code samples, organizations should update source code and other files. However, we are not aware of which files are most relevant to be explored by the clients. Moreover, it is interesting to assess how clients of conventional projects change their own forks and compare them to code samples.

3.7. Assessing Stack Overflow Questions

In this qualitative analysis, we assess Stack Overflow questions to better understand common problems faced by clients of code samples. Rationale: Code samples are created to support developers dealing with framework features. In addition, the literature shows that code samples are important to support learning [31]. Even more, a lack of code samples can be a barrier to understand frameworks and APIs [45]. Nonetheless, some aspects could make difficult code samples understand and use, such as an increase in complexity and size, and a decrease in readability. Therefore, it is important to explore whether code sample clients have problems when they refer to code samples. The details of this process, results, and implications are presented in Section 5.

3.8. Assessing GitHub Issues

We also assess GitHub issues to explore common needs related to code samples. Rationale: In this analysis, we aim to explore how code samples maintainers react when they face developers' issues. For example, do they change the code samples? Are the changes coming from internal comments? Moreover, we aim to explore whether the developers' questions (and other types of interactions) are different from Stack Overflow ones.

4. Empirical Results

4.1. Source Code (RQ1)

Source code metrics: Figure 7 presents the distribution of the source code metrics in terms 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 tiny: on the median, 9 files in the Android samples and only 4 in the Spring Boot samples. The number of lines of code per Java file is larger in Android (70.23) than in Spring Boot (25). However, the Android samples have more comments (32%) per file than the Spring Boot samples (7%). Finally, we see that the complexity is slightly higher in Android than in Spring Boot samples (1.48 vs. 1). We also detect that the Android samples are larger and slightly more complex than the Spring Boot ones.

When we compare code samples to conventional projects, the numbers confirm our initial impression that code samples are overall small and simple. Table 1 presents the comparison between code samples and conventional projects. They are statistically significantly different regarding the number of java files in both Android (**: medium effect) and Spring Boot (***: large effect). In other words, in both frameworks, code samples are smaller than conventional projects in terms of the number of java files (direction: $\downarrow$). Another metric in which both frameworks agree is cyclomatic complexity: code samples have statistically significant less complexity (Android: * small effect, direction: $\downarrow$; Spring Boot: *** large effect, direction: $\downarrow$) than conventional projects.

The other metrics (lines of code per file and relative comment lines) vary according to the framework. For example, Android code samples have more relative comment lines than conventional projects (***: large effect size, direction: $\uparrow$), while Spring Boot samples have fewer relative comment lines than conventional projects (***: large effect size, direction: $\downarrow$). There are fewer lines of code per file in Spring Boot samples (***: large effect size, direction: $\downarrow$) than conventional projects. However, when we analyze the Android framework, there is no statistically significant difference between samples and conventional projects.

File extensions: Table 2 presents the file extensions found in the analyzed samples. The Android samples are dominated by xml (15.73%), followed by java

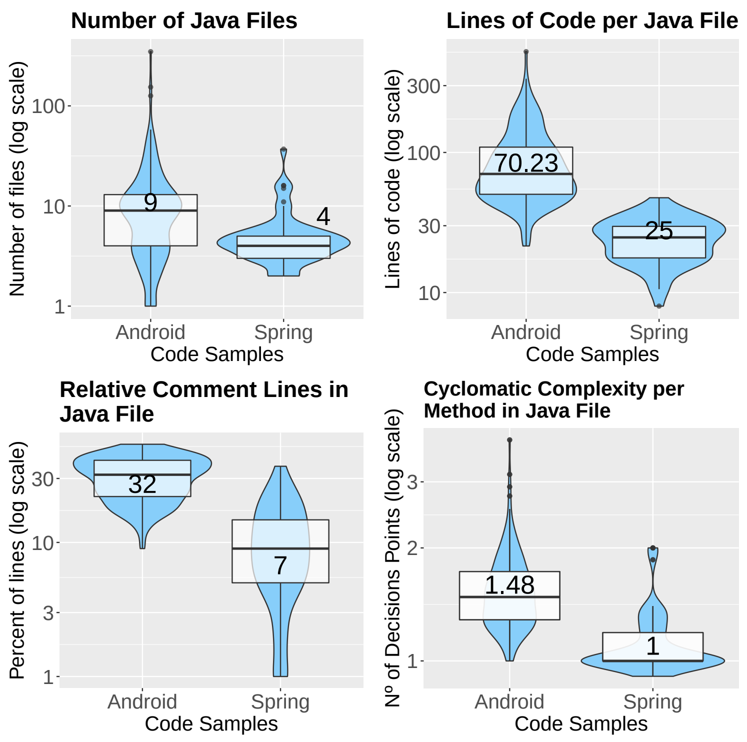


Figure 7: Source code metrics of code samples (RQ1).

(9.05%) and jar files (3.96%). The Spring Boot 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.

Table 2 also shows the file extensions found in the analyzed conventional projects. Android conventional projects are dominated by Java files (40%), followed by XML files (14%), and other extensions representing 41%. In Spring Boot conventional projects, most cases are Java files (56%), followed by XML files (10%), and other extensions (27%).

Comparing Android code samples with Android conventional projects, we can notice that most files in code samples are related to XML files (15.73%). In contrast, in conventional projects, the majority is related to source code files (*i.e.,* java extension) comprising 40.61%. This is an interesting behavior since there are more XML files in Android code samples than source code files. Our analysis points out to the following reasons: (i) Android often generates a considerable amount of XML files, especially to define UI layouts, (ii) code samples are complete projects providing a working environment to the users besides the source code, and (iii) source code should be simple and small in code samples.

Table 1: Comparing Code Sample and Conventional Projects. Statistically significant difference with small (*), medium (**) and large (***) effect. Not statistically significant different (-). Direction of the difference (Dir)

	Android		Spring	
Metrics	Sample x Conventional	Dir	Sample x Conventional	Dir
Java files	**	↓	***	↓
Lines of code per file	-	-	***	↓
Relative comment lines	***	↑	***	↓
Cyclomatic complexity	*	↓	***	↓

Table 2: File extensions of code samples and conventional projects (RQ1).

Android			Spring		
Extensions	#	%	Extensions	#	%
Code Samples					
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	67,17	other	1,379	58.59
Conventional Projects					
java	21,260	40.61	java	14,437	56.73
xml	7,608	14.53	xml	2,621	10.30
properties	687	1,31	properties	700	2.75
jar	427	1.01	md	430	1.69
kt	391	0.75	yml	264	1.04
other	21,983	41.97	other	6,996	27.49

When comparing Spring Boot code samples with Spring Boot conventional projects, in both cases, the majority of files are related to source code files (*i.e.,* java extension). In Spring Boot code samples, we found 12.49% of files related to the java extension, while in Spring Boot conventional projects over 56%. Unlike Android, Spring Boot does not generate the same amount of XML configuration files; nonetheless, XML files present a considerable amount in conventional projects. On the other side, it is important to highlight that, despite java files be the majority in Spring Boot samples, the low percentage, when compared to conventional projects, restate that (i) code samples provide a complete working environment and (ii) code samples are simple and small.

Configuration files: Table 3 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 Spring Boot samples contain `pom.xml` files, which rely 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, Spring Boot samples include `travis.yml` files.

Overall, we notice that both samples include configuration files to support their clients and adopt automation tools to improve overall quality [8, 22, 39]. This data also states that code samples provide a complete working environment for their users.

Table 3: Configuration files of code samples and conventional projects (RQ1).

Android			**Spring**		
Files	#	%	Files	#	%
Code Samples					
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
Conventional Projects					
build.gradle	732	1.40	pom.xml	566	2.22
manifest.xml	573	1.09	build.gradle	91	0.37
pom.xml	26	0.05	travis.yml	31	0.13

Table 3 also presents the numbers of working environment files in conventional Android and Spring Boot projects. In Android conventional projects, we found the same pattern as the one observed in code samples: build.gradle files on top (1.40%), followed by the mandatory manifest.xml (1.09%). When analyzing Spring Boot conventional projects, the top 3 files are the same as Spring Boot code samples. The pom.xml is on top of the most found configuration files (2.22%), followed by build.gradle (0.37%) and travis.yml (0.13%).

Lesson Learned 1: Framework code samples are overall simpler and smaller than conventional projects. 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 code).

4.2. Evolution (RQ2)

Evolutionary metrics: Figure 8 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 considering 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 Spring Boot ones are even older, having 1,924 days (5.2 years). Regarding the frequency of commits, the Android samples change once every 63 days, while the Spring Boot once every 15 days, on the median.

When we compare the analyzed code samples with conventional projects (Table 4), we did not find a statistically significant difference in the lifetime of Android code samples compared to Android conventional projects. However, the lifetime of code samples is slightly higher than the one found in conventional projects. In Spring Boot, we found a statistically significant difference in

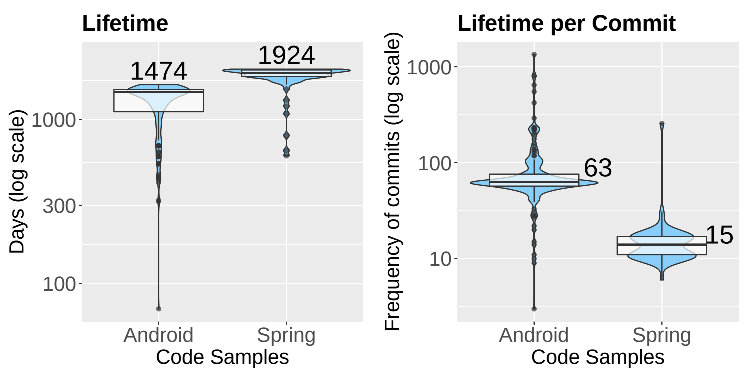


Figure 8: Evolutionary metrics of code samples (RQ2).

lifetime compared to conventional projects (***: large effect size, direction: ↑). In other words, our results show that code samples tend to be longer-lived than conventional projects using the analyzed frameworks. Regarding the comparison of lifetime per commit, there is a statistically significant difference in both Android (***: large effect, direction: ↑) and Spring Boot (**: medium effect, direction: ↑). This means that, despite a considerable evolutionary activity, code samples have a lower frequency of commits when compared to conventional projects. Next, we analyze the types of changes that happen in these commits. More specifically, we analyze changes per extension and changes in configuration files.

Table 4: Comparing Code Sample and Conventional Projects in RQ2 metrics. Statistically significant difference with small (*), medium (**) and large (***) effect. Not statistically significant different (-). Direction of the difference (Dir)

	Android		**Spring**	
Metrics	Sample x Conventional	Dir	Sample x Conventional	Dir
Lifetime	-	↓	***	↑
Lifetime per commit	***	↑	**	↑
Delay to update	***	↓	***	↓

File extension changes: Table 5 presents the changes per file extension both in code samples and conventional projects. We clearly see that the code samples are not static: several files are updated over the years. In both cases, xml files are the most changed, followed by Java, properties, and jar files.

Regarding comparing code samples and conventional projects, it is interesting to notice that Java files and XML files hold the first two places in conventional projects (Table 5). However, the Java file extension is the most changed type both in Android and Spring Boot conventional projects, differently from their code samples where xml extension is the most changed type of file during the evolution of the analyzed projects. In short, code samples tend to

Table 5: File extension changes in code samples and conventional projects (RQ2).

Android			**Spring**		
Extensions	#	%	Extensions	#	%
Code Samples					
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	58.25
Conventional Projects					
java	212,278	56.55	java	147,159	57.88
xml	52,366	13.95	xml	40,984	16.12
md	5,879	1.57	md	4490	1.77
jar	4,294	1.14	properties	3,675	1.45
properties	2,806	0.75	yml	2,007	0.79
other	97,782	26.04	other	55,948	21.99

change more configuration files than source code. This happens mainly because changes in source code are not as frequent as in conventional projects. Most of the changes in code samples only happen to update the source code to a more recent framework version.

Table 6 shows another view of this data: the actions performed on files: addition, modification, or removal. While in Android samples, most of the actions are to add files (53.03%), in Spring Boot samples, the majority is to modify existing ones (85.13%). In both cases, the removal of files is uncommon. When we compare code samples to conventional projects, the behavior is very similar. The majority of changes during the evolution of Spring Boot conventional projects is to modify existing files (Android - 63.52% and Spring Boot - 66.73%). Moreover, the removal of files in both Android and Spring Boot is also uncommon. The only difference is that in conventional Android projects, the action to modify files is more common than the addition of files (63.54% vs. 25.20%). In Android code samples, adding a file is more common than file modification (53.03% vs. 40.91%). This behavior is noticed in Android code samples because, after every update to a more recent framework version, files are automatically generated in the context of the code sample project. As shown later, in conventional Android projects, there is a migration delay to a new framework version, thus making file modifications more common.

Configuration file changes: Table 7 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 Spring Boot, 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.

Most of the code sample behavior previously presented is also observed in conventional projects in configuration file changes (Table 7). We highlight that

Table 6: Action type per file in code samples and conventional projects (RQ2).

Android			Spring		
File action type	#	%	File action	#	%
Code Samples					
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
Conventional Projects					
Modify	238,546	63.54	Modify	169,665	66.73
Add	94,596	25.20	Add	54,479	21.43
Delete	42,263	11.26	Delete	30,119	11.85
Total	375,405	100.00	Total	254,263	100.00

Table 7: Configuration file changes in code samples and conventional projects (RQ2).

Android			Spring		
Files	#	%	Files	#	%
Code Samples					
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
Conventional Projects					
build.gradle	7,257	1.93	pom.xml	24,884	9.79
manifest.xml	3,462	0.92	build.gradle	1,457	0.57
pom.xml	2,561	0.68	travis.yml	280	0.11

`pom.xml` takes the first place from `build.gradle` in Spring Boot conventional projects. The numbers of these files in Spring Boot code samples are very similar in configuration file changes (28.12% vs. 28.00%). Another point to highlight is that the percentage of configuration file changes in code samples is higher than in conventional projects. This happens due to the higher number of total files in conventional projects when compared to code samples.

Migration delay: Figure 9 presents the delay in the number of days the sample takes to migrate to new Android and Spring Boot frameworks versions. Spring Boot samples migrate much quicker than Android ones. While Spring Boot samples update the same day the new version is available (median zero days), the Android samples take 56 days to migrate on the median.

Figure 10 shows the versions that the code samples are adopting. We see that the Android samples mostly rely on[23] 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 represent older versions of Android. Regarding Spring Boot, most of the samples are based on version 2.0.5; in this case, we

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

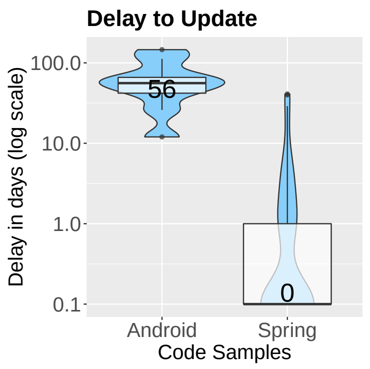


Figure 9: Migration delay in code samples (RQ2)

found no sample relying on versions under 2.0, which represents older Spring Boot versions.

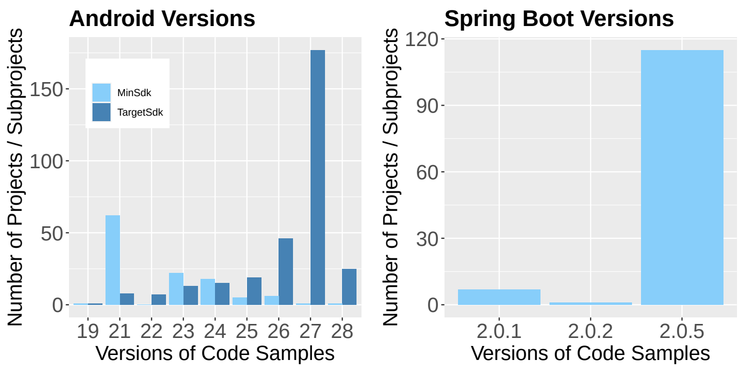


Figure 10: Code sample versions (RQ2).

To better understand why the Spring Boot 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 the Spring Boot ones. Indeed, we have seen in RQ1 that Android code samples are slightly more complex. In addition, Figure 11 (left) presents another view of the complexity and shows that the Android code samples rely more on Android APIs than the Spring Boot ones. 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 Spring Boot code samples are the same who maintain the Spring Boot framework itself. Figure 11 (right) shows the ratio of developers working on both code samples and framework. We notice that the ratio is quite large in Spring Boot: 75% of the developers who commit code in the samples have also committed in the framework Spring Boot; in Android, this ratio is zero. Therefore, having developers working on both code samples and framework may support their maintenance by decreasing migration delay.

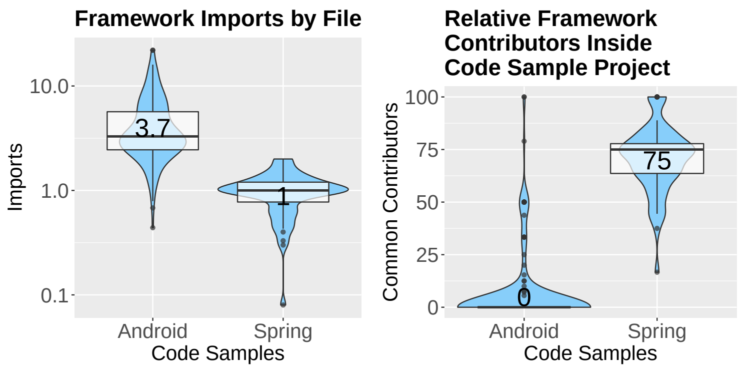


Figure 11: Dependency to the framework in number of imports (left) and ratio of developers in both code samples and framework (right).

Finally, to compare the migration delay to new framework versions, we analyzed the delay to update also in conventional projects. In this analysis, we found a statistically significant difference in the delay to update to new framework versions when comparing code samples to conventional projects (***: large effect, direction: ↓) in both Android and Spring Boot (Table 4). In other words, we show that code samples update faster to new framework versions than conventional projects. We believe this happens mainly because (i) code samples have an educational purpose, and thus it is essential to be updated, and (ii) developers who maintain the framework itself may also maintain its code samples.

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 when comparing with conventional projects and often rely on recent framework versions. Moreover, having developers working on both code samples and frameworks may decrease the migration delay.

4.3. Popularity (RQ3)

Table 8 summarizes 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 a small effect. That is, popular Android samples have statistically significantly more changes in shorter periods than the unpopular ones. In Spring Boot, we do not find any difference between the popular and unpopular code samples concerning the investigated metrics.

Table 8: Popularity analysis (RQ3). Comparison between popular and unpopular samples (Pop x Unp). Statistically significant difference with small (*) or medium (**) effect. Not statistically significant different (-). Direction of the difference (Dir)

	Android		Spring	
Metrics	Pop x Unp	Dir	Pop x Unp	Dir
Java files	**	↑	-	-
Lines of Code	**	↑	-	-
Relative comment lines	-	-	-	-
Cyclomatic Complexity	**	↑	-	-
Lifetime	-	-	-	-
Frequency of commits	*	↓	-	-

Lesson Learned 3: Popular Android code samples have more code files, are bigger, more complex, and change more frequently over time.

4.4. 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 Spring Boot ones. Despite a non-negligible number of forks in code samples, our data presented in Table 9 shows a lower number of forks in code samples when compared to conventional projects both in Android (*: small effect, direction: ↓) and Spring Boot (***: large effect, direction: ↓).

Table 9: Comparing Code Sample and General Projects in RQ4 metrics. Statistically significant difference with small (*), medium (**) and large (***) effect. Not statistically significant different (-). Direction of the difference (Dir)

	Android		Spring	
Metrics	Sample x Conventional	Dir	Sample x Conventional	Dir
Number of forks	*	↓	***	↓
Relative ahead forks	-	-	***	↑

Figure 12 (left) presents the distribution of the number of forks per code sample. We see that Android code samples have on the median 47 forks, while the third quartile is 112. In Spring Boot 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 Spring Boot the most forked is `gs-rest-service` (1,412 forks). The fact that there is a fork does 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 Spring Boot, this ratio is 15% (1,055 out of 7,025). Figure 12 (right) presents the distribution of forked code samples with commits. On the median, only 2% of the forked Android code samples have commits; in Spring Boot, this ratio is higher: 12%. Overall, we notice that most of the forked code samples are inactive. When we compare code samples to conventional projects, there is no statistically significant difference in the context of Android. In contrast, there is a statistically significant difference in Spring Boot (***: large effect, direction: ↑). This means that forks in Spring Boot code samples tend to be more active than forks in Spring Boot conventional projects.

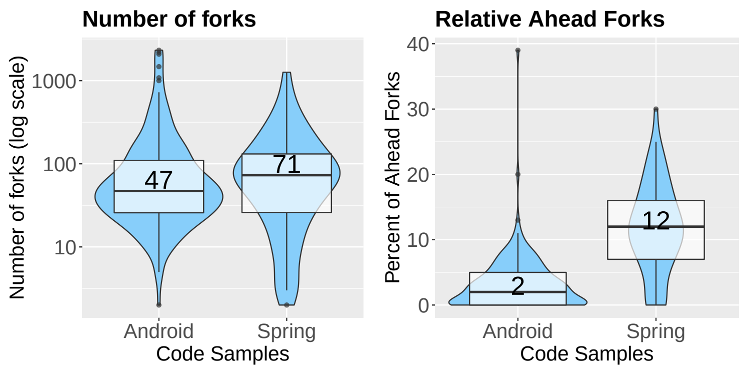


Figure 12: Code sample forks (RQ4).

Figure 13 presents the frequency of commits per forked code sample; we only show the forks with at least one commit. In this case, 7% and 9% of the forked Android and Spring Boot code samples have 10 or more commits. Most 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 Spring Boot, the ratios are equivalent: 26% have 2–3 commits while 16% have 4–10.

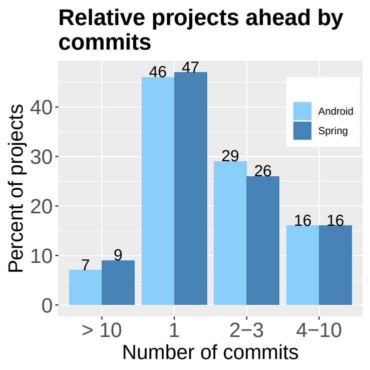


Figure 13: Commits in forked code samples (RQ4).

File extension changes in forked code samples: Table 10 shows the file extension changes in the forked code samples. We notice that developers change mostly xml, json, java, and jar files.

Table 10: 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

Lastly, Table 11 shows the actions performed on files. While in Android samples, most of the actions are to add files (56.97%), in Spring Boot, the majority is to modify existing ones (43.59%).

Table 11: 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.

5. Qualitative Study

In this second study, we assess Stack Overflow questions (Section 5.1) and GitHub issues (Section 5.2) to better understand Android and Spring Boot clients' most common problems when using the code samples. This study provides a complementary view for the first quantitative analysis. To guide this analysis, we propose the following research question: *What are the most common questions and needs about the code samples?*

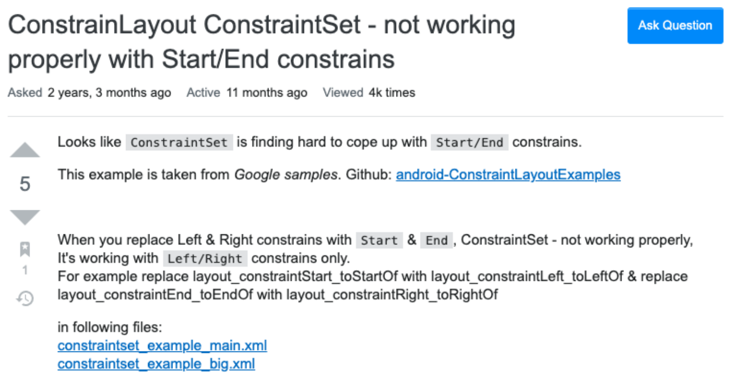
5.1. Stack Overflow Questions

5.1.1. Analysis

Stack Overflow is the *de facto* question & answer platform for software development: it hosts over 20M questions, helping millions of developers to learn and share their knowledge.[24] Questions on Stack Overflow can receive answers,

[24]`https://stackoverflow.com/company`

and the community is responsible for evaluating the quality of the proposed answers, giving positive or negative votes. Figure 14 presents an example of a Stack Overflow question[25] about the Android sample `android-ConstraintLayoutExample`.[26] In this case, the developer performs a simple modification in the sample (*i.e.*, replacing left & right constrains with start & end), and, according to him, the sample is not working as expected. As we can notice, the question has 5 positive votes and 4k views.


ConstrainLayout ConstraintSet - not working properly with Start/End constrains

Asked 2 years, 3 months ago Active 11 months ago Viewed 4k times

Looks like `ConstraintSet` is finding hard to cope up with `Start/End` constrains.

This example is taken from *Google samples*. Github: android-ConstraintLayoutExamples

When you replace Left & Right constrains with `Start` & `End`, ConstraintSet - not working properly, It's working with `Left/Right` constrains only.
For example replace layout_constraintStart_toStartOf with layout_constraintLeft_toLeftOf & replace layout_constraintEnd_toEndOf with layout_constraintRight_toRightOf

in following files:
constraintset_example_main.xml
constraintset_example_big.xml

Figure 14: Example of Stack Overflow question about an Android code sample.

We use the dataset provided by Stack Exchange[27] to mine Stack Overflow questions about Android and Spring code samples. We first run a script to select all questions with the URLs `github.com/googlesamples/` or `github.com/spring-guides/` in their bodies, which are official Android and Spring sample repositories on GitHub. From this data, we removed (i) questions with a score less than or equal to zero and (ii) questions without answers. This process resulted in 527 questions for Android and 87 questions for Spring, totaling 614. Figure 15 presents the distribution of the number of views of the selected questions. On the median, the Android questions have 890 views, while the Spring ones have 1,579. Notice that this is larger than general Android and Spring questions, which have 746 and 935 views, respectively, on the median.

After collecting the posts on Stack Overflow, we perform a manual classification. The second and third authors of this paper analyzed the questions using thematic analysis [7], a technique for identifying and recording *themes* in textual documents. This technique includes the following steps: (1) initial reading of the answers, (2) generating a first code for each answer, (3) searching for themes among the proposed codes, (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes [3]. Steps

[25]`https://stackoverflow.com/questions/49232559`
[26]`https://github.com/googlearchive/android-ConstraintLayoutExamples`
[27]`https://data.stackexchange.com/help`

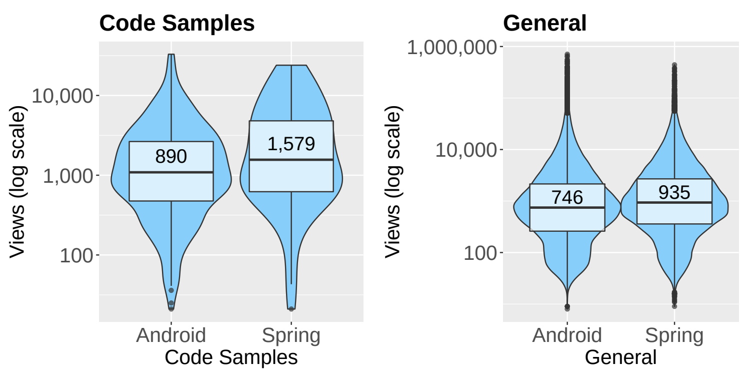


Figure 15: Distribution of the number of views of the selected Android and Spring questions.

1 to 4 were performed independently by two authors of this paper. We used the Cohen Kappa test [6] to measure the agreement: the score was 0.50 for Android (moderate agreement) and 0.34 for Spring (fair agreement). After this, the second and third authors held a sequence of meetings to resolve conflicts and assign the final themes (step 5).

5.1.2. Results

Our manual classification leads to four categories of questions:

- *Importing*: questions in which developers are trying to import the code sample to use or modify it but could not due to configuration issues.
- *Running*: questions in which developers are trying to run the code sample but could not due to runtime problems.
- *Modification*: questions in which developers are trying to modify or improve the code sample and faced some trouble.
- *Reference*: questions that include references to the code samples to illustrate some particular programming scenario or general doubts.

Figure 16 presents the distribution of the categories. The most common is the category *modification*, which represented 50.1% of Android and 44.8% of Spring questions. The second most common is *reference* (32.3% and 26.4%), which is followed by *running* (12% and 18.4%) and *importing* (5.7% and 10.3%) questions. Interestingly, both Android and Spring present the same order of questions, that is, (1) modification, (2) reference, (2) running, and (4) importing.

In the following lines, we detail each category and present concrete examples of the challenges faced by developers.

Importing. This category includes questions in which developers are trying to import the code sample to use or modify it but could not due to configuration issues, *e.g.,* external dependency is not properly imported, code is not

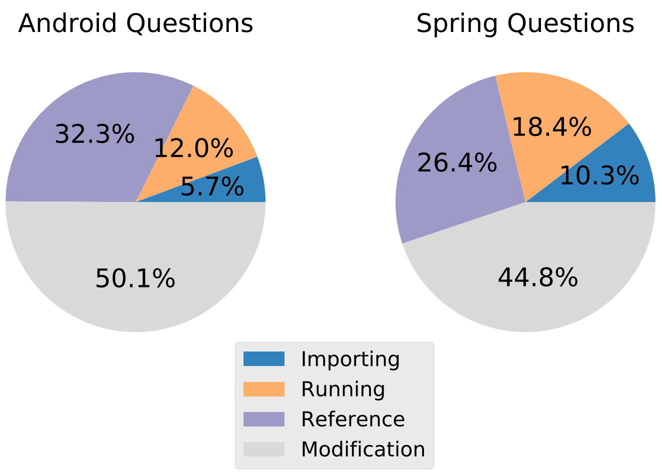


Figure 16: Stack Overflow questions by category.

compiling, IDE is not configured, etc. For example, in the following question, the developer cannot import and run the code sample in the Android Studio IDE.[28] The accepted answer simply presents step-by-step how the developer should successfully import the code sample using that IDE.

"No matter what projects I import they never work - Android Studio is always flagging this is not a Gradle build project [...] Can anybody tell me specifically how to import and run the following git repo in Android studio for example?"

Developers also have problems building the code samples, as in the following question, in which he is struggling with the gradle build.[29] The developer later discovers that the wrong Java version was being used (Java 1.8 instead of 1.7).

"I just cloned the project at GoogleSamples then cd to the native-activity dir. I typed: gradle clean build. And I am getting this: [...] I have no idea what's going on here. I updated to latest gradle 2.5 which supports 'model' in app script per the project requires."

As a final example in this category, we present a similar case in a Spring code sample. In this case, the developer tries to use Maven for building the code sample and faces an error message.[30] The solution involves adding a new dependency to the maven repository (in the pom.xml file) and upgrade maven to 3.0.5 above:

"I am going through this guide: https://spring.io/guides/gs/rest-service/ I use Maven for building, so I've fetched the pom.xml linked in the official Spring guide [...] I get the following error when running mvn install [...]"

[28] `https://stackoverflow.com/questions/31188849`
[29] `https://stackoverflow.com/questions/31506508`
[30] `https://stackoverflow.com/questions/22935840`

Running. This category includes questions in which developers are trying to run the code sample, but could not due to runtime problems. For instance, in the following question, the developer can run an Android code sample, however, it crashes in some specific cases.[31] The accepted answer states that this issue refers to a known bug, and there is no trivial solution to avoid it.

"I'm running the Barcode Reader example from Google Vision API, it works very well reading some 2d - pdf417 codes, but in some cases it crashes with a native exception attempting to use NewStringUTF like this: [...]"

Indeed, runtime problems may be diverse. The developers can run the code sample in the following two questions, but they face specific issues. In the first case, the Android developer[32] cannot run the code sample in some Android devices, whereas in the second case, the Spring developer[33] cannot kill the session.

"I'm testing Nearby connection API with the sample application available here: [...] It seems that this is not working for some devices. I successfully connected Samsung Galaxy S3 with Nexus 7 in both directions (S3 as host, N7 as slave, and vice versa). However, when I try to connect Samsung Galaxy S3 to Nexus 5, the connection ALWAYS fails, with status code 8005.'

"The problem that I am facing is, when I clicked the logout button this send a post request to the /logout endpoint to kill to session, but when I clicked the LogIn button again I expect to see the login Form Again."

Modification. This category is the most frequent in our manual classification. It includes questions in which developers are trying to modify or improve the code sample, but faced some trouble, for instance, while adding new features, using the sample in larger applications, performing migrations, etc. For example, in the following questions, the developers perform minor changes in the code sample, however, the modifications do not behave as expected. In the first question, the Spring modification resulted in an exception,[34] whereas in the second question, the developer reports a deformed image in Android.[35] In both cases, the answers are trivial, and it seems that the developers do not have enough experience.

"I'm having trouble with my first steps using Spring-Boot with JPA. I've started with a pretty minimalistic example from Git using Gradle. Now simply moving Customer to another package, let's say to hello2 results in an exception Caused by: java.lang.IllegalArgumentException: Not an managed type: class hello2.Customer."

"I tested with the GoogeSamples project android-Camera2Basic. But when I change the preview with a ratio of 1:1 image is deformed. Does anyone have an idea on this?"

[31]`https://stackoverflow.com/questions/43765499`
[32]`https://stackoverflow.com/questions/33763874`
[33]`https://stackoverflow.com/questions/37598036`
[34]`https://stackoverflow.com/questions/23366226`
[35]`https://stackoverflow.com/questions/34638651`

Besides performing minor changes, developers may also create applications based on the code samples. In the following examples, the developers are building custom cameras based on code samples provided by Android.[36][37]

"I'm building a custom camera using the new camera2 API. My code is based on the code sample provided by Google here. I can't find a way to get the camera preview in full screen. In the code sample, they use ratio optimization to adapt to all screens but it's only taking around 3/4 of the screen's height."

"I'm creating a custom camera capturing videos with the new camera2 API. My code is strongly inspired from the code provided by Google here. My camera preview has a button to switch from back to front camera and then from front to back camera [...]. For some reason, when I click on the "switch/swap camera" button for the first time, it brings be to the front camera as it should, BUT everytime I click again, the switch/swap doesn't work anymore."

As a final example, we present a question in which the developer aims to expand the code sample considerably, taking into account security issues.[38]

"I would like to be able to upload images to a server, handling errors and exceptions gracefully [...]. Using the example project gs-uploading-files I can upload files to a server using Spring Boot and Thymeleaf. In application.properties I set [...]. However several security and validation issues are unresolved when I upload files larger than 1MB."

Modification tags. To further explore the questions related to modification, we assess their tags. In Stack Overflow, a tag is a word or phrase that describes the topic of the question.[39] For that analysis, we select all tags of the analyzed questions and remove noisy ones, such as the framework name, framework versions, and others. Finally, we merge similar tags for the sake of clarity, for example, the tags "android-camera2", "android-camera", "camera", "camera2", "front-camera", and "camera-api" become *camera*.

Table 12 summarizes the most common tags for each framework. *Camera* is the most common tag in the Android framework (105 questions). It refers to the Camera API,[40] an Android library that provides camera features for distinct devices. The second most common tag is *vision* (36). The Mobile Vision API is part of the Machine Learning Kit[41] and provides a framework for finding objects in photos and video as face, barcode, and text detection.[42] The next tag is *setup* (29), which is a merge of two other tags: ndk and studio. The Android NDK[43] is a toolset that allows apps to be implemented in native code (using languages such as C and C++), while the Android Studio is the

[36]`https://stackoverflow.com/questions/39044494`
[37]`https://stackoverflow.com/questions/39022845`
[38]`https://stackoverflow.com/questions/40355743`
[39]`https://stackoverflow.com/help/tagging`
[40]`https://developer.android.com/guide/topics/media/camera?hl=en_us`
[41]`https://developers.google.com/ml-kit`
[42]`https://developers.google.com/vision/introduction`
[43]`https://developer.android.com/ndk`

official IDE for building Android apps.[44] The next tag is *dagger* (31), which is a dependency injection framework for Java, Kotlin, and Android.[45] Lastly, we find *architecture components* (24), which is a merge of architectural tags as android-viewmodel, android-room, and android-livedata; they are all related to the Android app architecture.[46]

Table 12: Most common tags of modification questions.

Android		**Spring**	
Tags	#	Tags	#
camera	105	security	36
vision	36	data	14
setup	29	social	6
dagger	31	cloud	4
architecture components	24	maven	3

For Spring Boot, we note that the most common tag is *security* (36). Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.[47] The second tag is *data* (14): Spring Data's goal is to provide a familiar and consistent, Spring-based programming model for data access.[48] The third tag is *social* (6), which is a tool to connect Spring application with Software-as-a-Service (SaaS) API providers such as Facebook, Twitter, and LinkedIn.[49] The next tag is *cloud* (4): Spring Cloud provides tools for developers to build some of the common patterns in distributed systems quickly.[50] Lastly, we have the tag *maven* (3). Apache Maven is a software project management and comprehension tool that can manage a project's build, reporting, and documentation.[51]

Overall, these results suggest that developers have issues modifying distinct types of code samples, as presented by the variation of detected tags. In both frameworks, the doubts are not concentrated on a single tag, but spread over several ones.

Reference. This final category contains questions with references to the code samples to illustrate some particular programming scenario or general doubts. For example, in the following question, the developer is simply illustrating his problem with reference to an Android code sample.[52]

"My main requirement would be to have a service having its own process and trigger its own geofencing event [...]. Then there is this code sample from Google

[44]`https://developer.android.com/studio/intro`
[45]`https://dagger.dev`
[46]`https://developer.android.com/jetpack/guide#recommended-app-arch`
[47]`https://spring.io/projects/spring-security`
[48]`https://spring.io/projects/spring-data`
[49]`https://projects.spring.io/spring-social/`
[50]`https://spring.io/projects/spring-cloud`
[51]`https://maven.apache.org/`
[52]`https://stackoverflow.com/questions/28355353`

showing how to use geofencing with google play services: Google samples geofencing. What I found so far is that we have to use an IntentService to trigger geofencing events, and from the docs, I've read it states that an IntentService terminates itself when its work is done"

In the next example, the developer is curious about the design of the code sample and looks for explanations about it.[53]

"Does anybody know why the Spring Boot Guide includes two different types of integration tests? [...]"

Finally, in the following question, the developer references the code sample to illustrate his doubt with a concrete example better.[54]

"Does the Google Mobile Vision API work offline? Or does it need Internet connectivity? The sample app does not require any Internet permission. Which means the API works entirely offline. I am looking for a positive confirmation of this. [...]"

Lesson Learned 5: Developers typically face problems when trying to modify the code samples, for example, when adding new features or performing minor changes to explore them. This category corresponds to 50% of the cases in Android questions and 45% in Spring Boot.

5.2. GitHub Issues

5.2.1. Analysis

To further explore developers' problems about code samples, we assess GitHub issues of the code samples. Here, we are interested in issues that cause changes in code samples. For this purpose, first, we selected all issues for studied code samples. Next, we removed all open issues as they do not cause code sample modification. When an issue causes modification in a repository, it is common to reference the commit or pull request with the modifications. We then removed issues without reference to commits or pull requests, only keeping the ones that cause modification. Lastly, we manually removed false positive references. Based on that, we found 269 GitHub issues from code sample repositories.

Based on the title, body, and comments of the issues, we classify them in the following categories: (i) *importing*, when developers try to import the code sample to use or modify it but could not due to configuration issues; (ii) *running*, when developers try to run the code sample but could not due to run-time problems; (iii) *modification*, when developers try to modify the code sample, and faced some trouble; (iv) *improvement*, when developers suggest an improvement into the code sample or their comments led code sample' maintainers to improve them; and (v) *question*, when developers are simply asking about code sample usage or better patterns.[55]

[53]`https://stackoverflow.com/questions/46732371`
[54]`https://stackoverflow.com/questions/40832882`
[55]The first three categories come from our prior analysis on Stack Overflow.

Furthermore, we look at the changed files in the commits related to the issue. We classify the changes as follows: (i) *documentation* changes, when maintainers edit documentation files as *readme.md*; (ii) *source code* changes, when maintainers edit Java files; and (iii) *configuration* changes, when maintainers edit configuration files, such as manifest.xml, pom.xml, or build.gradle.

The manual classification was performed as in the Stack Overflow study, that is, based on thematic analysis [7].

5.2.2. Results

Figure 17 presents the distribution of the categories after the manual analysis. The most common category is *improvement*, with 40.9% in Android and 65.5% in Spring Boot code samples. This may suggest that developers who use code samples want to improve them, and, at the same time, maintainers care about improvements and developers' requests. In both frameworks, the second most common is *importing*, with 30.3% in Android and 13.8% in Spring Boot code samples. *Running* has 16.7% in Android code samples and 11.8% in Spring Boot ones. The fourth category is *question* (7.6%) in Android but *modification* (4.9%) in Spring Boot. Finally, *modification* has 4.5% in Android code samples, and the *question* category has 3.9% in Spring Boot ones.

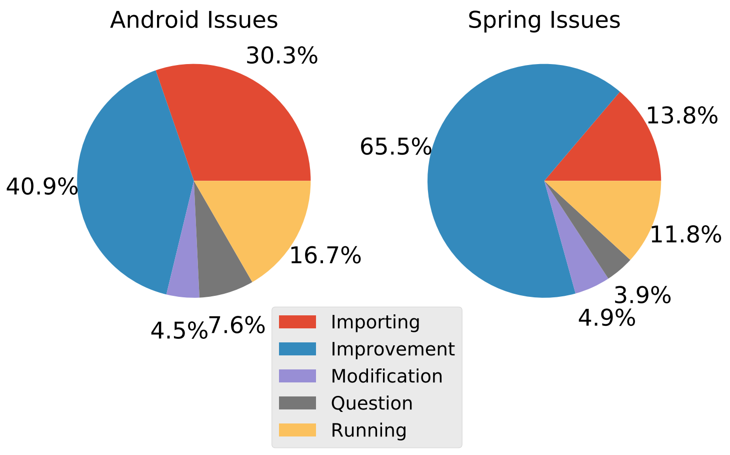


Figure 17: Distribution of issues' categories.

Figure 18 presents the distribution of changes type led by issues. In Android, we notice that *source code* changes are the most common type of change with 51.9%. Followed by *configuration* with 41.3% and *documentation* with 6.7%. In contrast, in Spring Boot, we observe that *documentation* is the most common type of change with 43%, while *configuration* and *source code* have 34.7% and 22.3%, respectively.

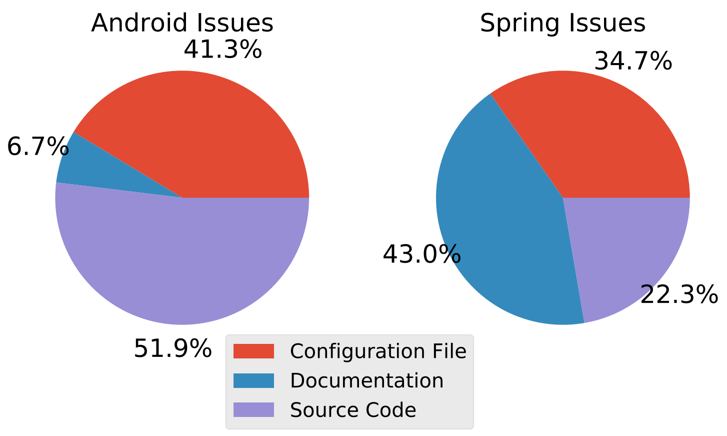


Figure 18: Distribution of modification types.

Lesson Learned 6: Developers create issues mainly suggesting improvements to code samples. Since we only assessed issues that changed the code samples, this shows that those issues are well accepted by code samples maintainers.

6. Implications

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

Code samples should be simple and small to facilitate their reuse, as stated by good development practices [40]. Indeed, the majority of the code samples provided by Android and Spring Boot follow this rule. However, this is not strict: we find that the code samples with more Java files are more likely to be popular than those with fewer Java files.

Code samples should provide working environments to ease their usage. Most Android and Spring Boot code samples are formed by source code and many other configuration files necessary to run them properly. Automated build and integration tools may also support both the creators and clients, improving their quality and reducing risks [8, 22, 39].

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 outdated and less attractive to the clients [12, 15, 19, 42]. Indeed, this practice is often performed by Android and Spring Boot 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. Our qualitative analysis on Stack Overflow and GitHub issues also suggests that the code samples are likely to change over time due developers' needs.

Code samples may benefit from scenarios where their developers also contribute to the framework itself. The migration delay may decrease when the overlap of developers is higher between code samples and framework. We recognize, however, that we should further explore this phenomenon in future research.

Code samples should encourage fork-change-learn approach. We recall that code samples facilitate and accelerate the learning process of features provided by frameworks [31, 40]. Despite the majority of the forked code samples are inactive, we find that the a non-negligible percentage is updated by their clients as a way to explore and learn them. Also, our findings in the qualitative analysis shows that developers are likely to change the code samples, which reinforces the importance of the learning perspective of the code samples. Thus, we recommend the cycle fork-change-learn to the clients kick start in a code sample.

Code sample creators can provide extensions guides to aid developers. We found that developers frequently try to modify or improve the code samples, but face some problems, for instance, expanding the sample with novel new features. We also detect that developers may even suggest the improvement of code samples via GitHub issues. For example, we find that many questions are created when developers try to use camera API on Android and security features on Spring Boot. Maybe developers would not create these questions if organizations made available extra content explaining how to evolve code samples, including the use of different related features (*e.g.,* more complex use of common features). For instance, the basic code sample explains how to use a camera to take a picture, but the extra content could explain how to switch between the front and back camera, turn on the flashlight, or take a picture within canvas drawing. This could help to spread the technology and also to support developers.

7. Threats to Validity

This section discusses the study limitations based on the four categories of validity threats described by Wohlin et al. [41]. 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 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 to avoid considering aspects that 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 than other programming languages and affect 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 many code samples in the GitHub repository.

External validity: Threats associated with external validity concern the degree to which the findings can be generalized 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 compared to other frameworks. This risk cannot be avoided because our focus is on the two frameworks presented in Section 3. 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 the first step towards the generalization of the results.

8. Related Work

Frameworks are used to support development, provide source code reuse, improve productivity, and decrease costs [14, 25, 30]. 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 [36]. 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 and explore properties of those samples.

Context-based code samples. Software engineering tools bring sophisticated search power into the development environment by extending the browsing and searching capabilities [11, 17, 29, 34, 36]. For instance, Holmes and Murphy [11] proposed a technique that recommends source code examples from a repository by matching structures of given code. FuzzyCatch [27] provides a code recommendation tool, based on fuzzy logic, for handling exceptions. XSnippet [34] 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 [11]). However, the samples provided by these systems are highly dependent on a particular development context. In contrast, code samples typically are complete projects that organizations 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 compared to samples automatically generated by tools.

Mining API usage examples. Complementing the aforementioned tools, many studies confirmed 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 [5, 13, 23, 24, 28, 46, 1, 38, 26]. Most of these work retrieve the so-called code snippets to support API learning, whereas our work focuses on complete projects of framework code samples. In addition, our work is not focused on proposing an approach to mine code samples, but analyze the characteristics of these code samples.

Assessing Q&A code snippets. Nasehi et al. [35] focused on finding the characteristics of a good example on Stack Overflow. They adopted an approach based on high/low voted answers, the 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. [43] 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. A similar work was done by Uddin *et al.* [9] that assesses the prevalence and vulnerabilities of share code examples using C# unsafe keyword in Stack Overflow. They assess using regular expressions and manual checks. Meldrum *et al.* [20] evaluate the quality of code snippets on Stack Overflow, exploring aspects as reliability and conformance to programming rules, readability, performance, and security. Finally, studies are analyzing the adoption of code snippets [10, 33, 44]. For instance, Roy and Cordy [33] 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 the properties of code snippets and their adoption on real projects. However, our work targets entire code sample projects instead of code snippets.

9. Conclusion

We proposed a large-scale empirical study to understand better how these code samples are maintained and used by developers. By assessing 233 code samples about Android and Spring Boot, 233 conventional projects related with these frameworks 614 Stack Overflow questions and 269 GitHub issues, we investigated aspects related to their source code, evolution, popularity, client usage, and developers' problems. We found that most code samples are small and simple, provide a working environment for the clients, and rely on automated build tools. They frequently change, for example, to adapt to new framework versions. We also detected that clients commonly fork the code samples, however, they rarely modify them. We also found that the most common problem happens when developers try to modify the code samples. Finally, we found that the most common type of GitHub issue on code sample repositories is related to improvements.

We reiterate the most interesting implications to support the maintenance and usage of code samples:

- Code samples should be simple and small to facilitate their reuse, as stated by guidelines and followed by most of the code samples of Android and Spring Boot.
- Code samples should provide working environments to ease their usage and rely on automated build and integration tools 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 outdated 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.
- Code sample creators can provide extensions guides to aid developers. As developers frequently try to modify the code sample, this could help spread the technology and train them.

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 performance and security. Finally, we plan to perform a survey with the creators and clients of the code samples to understand better major limitations and benefits from their point of view.

10. Acknowledgment

This research is supported by CNPq (438017/2018-8 and 133898/2020-2).

References

[1] Barnaby, C., Sen, K., Zhang, T., Glassman, E., Chandra, S., 2020. Exempla gratis (eg): code examples for free, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1353–1364.

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

[3] Brito, A., Valente, M.T., Xavier, L., Hora, A., 2020. You broke my code: Understanding the motivations for breaking changes in APIs. Empirical Software Engineering 25, 14581492.

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

[5] Buse, R.P.L., Weimer, W., 2012. Synthesizing api usage examples, in: International Conference on Software Engineering, pp. 782–792.

[6] Cohen, J., 1960. A coefficient of agreement for nominal scales. Educational and psychological measurement 20, 37–46.

[7] Cruzes, D.S., Dyba, T., 2011. Recommended steps for thematic synthesis in software engineering, in: International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275–284.

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

[9] Firouzi, E., Sami, A., Khomh, F., Uddin, G., 2020. On the use of c# unsafe code context: An empirical study of stack overflow, in: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–6.

[10] Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., Irlbeck, M., 2011. On the Extent and Nature of Software Reuse in Open Source Java Projects, in: International Conference on Top Productivity Through Software Reuse, pp. 207–222.

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

[12] Hora, A., Robbes, R., Valente, M.T., Anquetil, N., Etien, A., Ducasse, S., 2018. How do developers react to API evolution? a large-scale empirical study. Software Quality Journal 26, 161–191.

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

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

[15] Kula, R.G., German, D.M., Ouni, A., Ishio, T., Inoue, K., 2018. Do developers update their library dependencies? Empirical Software Engineering 23, 384–417.

[16] Lethbridge, T.C., Singer, J., Forward, A., 2003. How software engineers use documentation: The state of the practice. IEEE Software , 35–39.

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

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

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

[20] Meldrum, S., Licorish, S.A., Owen, C.A., Savarimuthu, B.T.R., 2020. Understanding stack overflow code quality: A recommendation of caution. Science of Computer Programming 199, 102516.

[21] Menezes, G., Cafeo, B., Hora, A., 2019. Framework code samples: How are they maintained and used by developers?, in: 13th International Symposium on Empirical Software Engineering and Measurement(ESEM), pp. 1–11.

[22] Meyer, M., 2014. Continuous integration and its tools. IEEE Software 31, 14–16.

[23] 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.

[24] 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.

[25] Moser, S., Nierstrasz, O., 1996. The effect of object-oriented frameworks on developer productivity. Computer 29.

[26] Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., Di Penta, M., 2019. Focus: A recommender system for mining api function calls and usage patterns, in: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), IEEE. pp. 1050–1060.

[27] Nguyen, T., Vu, P., Nguyen, T., 2020. Code recommendation for exception handling, in: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1027–1038.

[28] Niu, H., Keivanloo, I., Zou, Y., 2017. Learning to rank code examples for code search engines. Empirical Software Engineering 22, 259–291.

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

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

[31] Robillard, M.P., DeLine, R., 2011. A field study of API learning obstacles. Empirical Software Engineering 16, 703–732.

[32] Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J., 2006. 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, pp. 1–33.

[33] Roy, C.K., Cordy, J.R., 2010. Near-miss function clones in open source software: An empirical study. Journal of Software: Evolution and Process 22, 165–189.

[34] Sahavechaphan, N., Claypool, K., 2006. Xsnippet: Mining for sample code, in: Conference on Object-oriented Programming Systems, Languages, and Applications, pp. 413–430.

[35] Sillito, J., Maurer, F., Nasehi, S.M., 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.

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

[37] 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, pp. 301–310.

[38] Uddin, G., Khomh, F., Roy, C.K., 2020. Mining api usage scenarios from stack overflow. Information and Software Technology 122, 106277.

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

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

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

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

[43] Yang, D., Hussain, A., Lopes, C.V., 2016. From Query to Usable Code: An Analysis of Stack Overflow Code Snippets, in: International Conference on Mining Software Repositories, pp. 391–402.

[44] 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.

[45] Zhang, J., He, J., Ren, Z., Zhang, T., Huang, Z., 2019. Enriching api documentation with code samples and usage scenarios from crowd knowledge. IEEE Transactions on Software Engineering PP, 1–1. doi:10.1109/TSE.2019.2919304.

[46] 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.