# Exploring Code Samples Characteristics and Their Impacts on Software Ecosystems

*Gabriel Santana de Menezes*

# Exploring Code Samples Characteristics and Their Impacts on Software Ecosystems[1]

*Gabriel Santana de Menezes*

**Advisor:** *Prof. Bruno Barbieri de Pontes Cafeo, Ph.D.*
**Co-Advisor:** *Prof. André Cavalvante Hora, Ph.D.*

Dissertation delivered to the Faculty of Computing (FACOM/UFMS) as part of requirements to obtain the title of Master in Computing Science.

**UFMS**
**August/2022**

[1]This work was supported by CNPq Grant No. 133898/2020-2

# Acknowledgements

Since graduation, when I worked on a research project that later became my master's work, I always feel lucky to have Professor Bruno and Professor André as advisors. I am immensely grateful to my advisor, Professor Bruno Cafeo, who helped me a lot throughout this master's project, for his patience and enthusiasm to share his knowledge, which motivated me to continue. My thanks go to my co-advisor, Professor André Hora, who helped us in the construction of the work, through meetings where we discussed ideas about the work and through revisions of texts that helped me in the evolution and improvement of my way of writing. I also thank Professor Awdren Fontão, either for his collaboration in the development of scientific papers, or suggestions reported on my qualification board or as coordinator of the Software Engineering Laboratory (LEDES). I am also immensely grateful to Professor Hugo Borges for the contributions and suggestions made on my qualification board.

I would like to thank my mother, Ana Maria, who always educated and supported me from my childhood until her last day of life. I dedicate this research to her. I also want to thank my girlfriend, Tatiane, who supported and helped me during the difficult times throughout this research. This includes my brother, Leandro, and my father, Aparecido.

I would also like to thank the friends I made during this research. Sharing knowledge between us was very enriching. I thank all the members of LEDES. Special thanks to Adriano Marques, Camila Minei, Gabriel Colman, Karolina Milano, Matheus Melo, Pedro Arantes, Sábia Belle and Willian Marotzk.

Finally, I gratefully acknowledge CNPq for the funding that made this research possible.

# Abstract

Modern software systems are commonly built on top of frameworks, libraries, and APIs (platforms). The environments where exists relations between organizations that maintain these platforms and clients that use features of these platforms are known as the Software Ecosystem. In this context, organizations develop code samples to help their client with learning barriers. Code samples are small software projects, with educational purposes, and teach how to use platform features. However, we know little about code sample characteristics and their relation with organizations and clients. In this work, we aim to fill these gaps by assessing four different aspects. *First*, comparing code samples with conventional projects through their source code. *Second*, exploring the code sample usage via Stack Overflow and GitHub. *Third*, assessing the profile of actors that interact with code samples. *Fourth*, maintenance of code samples and their impact on clients. We found that code samples are smaller and simpler than conventional projects. We also found that code sample changes less frequently but updates faster to new platform versions than conventional projects. Regarding code sample usage, we found that the copy/paste approach is low used by clients. Also, we noted that the most common problem faced by clients is when they try to modify the code sample and improvements are the most common need from clients. Regarding actors around code samples, we found that the target audience of code samples can range from inexperienced to experienced clients. Also, we noted that platforms of different organizations seem to have different target audiences. Also, code sample maintainers are aged and unpopular on GitHub. Finally, about code sample maintenance, we found that code modifying is the most common maintenance activity of code samples, but Pull Request management plays an essential part in maintenance time. We also found that code samples become less complex but larger and less readable over time.

# Resumo

Os sistemas de *software* modernos são geralmente construídos sobre *frameworks*, bibliotecas e APIs (plataformas). Ambientes onde existem relações entre as organizações que mantêm essas plataformas e os clientes que utilizam recursos dessas plataformas são conhecidos como Ecossistema de *Software*. Nesse contexto, as organizações desenvolvem *code samples* para ajudar seus clientes com as barreiras de aprendizado. *Code samples* são pequenos projetos de *software*, com fins educacionais, e ensinam como usar os recursos da plataforma. No entanto, sabemos pouco sobre as características de *code samples* e sua relação com organizações e clientes. Neste trabalho, pretendemos preencher essas lacunas avaliando quatro aspectos diferentes. *Primeiro*, comparando *code samples* com projetos convencionais através de seu código-fonte. *Segundo*, explorando o uso de *code samples* via Stack Overflow e GitHub. *Terceiro*, avaliando o perfil dos atores que interagem com *code samples*. *Quarto*, manutenção de *code samples* e seu impacto nos clientes. Descobrimos que os *code samples* são menores e mais simples do que os projetos convencionais. Também descobrimos que *code samples* mudam com menos frequência, mas atualiza mais rapidamente para novas versões de plataforma do que os projetos convencionais. Em relação ao uso de *code samples*, descobrimos que a abordagem copiar/colar é pouco utilizada pelos clientes. Além disso, notamos que o problema mais comum enfrentado pelos clientes é quando eles tentam modificar a *code samples* e as melhorias são a necessidade mais comum dos clientes. Em relação aos atores em torno dos *code samples*, descobrimos que o público-alvo dos *code samples* pode variar de clientes inexperientes a clientes experientes. Além disso, notamos que plataformas de diferentes organizações parecem ter diferentes públicos-alvo. Além disso, os mantenedores de *code samples* são antigos e impopulares no GitHub. Por fim, sobre a manutenção de *code samples*, descobrimos que a modificação de código é a atividade de manutenção mais comum de *code samples*, mas o gerenciamento de Pull Rquests desempenha um papel essencial no tempo de manutenção.

Também descobrimos que os *code samples* se tornam menos complexas, mas maiores e menos legíveis ao longo do tempo.

# Contents

# List of Figures

# List of Tables

# Acronyms

**PR** Pull Request. 6, 54, 63, 65, 66, 69–71, 84, 86, 89

**SECO** Software Ecosystem. xv, 1, 2, 5–8, 31, 51, 54, 60, 63, 70, 89, 90

**SO** Stack Overflow. xv, 2, 3, 6, 51–56, 64, 65, 68, 69

# Chapter 1

# Introduction

Nowadays, software development is commonly supported by frameworks, libraries, and APIs. In the context of this work, we called framework, library, and API as platforms. The use of these platforms can provide feature reuse, improve productivity, and decrease costs [44, 68, 80]. These platforms support the development of different niches of tools, like mobile apps, web apps, responsive interfaces, cross-platform systems, cloud computing, distributed systems, and others. These platforms are widely used in the industry. In the Java ecosystem, for example, there are more than 450,000 platforms available in the Maven repository [27]. In the Python ecosystem are more than 350,000 platforms [26] made available via Python Package Index. And in the JavaScript ecosystem, there are more than 400,000 platforms [14].

In some environments, can exists relations between organizations that develop and maintain these platforms and clients that use the functionalities provided by the platforms to build new tools or software. This environment is studied as Software Ecosystem (SECO) [24, 45, 51]. These relations are not limited to just technical, but also social and business relations [8, 25, 51]. In this context, it is common that newcomers clients want to learn how to use the platform features. In the same way, experienced clients want to update their knowledge about new features provided by the platform. For both situations, in this work we called as learning process.

There are several barriers that the clients may face on the learning platform usage [81, 82, 96, 98, 115]. For instance, the lack of code examples that illustrate scenarios of platform usage [81, 103]. Another example is that due to the competitiveness of the job market, clients need to learn how to use the platform as quickly as possible [115]. To facilitate and accelerate the learning process of features provided by platforms, organizations commonly made available code samples to assist development efforts [20, 59]. Code samples are small software projects, with educational purposes and stored in code repositories, implementing platform functionalities as examples to clients [20, 59, 93]. Code samples are often provided by worldwide software projects and organizations, such as Android [30], Spring [94], Google Maps [31], Twitter [101], Microsoft [63], to name a few.

Despite the widespread adoption of code samples by organizations, we know little about their structure, maintenance, and usage. Exploring these aspects makes a two-way contribution. *First*, is the increase in the knowledge that exists about code samples in the literature. *Second*, assist organizations in building and maintaining their code samples. For example, we do not know the differences and similarities between code samples and conventional projects. We know that conventional projects have been well explored over the years. Therefore, if organizations know which aspects of code samples are similar to conventional projects, they can use similar solutions to possible problems related to these aspects in code samples. If the possible problems found in code samples are about aspects in which they differ from conventional projects, organizations should look for alternative solutions. From this perspective, we created the *first goal* of this work: Explore the differences and similarities between code samples and conventional projects, comparing structural evolutionary and usage characteristics, to increase the knowledge about code samples. We show this study in Chapter 3.

When exploring the use of code samples compared to conventional projects, we realized that the approach studied is not often used by code sample clients. In addition to help to fill the lack of knowledge about code sample usage, exploring how clients use code samples can help organizations to understand the clients' needs and their problems faced with code sample use. In this way, organizations can direct efforts to meet these needs and create tools that help clients to face problems these problems. This motivated us to create our *second goal*: Assess code sample usage and their problems faced by clients, through GitHub and SO, to help organizations to deal with the creation of solutions for these problems. We present this study in Chapter 4

Since code samples belong to a SECO context, it is natural that there is an interaction between them and other actors that belong to their SECO. In Chapter 4, we found that the most common problem reported by clients was when they tried to change the code sample to another context, extending its usage. This made us think about the degree of experience of code sample clients. By getting to know your profile better, organizations can create code samples that are increasingly suitable for the target audiences of their code samples. In addition to clients, maintainers also interact with code samples. To help fill the lack of knowledge about them, we also evaluated their characteristics. These reasons led us to create our *third goal*: Explore characteristics from actors that interact with code samples, via SO and GitHub information, to fill the gap of knowledge and help organizations to create more suitable code samples. We present this study in Chapter 5.

Another little-explored aspect is the code sample maintenance. In our previous study [59], we assessed code sample evolution by analyzing commit frequency, lifetime, most changed file types, and time to upgrade to a new platform version. However, the results obtained were still not enough to know, for example, if code samples get larger and more complex over time. In addition, Chapter 4 stated that clients typically face problems when they try to modify the code samples and their main suggestion of change is the code sample improvement. This is important because code samples are projects with the educational purpose of helping clients to understand how to use the platform. To fill this gap of knowledge and to help organizations to understand the maintenance activities happen, we built our *fourth goal*: Evaluate how organizations maintain their code samples over time and the impact of this maintenance on clients, analyzing their maintainers' activities, source code evolution, and clients' doubts on Stack Overflow (SO), to provide insights for other organizations. We show this study in Chapter 6.

After carrying out the studies, aiming to achieve the goals, we can highlight the following contributions of this work. First: We provide a large empirical study on code samples from Android, AWS, Azure, and Spring platforms, to increase the knowledge about code samples, their organization, and their clients. Second: We provide a set of information on how code samples are created and maintained by the organizations analyzed. Such information may help organizations to create and maintain their code samples. Third: We provide an initial study about the profile of clients from code samples. This may encourage further studies on who the clients of code samples are and help organizations to build more appropriate code samples. Fourth: 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.

As a result of this work, we had two published articles. A workshop paper accepted in the *Workshop on Software Visualization, Evolution, and Maintenance* [10], and a journal paper accepted in the *Journal of Systems and Software* [60]. The rest of this work is organized as follows. Chapter 2 presents the knowledge needed to understand this work. Chapter 3 shows details about the study comparing code samples and conventional projects. Chapter 6 presents the study about code sample maintenance and the characteristics of code sample maintainers. Chapter 5 and Chapter 4 bring the study on who the code sample's clients are and how they use them. Finally, Chapter 7 presents the conclusion of this work and the future work.

# Chapter 2
# Background

In this chapter, we present the background needed to understand this work. Section 2.1 presents definitions and characteristics of SECOs. Section 2.2 presents concepts about learning process and barriers faced by SECO clients. Section 2.3 presents definitions and properties of code samples. Finally, Section 2.4 presents characteristics from organizations that were selected and the motivation to their selection.

## 2.1 Software Ecosystem

The field of Software Ecosystem (SECO) is still recent and immature, but it is maturing and consolidating through the growing number of published papers, presence in journals, empirical models, and the number of ecosystems analyzed [51, 52] Through this consolidation, SECO has different definitions. Here we present the definitions important to our context. Bosch defines SECO as a set of software solutions that allow activities and transactions by actors in a social or business ecosystem and the organizations that provide these solutions [8]. Manikas stated that the software and actor interaction in relation to a common technological infrastructure, results in a set of contributions and influences directly or indirectly the ecosystem [51].

The SECO field uses theories from other fields, including concepts of interaction. However, in the SECO context, the software interaction, and actors' interaction have equal importance to keep the ecosystem alive [52]. In the context of this work, we highlight two relevant actors: organization and clients. The organization is responsible to provide the platform, to define good practices for its use, and to attempt to clients' needs to evolve the platform [40].

The client uses the platform features to create new projects [40].

The interactions between SECO actors can happen through functionalities provided by repositories. We can mention the interactions on Q&A sites like Stack Overflow (SO) and on remote repository managers like GitHub. SO is a Q&A platform for professional and enthusiast programmers. [1] The SO uses an approach that the user community itself decides which questions and answers are most relevant. Good answers are voted up and rise to the top, and the best answers show up first so that they are always easy to find. In the same way, questions and answers have their metric of relevance, users also have theirs, called reputation. Reputation score goes up when others vote up your questions, answers, and edits. At the highest levels, the user can access special moderation tools, and work alongside our community moderators to keep the site focused and helpful. In the context of SECO, clients can interact with other actors, using SO to solve their doubts or find practical solutions to problems that other people may have already faced. They can find the best answers, with relevant comments and code snippets. These responses are often made by highly reputable users, can be considered experienced clients or even organization people.

GitHub is a code hosting platform for version control and collaboration. It lets the users work together on projects from anywhere. [2] As SECOs are composed by the interaction between actors around a platform, GitHub is a useful tool to allow these interactions through Pull Request (PR) and Issues. PRs are essential for collaboration on GitHub. A PR contains proposed changes and requests to someone to review and merge them into the repository. Trough PR, clients can contribute to platform inserting and modifying code. On the other hand, Issues allows clients to report problems or request new changes to the platform. GitHub Issues is a tracking tool that is integrated with your GitHub repository. [3] Issues are useful for discussing specific details of a project such as bug reports, planned improvements and feedback. [4]

Figure 2.1 shows an example of SECO. Android is the platform in the center of SECO. Google is the organization and it is responsible for making Android available and for evolving it. Developers, here named as clients, use the Android software development kit to create Android apps and make them available in Google Play.

[1] https://stackoverflow.com/tour
[2] https://docs.github.com/en/get-started/quickstart/hello-world
[3] https://www.ibm.com/garage/method/practices/think/tool_github_issues/
[4] https://docs.github.com/en/get-started/quickstart/communicating-on-github

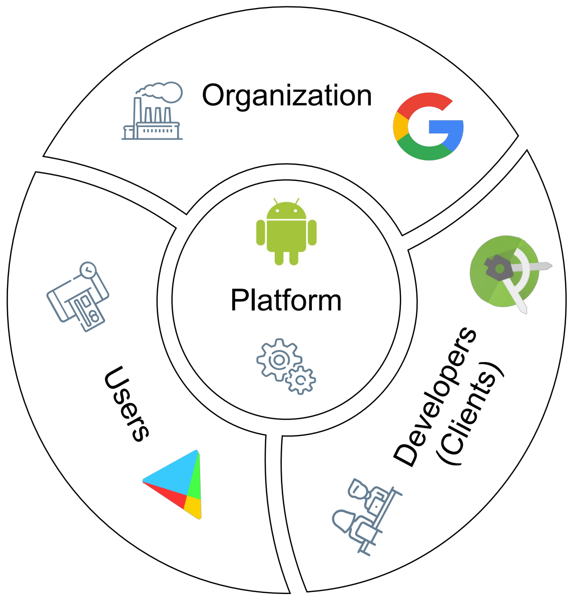


Figure 2.1: SECO representation

To build a successful SECO, organizations need to meet the ecosystem needs, and use business or motivation to incentive the actors to contribute to the ecosystem evolution [51, 52]. The interaction between the actors results in contributions to evolve the platform and the ecosystem it self. A contribution can be technical, for instance, a commit to the platform repository or an external plugin. A contribution can be non-technical, for example, user data or reselling [51]. In the same way, SECO needs to be organized and even managed in some aspects, either by for-profit organizations or by non-profit communities [51].

If the organization does not take successful strategies to organize, manage and govern, the SECO can to fail on meet the client's needs into the platform [45, 105]. When this situation occurs frequently over a long period of time, clients may abandon the use of the platform and consequently abandon the SECO. An important aspect of SECO management is the learning process of clients on understanding features provided in platforms. For that, organizations have to create alternatives to help their clients in this process.

## 2.2 Learning Process in SECO

One of the strategies that organizations can take is to help with the barriers that clients face when they learn to use the platform. The process of clients learning how to use features provided by the platform plays an important role in SECO life cycle. However, there are some barriers that make it difficult for the client to learn and use the specific platform features: (i) possible lack of motivation to read traditional documentation [98], (ii) difficulty to understand and to use specific features [96], (iii) large number of features [82], (iv) need for quick learning [115], and (v) few complete examples [81].

Clients in a SECO seek to overcome these barriers through different sources of information. The main and most complete is the traditional and official documentation made available by organizations. Blog posts and videos can also be another source of information to assist clients, usually developed by more experienced clients within SECO or even members of the organization. We can also mention the Q&A sites, which provide more explanatory descriptions of specific platform usages relevant to the clients, and are generally of good quality [3, 75]. Besides that, as we know, platforms receive constant updates and the information found in Stack Overflow should be updated too, including code as well [75]. In an attempt to combine the benefits found in both proposals, that is, knowledge and use of specific platform functionalities, frequently with code, and knowledge about best practices and the frequent updating provided by the organization, code samples are raise as an alternative to help clients on platform features learning.

## 2.3 Code Sample

As a relatively recent concept coming from the industry, organizations define code samples in different ways. For example, Oracle states that "code sample is provided for educational purposes or to assist your development or administration efforts [93]." Similarly, Spring reports that "code samples are designed to get you productive as quickly as possible [95]." And Mozilla says "code samples need to be simple enough to be understandable, but complex enough to do something interesting, and preferably useful [20]." Literature defines code sample as a complete software project with education purpose, made available by organizations, to assist their clients on understanding, using and staying up to date with their product features [59].

The number of organizations that provides code samples to help their clients to deal with their platform features is growing. For example, Google provides code samples to help clients to use Android features [30, 31]. Spring ecosystem has more than 60 code samples, to support Spring Boot clients to build web application [94] and over 30 code samples to support Spring Cloud clients on the build of distributed systems [18]. Also, we can cite organizations Twitter [101], Microsoft Azure [62] and Amazon AWS [2]. This growth is perhaps due to the need that clients have to obtain examples of how to use the platform features. For instance, a recent research with more than 2,000 developers showed that almost 80% of participants think the lack of examples of platform usage is a problem in its understanding [112].

To illustrate the important characteristics of code samples already explored in the literature, we selected an example of a code sample. The selection of the code sample was to corroborate the characteristics already evaluated. The chosen code sample is *gs-spring-boot*, made available on GitHub [5] by Spring Boot to help clients to learn how to build a Spring Boot application with minimal configuration.

**Code samples should be simple and small to facilitate reuse and understanding [20, 59]**. We can notice these characteristics in our example. Figure 2.2 shows the folders and files of *gs-spring-boot*. We can observe a total of six source code files. Figure 2.3 and Figure 2.4 illustrates the main lines of code from *gs-spring-boot*. Figure 2.3 presents the main Java class of the code sample. This class is important because it initiate the Spring Boot application through the method run from *SpringApplication* class. From that, *SpringApplication* bootstraps our application and starts the auto-configured Tomcat web server. We need to pass *MyApplication.class* as an argument to the run method to tell *SpringApplication* which is the primary Spring component. [6]

Figure 2.4 presents the class responsible by to handle incoming web requests, an important feature for web applications. While *@RestController* annotation defines that as a web request handler, *@RequestMapping* annotation maps all requests to the "/" address will be handled by the index method. We can see the few lines of code and low complexity, mainly in the class *HelloController.java*. We can assess the low complexity through a few numbers of linearly-independent paths. The combination of platform features with small and simple code can be useful to newcomers to understand the platform.

**Code samples should provide working environment** [59]. Code samples are formed by source code as well as other configuration files needed to run them properly. Automated build and integration tools may also support both creators and clients, improving quality and reducing risks [22, 61, 104]. In Figure 2.2 we can see files as *build.gradle* and *setting.gradle*, *pom.xm* and *.travis.yml*. The files *build.gradle* and *setting.gradle* are from *Gradle Build Tool*, which is an open-source build automation tool focused on flexibility and performance, and uses scripts written with Groovy or Kotlin DSL. [7]. The file *pom.xml* comes from *Apache Maven*, which is a software project management and comprehension tool, based on the project object model (POM). Maven can manage a project's build, reporting, and documenting from a central piece of

[5] https://github.com/spring-guides/gs-spring-boot

[6] https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#getting-started.first-application.code.main-method

[7] https://docs.gradle.org/current/userguide/userguide.html

```
│ complete/
├── build.gradle
├── gradle/wrapper/
│   ├── gradle-wrapper.jar
│   └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── mvnw
├── mvnw.cmd
├── pom.xml
├── settings.gradle
├── src/main/java/com/example/springboot/
│   ├── Application.java
│   └── HelloController.java
└── test/java/com/example/springboot/
    ├── HelloControllerIT.java
    └── HelloControllerTest.java
initial/
├── build.gradle
├── gradle/wrapper/
│   ├── gradle-wrapper.jar
│   └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
├── mvnw
├── mvnw.cmd
├── pom.xml
├── settings.gradle
└── src/main/java/com/example/springboot/
    ├── Application.java
    └── HelloController.java
test/
└── run.sh
.gitignore
.travis.yml
CONTRIBUTING.adoc
LICENSE.txt
LICENSE.writing.txt
README.adoc
```

Figure 2.2: Folders and files from *gs-spring-boot* in 2022.

information [8]. The file *.travis.yml* is from Travis CI, which is a Continuous Integration / Continuous Delivery (CI/CD) platform that enables developers to quickly and easily build, test, and deploy code [9].

**Code samples should evolve and keep up to date, otherwise they become**

[8] https://maven.apache.org/
[9] https://www.travis-ci.com/about-us/

```java
//Application.java
@SpringBootApplication
public class Application {

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

  @Bean
  public CommandLineRunner commandLineRunner(ApplicationContext ctx) {
    return args -> {

      System.out.println("Let's inspect the beans provided by Spring
         Boot:");

      String[] beanNames = ctx.getBeanDefinitionNames();
      Arrays.sort(beanNames);
      for (String beanName : beanNames) {
        System.out.println(beanName);
      }
    };
  }
}
```

Figure 2.3: Application.java from code sample *gs-spring-guides* in 2022

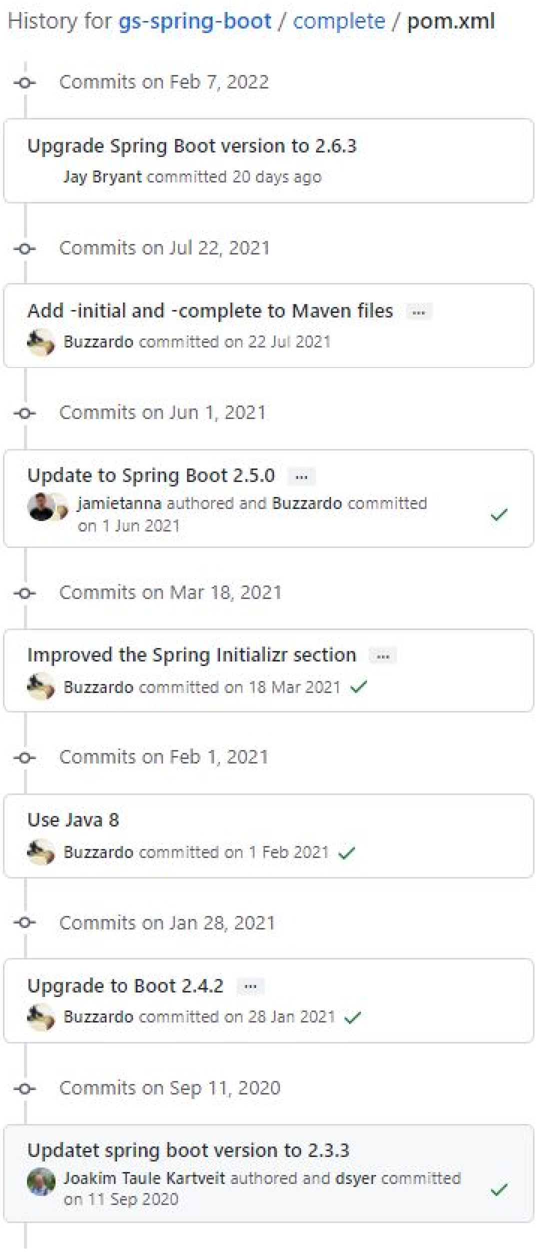
```java
//HelloController.java
@RestController
public class HelloController {

  @RequestMapping("/")
  public String index() {
    return "Greetings from Spring Boot!";
  }
}
```

Figure 2.4: HelloController.java from code sample *gs-spring-guides* in 2022

**outdated and less attractive to their clients** [35, 46, 55, 59, 107]. Figure 2.5 presents the last commits performed into file *pom.xml* for *gs-spring-boot*. We can see four Spring Boot updates and we also see an update to Java 8. We see *gs-spring-boot* keep up to date with new Spring Boot versions as well as other technologies.

## 2.4 Selected Platforms and Organizations

As previously stated, many organizations rely on code samples to help their clients to deal with platform features [2, 18, 30, 31, 62, 63, 94, 101]. In this section, we present information about the platforms and organizations that were used in the studies presented in this work.


History for gs-spring-boot / complete / pom.xml

Commits on Feb 7, 2022

Upgrade Spring Boot version to 2.6.3
Jay Bryant committed 20 days ago

Commits on Jul 22, 2021

Add -initial and -complete to Maven files ...
Buzzardo committed on 22 Jul 2021

Commits on Jun 1, 2021

Update to Spring Boot 2.5.0 ...
jamietanna authored and Buzzardo committed on 1 Jun 2021 ✓

Commits on Mar 18, 2021

Improved the Spring Initializr section ...
Buzzardo committed on 18 Mar 2021 ✓

Commits on Feb 1, 2021

Use Java 8
Buzzardo committed on 1 Feb 2021 ✓

Commits on Jan 28, 2021

Upgrade to Boot 2.4.2 ...
Buzzardo committed on 28 Jan 2021 ✓

Commits on Sep 11, 2020

Updatet spring boot version to 2.3.3
Joakim Taule Kartveit authored and dsyer committed on 11 Sep 2020 ✓

Figure 2.5: Commits that edited *pom.xml* into *gs-spring-boot*.

***Android:*** The Android platform[10] allows the creation of Android apps for several devices, such as smartphones, smartwatches, and TVs. Android code samples are publicly available on GitHub[11] and help clients deal with Android features, such as permissions, pictures, and video manipulation, background tasks, notifications, networks, multiple touch events, among many others.

***AWS:*** Amazon Web Services (AWS) is the world's most adopted and most comprehensive cloud platform. Millions of customers, including the fastest-growing startups, large enterprises, and the largest government agencies are using AWS to lower their costs, become more agile, and innovate faster. [12]

[10]https://developer.android.com/guide/platform
[11]https://github.com/googlesamples
[12]https://aws.amazon.com/what-is-aws/

***Azure:*** The Azure cloud platform consists of more than 200 cloud products and services designed to help clients bring new solutions to life to solve today's challenges and create the future. Build, run and manage multi-cloud, on-premises, and edge applications with the tools and frameworks of clients' choice. [13]

***Spring Boot***: Spring Boot is the world's most popular Java platform [14] aiming to make it easy to create stand-alone, production-grade Spring based Applications. [15] The Spring Boot platform belongs to the Spring ecosystem. Spring Boot provides an embedded web server and also gives a base set of dependencies to simplify build configuration. Furthermore, Spring Boot provides production-ready features such as metrics, health checks, and externalized configuration.

***Motivation***: We select these platforms due to several reasons. *First*, they are relevant and worldwide adopted by millions of clients. *Second*, they support the creation of distinct and important niches of apps: mobile, web, and cloud computing. *Third*, their base of code samples is publicly available on GitHub. *Fourth*, their code samples are written in Java.

[13]https://azure.microsoft.com/pt-br/overview/what-is-azure/
[14]https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/
[15]https://spring.io/projects/spring-boot

Chapter 3

# Code Samples vs Conventional Projects

**Context:** By definition, code samples have different purposes when compared to conventional projects. While conventional projects are mainly interested in the correct execution of implemented functionalities, code samples aim particularly at characteristics such as code quality, platform version up to date, and reusability. Although their interests differ, we can find both similar and different aspects. On the one hand, as code samples are made for educational purposes, they should be smaller than conventional projects in terms of source code. Likewise, code samples should be simpler and easier to understand than conventional projects. But on the other hand, just like conventional projects, code samples also evolve over time [59] and according to Lehman's laws [48] of software evolution, they should get bigger and more complex over time.

**Problem:** Since conventional projects are well-known, there are plenty of studies targeting them [13, 28, 111, 113]. To the best of our knowledge, there are no studies comparing code sample source code and evolution aspects with conventional project source code and evolution aspects. In the same way, there are no studies comparing code sample usage against conventional project usage. In our previous work [59], we analyzed 233 code samples from two platforms: Android and Spring Boot. We assessed aspects related to source code and evolution, popularity, and client usage. As the result of this work, we generated a set of implications. Among them, we have that: (1) code samples should be simple and small; (2) code samples should keep up to date with the platform versions; (3) code samples are infrequently used through the fork approach. From these implications, some questions arise. For example, are code samples simpler and smaller than conventional projects? Are code samples updated to a new platform version, faster than conventional projects? What is the difference between code samples and conventional projects in terms of evolution? Code samples are used as conventional projects? Answering these questions, we first, fill the lack of knowledge about similarities and differences between code samples and conventional projects. Second, we can help organizations deal with potential code sample problems. For example, in the case of code samples being similar to conventional projects, one can use well-known conventional project solutions for code sample problems. Otherwise, we need to find new alternatives to solve problems already explored in conventional projects in the context of code samples.

**Purpose:** In this chapter, we aim to achieve the following goal: Explore the differences and similarities between code samples and conventional projects, comparing structural evolutionary and usage characteristics, to increase the knowledge about code samples. To do so, we conducted studies on top of already found results of code samples [59] by assessing the characteristics of conventional projects and compare them with code samples characteristics. To do so, we defined three research questions: (RQ1) What are the source code characteristics of code samples in comparison to conventional projects? (RQ2) How do code samples evolve over time in comparison to conventional projects? (RQ3) How are code samples used by clients compared to conventional projects?

**Structure:** The rationale of research questions and study design are presented into Section 3.1. Section 3.2 presents the results obtained in the exploratory study. Section 3.3 presents the implications of the results. Section 3.4 shows the threats to validity and how we mitigated them. Section 3.5 describes related work and their differences when compared to our work. Finally, Section 3.6 presents the conclusion.

## *3.1 Study Design*

In this section, we present the steps taken to answer our research questions. Section 3.1.1 presents selection method for conventional projects. Section 3.1.2, Section 3.1.3 and Section 3.1.4 present the research questions, their rationale and study design. Finally, Section 3.1.5 presents the study design to compare code samples to conventional projects.

### 3.1.1 Conventional Projects Selection

The set of code samples and their results was obtained from our previous study [59]. We used 233 Java code samples, 176 from the Android platform and 57 from Spring Boot. Details about each platform and the motivation that led us to select them were presented in Section 2.4. To compare the characteristics of code samples to conventional projects, we need to select a set of conventional projects. To do so, we take three steps. *First*, to select relevant conventional projects, we selected GitHub's top 5,000 Java repositories sorted by the number of stars. We focus on Java because the code samples were in Java language as well. *Second*, from the set of 5,000 repositories already selected, we exclude all repositories that were not Android or Spring Boot projects. We classified a repository as an Android project if there is at least one Java file importing an Android library. In a similar way, we classified a repository as a Spring Boot project if there is at least one import to a Spring Boot library into any Java file. *Third*, we selected, randomly, 176 Android conventional projects and 57 for Spring Boot. These numbers of projects were chosen to keep the same proportion of code samples in terms of Android and Spring Boot projects.

### 3.1.2 (RQ1) What are the source code characteristics of code samples in comparison to conventional projects?

In this research question, we assess the last version of the source code from conventional projects (35,697 Java files) and extract three data: source code metrics, file extensions, and configuration files, as summarized in Figure 3.1.

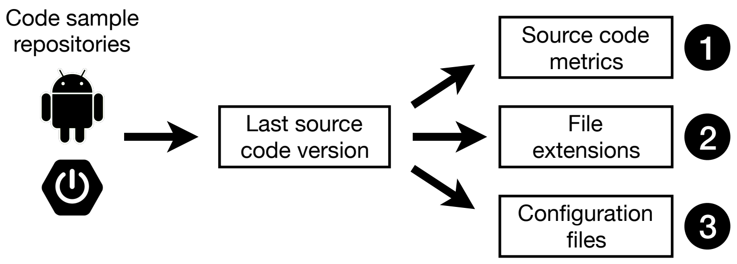


Figure 3.1: Source code analysis (RQ1).

1. Source code metrics: We first assess the current state of the conventional projects by computing source code metrics with the support of the software analysis tool Understand.[1] 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 [54]. Code samples are not different; ideally, they should be concise and

[1]https://scitools.com

simple [20, 59]. This means that code samples need to be simpler than conventional projects. In addition, code comment is important to any piece of code [49]. However, it may be even more relevant to samples than to conventional projects, as they provide inline comments to help their clients.

2. File extensions: We extract the file extensions found in conventional projects, to compare to code samples, for a better understanding of their content in addition to source code files. Rationale: In our previous study [59], we found that the most common extension files in code samples are *xml*, *Java*, *jar*, *md*, *json*, *properties* and *adoc*. To understand whether the presence of this files is a characteristic of code sample or a characteristic of any project from this platforms (Android or Spring Boot), we need to extract the file extensions from conventional projects and compare to code sample projects.

3. Configuration files: In addition to the file extensions, we also compute the most common configuration files from conventional projects. In our previous study [59], we found that code samples are made available with the support of automated build and integration tools. Rationale: With a similar result between code samples and conventional projects, we could conclude that the 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 [22, 61, 104].

### 3.1.3 (RQ2) How do code samples evolve over time in comparison to conventional projects?

In this research question, we assess all versions (*i.e.* 96.867 commits) of conventional projects and extract: evolutionary metrics, file extension changes, configuration file changes, and migration delay, as presented in Figure 3.2.

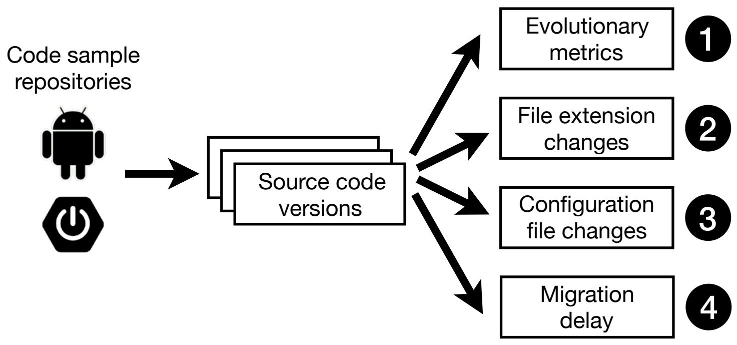


Figure 3.2: Evolutionary analysis (RQ2).

1. Evolutionary metrics: We compute metrics to assess the evolution of conventional projects to compare with code samples. Specifically, we extract two evolutionary metrics: frequency of commits and lifetime. Lifetime is computed

as the number of days between the first and the last project commit. Rationale: In our previous study [59], we found that code samples are updated over time. Since we do not have a threshold to guide if the frequency of code samples is high or low, we need to extract and compare them with conventional projects. In addition, the frequency of changes can be related to the platform updates.

2. File extension changes: We analyze the file extension changes over time to better understand how conventional projects of Android and Spring Boot are actually maintained and compare them to code samples. Rationale: In our previous study, [59], we assessed how code samples evolve considering file extension changes. To better understand that, we need to know how conventional projects of these platforms are maintained and compare them to understand if the way that code sample evolved is a characteristic of the code sample itself or is a characteristic of projects using these central platforms.

3. Configuration file changes: We analyze the modifications in the configuration files of conventional projects to assess and compare whether the automation tools are being updated as in the code sample. 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. In our previous study, [59] we assessed the configuration file changes in code samples. To better understand the result obtained, we also need to explore how conventional projects from the same central platforms.

4. Migration delay: We compute the migration delay between projects and their platforms. In other words, we assess how long it takes for conventional projects to migrate to new platform versions and compare them with code samples. Rationale: As client projects, code samples and conventional projects are dependent on their platforms. When these platforms evolve and provide new versions, the code samples (as any other platform client project) should be updated. Otherwise, they will be frozen on past versions and become less attractive to their clients [35, 46, 55, 107]. As a result of our previous study [59], we found that code samples change over time to keep up to date with new platform versions. However, since code samples are projects used for educational purposes, it is important to be updated as fast as possible. We need to extract migration delay from conventional projects and compare them with code samples to better understand the code sample results.

### 3.1.4 (RQ3) How are code samples used by clients compared to conventional projects?

In this research question, we aim to assess the code sample usage by computing two metrics: number of forks and number of ahead forks. We assess

60,142 forks from conventional projects. An ahead fork is a fork that received at least one commit after being forked. Rationale: Fork can be seen as a measure of popularity [7]. After forking, the client can update the code or simply not perform any change. If the forked project is updated, this may indicate that the client is somehow exploring the code sample, possibly, by running and improving it.

### 3.1.5 Comparative Analysis

After extracting the metrics from the conventional projects that were described earlier, we need to compare them with the metrics from code samples extracted in our previous study [59] We know that only an absolute comparison with metrics maybe not be enough. So, to get more confident in our results, we decided to apply statistical tests. We choose the Mann-Whitney test [89] to assess the difference between code sample and conventional projects metrics and using the 5% confident level (i.e., *p-value* < 0.05). The Mann-Whitney test does not assume a normal distribution since it is a non-parametric statistical. Also, this test assesses if two independent distributions have equally large values [111]. To show the effect size of the difference between the metrics, we compute Cliff's Delta (or *d*). Following previous guidelines [83], 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.

## 3.2 Results

In this section, we present the results obtained through the steps detailed in Section 3.1. We divided this section into subsections for each research question. Sections 3.2.1 shows results for RQ1, Section 3.2.2 for RQ2 and Section 3.2.3 for RQ3.

### 3.2.1 (RQ1) What are the source code characteristics of code samples in comparison to conventional projects?

Source code metrics: When we compare code samples to conventional projects, the numbers confirm our initial impression that code samples are overall smaller and simpler than conventional projects. Table 3.1 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 platforms, code samples are smaller than conventional projects in terms of the number of Java files (direction: ⇃). Another metric in which both platforms agree is cyclomatic complexity: code samples have statistically significant less complexity (Android: * small effect, direction: ↓; Spring Boot: *** large effect, direction: ↓) than conventional projects.

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

Table 3.1: 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 | * | ↓ | *** | ↓ |

File extensions: Table 3.2 shows the file extensions found in code samples and 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 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.

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 being the majority in Spring Boot samples, the low percentage, when compared to conventional projects, restates that (i) code samples provide a complete working environment and (ii) code samples are simple and small.

Table 3.2: 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 |

Configuration files: Table 3.3 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: Code samples are overall simpler and smaller than conventional projects. We also find that code samples, as conventional projects, rely on tools to automate build and integration (e.g., Gradle, Maven, and Travis) and provide a working environment to the clients (*i.e.* including jar, xml, properties, and other files in addition to code).

Table 3.3: 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 |

## 3.2.2 (RQ2) How do code samples evolve over time in comparison to conventional projects?

When we compare code samples with conventional projects (Table 3.4), 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 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 platforms. 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 3.4: 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 3.5 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.

Regarding comparing code samples and conventional projects, it is interesting to notice that Java files and XML files hold the first two places in con-

Table 3.5: 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 |

ventional projects (Table 3.5). 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 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 platform version. But not exclusively, and careless evolution may impact on clients' understanding of it and, consequently, on its usefulness.

Table 3.6 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 are 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 platform 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 platform version, thus making file modifications more common.

Table 3.6: 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 |

Configuration file changes: Table 3.7 presents the most changed configuration files. We notice that `build.gradle` files are the most changed in both platforms. 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.

Table 3.7: 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 |

Most of the code sample behavior previously presented is also observed in conventional projects in configuration file changes (Table 3.7). We highlight that `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: Finally, to compare the migration delay to new platform 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 platform versions when comparing code samples to conventional projects (***: large effect, direction: ↓) in both Android and Spring Boot (Table 3.4). In other words, we show that code samples update faster to new platform 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 platform itself may also maintain its code samples.

Lesson Learned 2: Like conventional projects, code samples also evolve. However, code samples are changed less frequently. Despite this, code samples are updated more quickly for new versions of the platform. Also, while changes in conventional projects are mostly done in source code, in code samples they happen more in configuration files.

### 3.2.3 *(RQ3) How are code samples used by clients compared to conventional projects?*

We adopt the fork metric as a proxy of client usage for the code samples. Table 3.8 shows the comparison between code samples and conventional projects. In terms of the number of forks, we can note that for both cases code samples presented less forks than conventional projects. For Android, we have a small effect (*) with ↓ direction, and for Spring Boot we have a large effect (***) with ↓ direction. This may be a consequence of the way we select conventional projects. We selected the 5,000 Java projects with the most stars. Projects with more stars tend to have more forks as they are more popular.

Table 3.8: 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 | - | - | *** | ↑ |

In terms of ahead forks, 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.

Lesson Learned 3: The approach of using code samples via fork has low usage by clients compared to conventional projects. Despite this, there are indications that code sample forks may receive more updates than conventional project forks.

## 3.3 Implication

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

Be a simple kind of project: code samples are simpler and smaller than conventional projects to facilitate the understanding and readability. Indeed, the majority of the code samples provided by Android and Spring Boot follow this rule.

Keep the environment pleasant: As with conventional projects, code samples provide a working environment to facilitate CI/CD activities. For code samples, even more frequent than changes in source code files, changes in configuration files are important to keep the code sample updated for platform versions and other dependencies. In addition, to manage build, compile, testing, deployments, monitoring, and other activities.

Quick on the draw: code samples should be updated to newer versions of the platform more quickly than conventional projects. Given the educational purpose of code samples, they should always be updated as they are one of the reliable sources of how to use the platform's features. Otherwise, they may become out of date and useless for your purpose. On the other hand, conventional projects are more complex and upgrading to a new platform version may impact the need for many changes.

Don't fork up: when evaluating the use of code samples, we were able to verify that the fork approach is not popularly adopted by clients. To better understand how code samples are used, it is necessary to go depth on other ways to use them. Like the repository download approach or the copy and paste approach.

## 3.4 Threats to Validity

This section discusses the study limitations based on the four categories of validity threats described by Wohlin et al. [106]. 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.

*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 Android and Spring Boot platforms. 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.

## 3.5 Related Work

Platforms are used to support development, provide source code reuse, improve productivity, and decrease costs [44, 68, 80]. Often there is a steep learning curve involved when developers adopt platforms. Development based on code samples provides the benefits of code reuse, efficient development, and code quality [91]. Moreover, with the popularity and relevance of the Question and Answer (Q&A) sites such as Stack Overflow, some studies propose approaches and tools to search and/or retrieve source code samples and explore the 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 [34, 50, 78, 85, 91]. For instance, Holmes and Murphy [34] proposed a technique that recommends source code examples from a repository by matching structures of given code. FuzzyCatch [72] provides a code recommendation tool, based on fuzzy logic, for handling exceptions. XSnippet [85] provides a context-sensitive code assistant platform that provides sample source code snippets for developers. In general, these tools help locate samples of code, demonstrate the use of platform, and fasten development by exploring the syntactic context provided mainly by the IDE to recommend code samples more relevant to developers (as in Strathcona [34]). 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 platforms. Therefore, it is expected that the types of code samples explored in this work 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 platform APIs, and proposed approaches to mine API usage examples from open code repositories and search engines [4, 16, 42, 64, 65, 71, 73, 102, 116]. Most of these works retrieve the so-called code snippets to support API learning, whereas our work focuses on complete projects of platform 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. [90] 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. [108] 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.* [23] 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.* [56] evaluate the quality of code snippets on Stack Overflow, exploring aspects such as reliability and conformance to programming rules, readability, performance, and security. Finally, studies are analyzing the adoption of code snippets [33, 84, 109]. For instance, Roy and Cordy [84] 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 in real projects. However, our work targets entire code sample projects instead of code snippets.

## 3.6 Conclusion

In this chapter, we proposed an extension of our previous study [59] to compare and better understand code samples against the conventional projects from the same central platform. By assessing 233 conventional projects related to Android and Spring Boot platforms, we investigated aspects related to their source code, evolution, and usage. We found that most code samples are smaller and simpler than conventional projects, both of them provide a working environment, and rely on automated build tools. They frequently change, for example, to adapt to new platform versions, but not exclusively, making it necessary to carry out a more in-depth study on how code samples are evolved and how this impacts clients. Also, the fork approach is an unusual way that clients use code samples. It is interesting to delve into other approaches to the use of code samples.

CHAPTER 4

# Usage of Code Samples

**Context:** Since the relation between clients, organization, and the platform forms a SECO, code samples are also included in this context. Due to the barriers that can turn difficult the learning process of the platform's features, code samples can be seen as an alternative to mitigate these barriers and help clients to learn and use platform features and interact with the SECO.

**Problem:** Besides that, there is small knowledge of how clients use code samples. In our previous study [59], we assess code sample usage through the fork approach and we found that, in general, a few clients use forks on code samples and even fewer clients perform changes on these forks and send them to GitHub. In Chapter 3, we noted that the fork usage is fewer in code samples than in conventional projects. However, there are some questions unanswered yet, such as what are the most common doubts raised from code sample usage? What are the most common needs or problems faced by clients on code sample usage? Beyond the fork approach, how do clients use code samples? Do they copy and paste the code instead of forking? In this way, knowing how clients use code samples, can help to build code samples that are more adaptable to clients' expectations and needs. In addition, the knowledge about the most common troubles faced by clients when they use code samples can help organizations to create solutions and avoid client frustration.

**Purpose:** To explore and better understand how clients use code samples, we create our fist research question: (RQ1) How do clients use code samples in their own projects? In addition, to assess the most common problems and needs of code sample clients, we create our second and third research questions: (RQ2) What are the most common questions about code samples on Stack Overflow? (RQ3) What are the most common issues that impact code samples on GitHub?

**Structure:** Section 4.1 shows methods and steps used to conduct the study. Section 4.2 shows the results obtained after we performed steps of Section 4.1. Section 4.3 presents the implications of results. Section 4.4 shows the threats to validity of this study and how we mitigated them. Section 4.5 presents the related work and differences to our study. Finally, Section 4.6 presents the conclusion of this study.

## 4.1 Study Design

As in Chapter 3, in this chapter we focused only in two platforms: Android and Spring Boot (details about each platform were presented in Section 2.4). Since we are interested in assessing long-term aspects of code sample usage, we selected platforms with more aged code samples. In this way, we have more historical data from both Stack Overflow and GitHub. Finally, we got 233 to be analyzed, being 176 from Android and 57 from Spring Boot.

### 4.1.1 (RQ1) How do clients use code samples in their own projects?

In this research question, we aim to explore code sample usage through source code analysis. In our previous study [59] and in Chapter 3, we assess code sample usage and compare with conventional projects usage through the fork approach. We found that fork approach has low usage on code samples comparing with conventional projects. In this research question, we want to assess the code sample usage through the copy/paste approach.

Since it is not viable to look into the entire GitHub repository, for code sample usage, we use a set of criteria to select our sample. We selected only repositories from watchers of code samples. When users start to "watch" a repository, they receive notifications on new discussions, as well as events about this repository in their "GitHub activity feed [1]". In that way, we choose only repositories from users interested in code sample changes. As another selection criteria, we exclude all repositories that are *forks*, because we already look for forks in our previous study.

After the repository selection, we need to choose which approach to assess code sample usage. To do so, we choose to explore duplicate code through copy/paste detector tools. For that task, we used the CPD library provided by the PMD tool. The PMD is an open-source tool and provides, besides others functionalities, a Copy/Paste Detector (CPD) tool for finding duplicate code.

[1] https://docs.github.com/en/rest/reference/activity

CPD uses the Karp-Rabin string matching algorithm. [2] This tool scans the files themselves for duplicate code, also it is successful in returning similar code across different files [58]. As next step, we ran the CPD library for each pair: *code sample* and *watcher's repository* of the code sample itself.

### *4.1.2 (RQ2) What are the most common questions about code samples on Stack Overflow?*

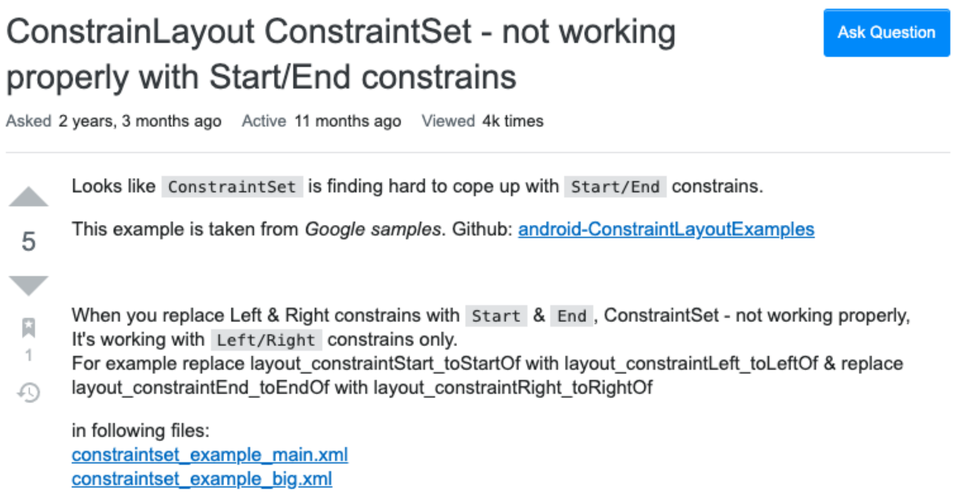
Code samples are created to support clients dealing with platform features. In addition, the literature shows that code samples are important to support learning [82]. Even more, a lack of code samples can be a barrier to understand central platforms [112]. Nonetheless, some aspects could make it difficult for clients to understand and use, such as an increase in complexity and size, and a decrease in readability. So, this research question aims to explore the most common problems faced by clients regarding code sample usage. Rationale: identifying these problems, first, fill the void of knowledge about it in the state-of-the-art and second, it will help organizations that develop code samples to become aware of these problems and also motivate them to create solutions to mitigate them.

To answer this research question we conduct a qualitative study, assessing Stack Overflow questions related to code samples. 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.[3] Questions on Stack Overflow can receive answers, and the community is responsible for evaluating the quality of the proposed answers, giving positive or negative votes.

Figure 4.1 presents an example of a Stack Overflow question[4] about the Android sample `android-ConstraintLayoutExample`.[5] In this case, the client 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.

We use the dataset provided by Stack Exchange[6] 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

[2] https://pmd.github.io/latest/pmd_userdocs_cpd.html
[3] https://stackoverflow.com/company
[4] https://stackoverflow.com/questions/49232559
[5] https://github.com/googlearchive/android-ConstraintLayoutExamples
[6] https://data.stackexchange.com/help


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

process resulted in 527 questions for Android and 87 questions for Spring, totaling 614. Figure 4.2 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.

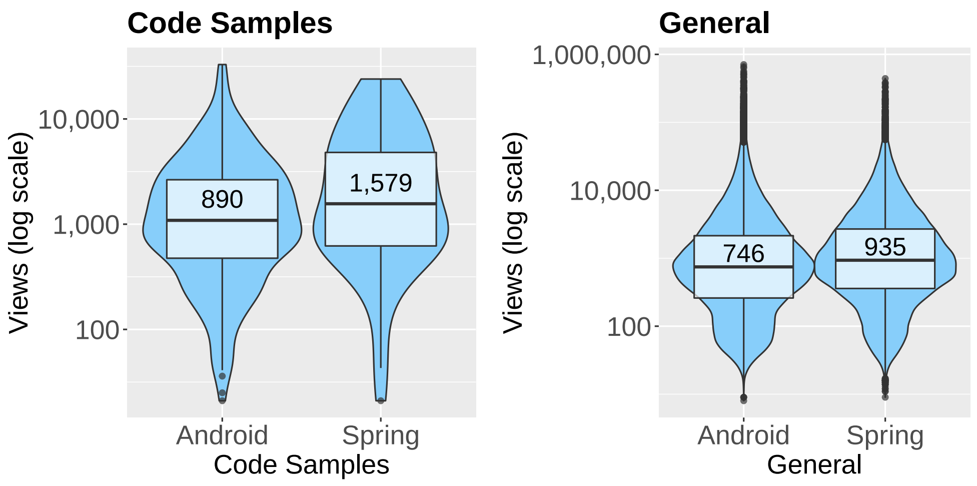


Figure 4.2: Distribution of the number of views of the selected Android and Spring questions.

After collecting the posts on Stack Overflow, we perform a manual classification. We analyzed the questions using thematic analysis [21], 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, and (4) reviewing the themes to find opportunities for merging, and (5) defining and naming the final themes [12]. Steps 1 to 4 were performed independently by two authors of this paper. We used the Cohen Kappa test [19] 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).

Our manual classification leads to four categories of questions: Importing: questions in which clients are trying to import the code sample to use or modify it but could not due to configuration issues. Running: questions in which clients are trying to run the code sample but could not due to runtime problems. Modification: questions in which clients 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.

### 4.1.3 (RQ3) What are the most common issues that impact code samples on GitHub?

While RQ2 aims to explore Stack Overflow questions, here we are interested in GitHub Issues. We assess GitHub issues to explore common needs related to code samples. In this analysis, we aim to explore common issues that impact code samples. In other words, we want to assess issues that led to code sample changes. With that, we can answer questions like: Do clients need to change code samples? Moreover, we aim to explore whether the clients' questions (and other types of interactions) are different from Stack Overflow ones.

To conduct this study, the *first* step was to select all issues for the studied code samples. On the *second* step, we removed all open issues, since they do not cause code sample changes yet. When an issue causes modification in a repository, it is common to reference the commit or pull request with the modifications. The *third* step was to exclude issues without reference to commits or pull requests, only keeping the ones that cause modification. In the *fourth* step, we manually removed false positive references. Based on that, we found 269 GitHub issues from code sample repositories.

After the issue selection, we performed a manual classification of each one. Based on the title, body, and comments of the issues, we classify them in the following categories: (i) *importing*, when clients try to import the code sample to use or modify it but could not due to configuration issues; (ii) *running*, when clients try to run the code sample but could not due to run-time problems; (iii) *modification*, when clients try to modify the code sample and faced some trouble; (iv) *improvement*, when clients suggest an improvement into the code sample or their comments led code sample' maintainers to improve them; and (v) *question*, when clients are simply asking about code sample usage or better patterns.[7]

Furthermore, we look at the changed files in the commits related to the issues. 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 [21].

## 4.2 Results

In this section, we present the results of the study for each research question. Section 4.2.1 presents the results about code sample usage (RQ1). Section 4.2.2 shows the results related to common issues on GitHub (RQ2). Finally, Section 4.2.3 explain the results of most common Stack Overflow questions.

### 4.2.1 (RQ1) How do clients use code samples in their own projects?

Table 4.1 shows the result about the number of code sample watchers. From the 233 analyzed code samples we have a total of 20,197 watchers, 16,232 belong to Android samples, and 3,965 to Spring Boot samples. For the Android platform, we found that only 515 from 16,232 (3.2%) watchers use code sample code inside their own GitHub projects. In the same way, for the Spring Boot platform, we found that only 43 from 3,965 (1.1%) of the watchers use code sample code in their own GitHub projects. Table 4.1 also presents another view of code sample usage, it also presents the number of watchers' repositories. We found 145,064 repositories, 115,283 from Android watchers, and 29,781 from Spring Boot watchers. In the Android platform, from 115,283 only 638 (0.6%) of repositories from watchers contain code sample code. For the Spring Boot platform we found from 29,781 only 48 (0.02%) of repositories from watchers present code sample code.

[7]The first three categories come from our prior analysis on Stack Overflow.

Table 4.1: Number of watchers.

| | Android | | Spring | |
|---|---|---|---|---|
| Metric | n° | % | n° | % |
| Watchers with usage | 515 | 3.2% | 43 | 1.1% |
| Watchers without usage | 15,717 | 96.8% | 3,922 | 98.9% |
| Repositories with usage | 638 | 0.6% | 48 | 0.02% |
| Repositories without usage | 114,645 | 99.4% | 29,733 | 99.8% |

There are some possible reasons that can explain the low usage of code samples snippets within watcher's projects. For example, given that code samples are simple and small projects, which help in learning the platform's functionalities. The use of code samples may demands adaption to meet complex scenarios often found in real projects. In this case, the copy/paste approach is not enough.

Table 4.2 presents the most common code samples found into watcher's repositories. In the Android context, the most common code sample is *android-Camera2Basic*, which is present in 136 repositories (21.3%), and it is related to the usage of Camera2 API to capture images. [8]. The second most common is *android-vision* and this sample demonstrates the usage of features from vision API for detecting faces and barcodes [9] The third most common sample is *android-BluetoothLeGatt*, presented into 43 repositories (6.7%) and this sample demonstrates how to use the Bluetooth LE Generic Attribute Profile (GATT) to transmit arbitrary data between devices [10]. The fourth is the sample *android-BluetoothChat*, we found it is present into 35 repositories (5.4%) and this sample shows how to implement two-way text chat over Bluetooth between two Android devices, using all the fundamental Bluetooth API capabilities [11]. The fifth sample is *android-Camera2Video*, presented into 32 repositories representing 5% and helps clients to record video using the new Camera2 API in Android [12]. The others code samples were presented in 286 (45%) repositories.

Also in Table 4.2, we have the most common code samples of Spring Boot platform. The most common sample is *gs-uploading-files* found in 12 (25%) projects. This code sample helps Spring Boot's clients to deal with the process of creating a server application that can receive HTTP multi-part file up-

[8] https://github.com/android/camera-samples/tree/main/Camera2Basic

[9] https://github.com/googlesamples/android-vision

[10] https://github.com/googlearchive/android-BluetoothLeGatt/blob/559163eed3fbd777df0d9a1dfb2b792c827b9528/README.md

[11] https://github.com/googlearchive/android-BluetoothChat/tree/62adaa391f4d6714172451b42f6f665f39fbe7bb

[12] https://github.com/googlearchive/android-Camera2Video/tree/c5029e62a2c52f39c43c503ebebd67fbf861e74e

loads [13]. The second most common is *tut-rest*, found in 9 (18.8%) repositories. This code sample teaches how to build a RESTful services with Spring [14]. The third most common code sample *gs-producing-web-service*, found in 5 repositories (10.4%), helps clients to create a SOAP-based web service server with Spring [15]. The fourth sample is *gs-rest-service*, found in 5 repositories (10.4%). This code sample helps Spring Boot's clients on Building a RESTful Web Service with Spring. The fifth code sample is *gs-accessing-data-mongodb*, found in 5 repositories (10.4%), assisting on learning how to persist data in MongoDB [16] The others samples are present in 12 (25%) repositories.

Table 4.2: Most common code samples used into watchers' repositories.

| Android | | | Spring | | |
|---|---|---|---|---|---|
| Code Sample | n° | % | Code Sample | n° | % |
| android-Camera2Basic | 136 | 21.3% | gs-uploading-files | 12 | 25% |
| android-vision | 106 | 16.6% | tut-rest | 9 | 18.8% |
| android-BluetoothLeGatt | 43 | 6.7% | gs-producing-web-service | 5 | 10.4% |
| android-BluetoothChat | 35 | 5.4% | gs-rest-service | 5 | 10.4% |
| android-Camera2Video | 32 | 5% | gs-accessing-data-mongodb | 5 | 10.4% |
| Others | 286 | 45% | Others | 12 | 25% |
| Total | 638 | 100% | Total | 48 | 100% |

Despite the low frequency use of code sample source code in its watchers repositories as presented in Table 4.1, we found the use of code samples in relevant repositories to the community. Table 4.3 present a set of relevant watchers' repositories. For instance, the repository *SimplifyReader* from watcher *chentao0707* presents 1,655 forks, and more than 4,500 stars and use 118 from Android's code sample *SwipeRefreshMultipleViews*. Also, the repository *SamrtTubeNext* developed by *yuliskov* have 411 forks, more than 5,000 stars and have a huge code sample usage with 901 lines from Android' code sample *MediaBrowserService*.

Table 4.3: Relevant watcher's repositories

| Watcher | Repository | Forks | Stars | Duplicate Lines | Code Samples Usage |
|---|---|---|---|---|---|
| chentao0707 | SimplifyReader | 1,655 | 4,576 | 118 | SwipeRefreshMultipleViews |
| yuliskov | SmartTubeNext | 411 | 5,176 | 901 | MediaBrowserService |

Also about code sample usage, Figure 4.3 presents the number of duplicate code lines and the percentage of duplicate code lines for watcher's repositories.

[13] https://github.com/spring-guides/gs-uploading-files
[14] https://github.com/spring-guides/tut-rest
[15] https://github.com/spring-guides/gs-producing-web-service
[16] https://github.com/spring-guides/gs-accessing-data-mongodb

In an absolute analysis, we can see that Android has more variation between the usage. There are repositories with few lines until repositories with more than 5,000 lines, and the median is 210.5 duplicate lines of code. In Spring Boot, there is a concentration of use of source code of code samples, with the median watchers using 36 lines of code. Looking at the relative view, also in Figure 4.3, Android has a high distribution but the median is only 4.45%, while in Spring Boot has 15,34% of relative duplicate code lines.

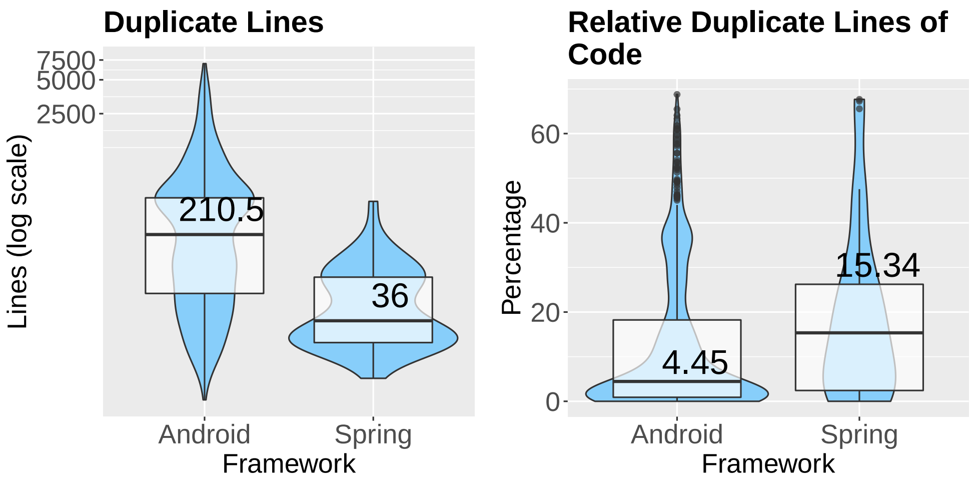


Figure 4.3: Number of Duplicate Lines of Code (left) and Relative Number of Duplicate Code Lines (right).

Lesson Learned 7: The code sample usage through copy and paste approach has low usage. Despite that, projects relevant to the community can benefit from its use

### 4.2.2 (RQ2) What are the most common questions about code samples on Stack Overflow?

Figure 4.4 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.

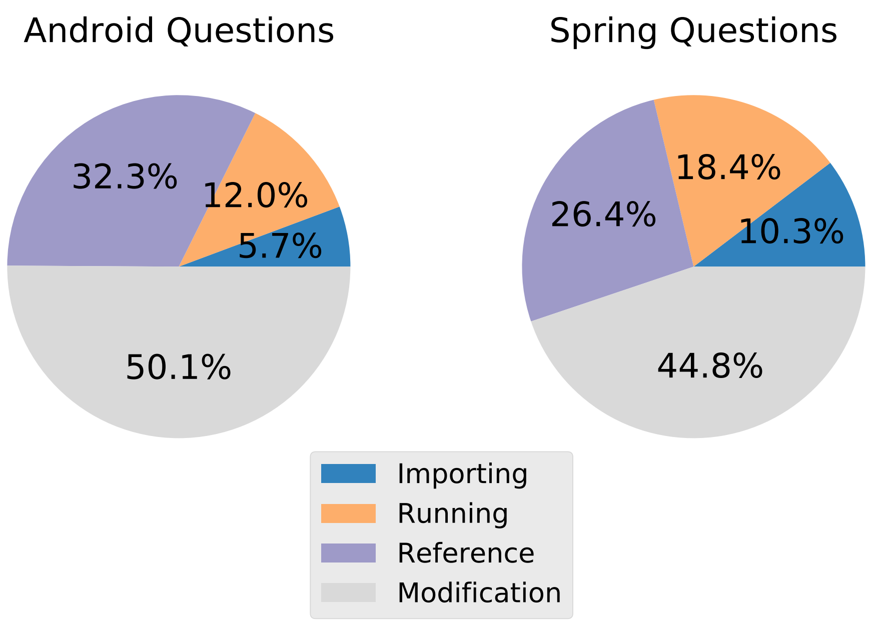


Figure 4.4: Stack Overflow questions by category.

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

**Importing.** This category includes questions in which clients 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 compiling, IDE is not configured, etc. For example, in the Question 1, the client cannot import and run the code sample in the Android Studio IDE.[17] The accepted answer simply presents step-by-step how the client should successfully import the code sample using that IDE.

Question 1: “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?”

Clients also have problems building the code samples, as in the following question, in which he is struggling with the gradle build.[18] The client itself later discovers that the wrong Java version was being used (Java 1.8 instead of 1.7).

[17] https://stackoverflow.com/questions/31188849
[18] https://stackoverflow.com/questions/31506508

Question 2: “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 Question 3, the client tries to use Maven for building the code sample and faces an error message.[19] The solution involves adding a new dependency to the maven repository (in the pom.xml file) and to upgrade maven to 3.0.5 above:

Question 3: “I am going through this guide: 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 [...]”

**Running.** This category includes questions in which clients are trying to run the code sample, but could not due to runtime problems. For instance, in Question 4, the client can run an Android code sample, however, it crashes in some specific cases.[20] The accepted answer states that this issue refers to a known bug, and there is no trivial solution to avoid it.

Question 4: “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 clients can run the code sample in the following two questions, but they face specific issues. In the first case, Question 5, the Android client[21] cannot run the code sample on some Android devices, whereas in the second case, Question 6, the Spring client[22] cannot kill the session.

[19]https://stackoverflow.com/questions/22935840
[20]https://stackoverflow.com/questions/43765499
[21]https://stackoverflow.com/questions/33763874
[22]https://stackoverflow.com/questions/37598036

Question 5: "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."

Question 6: "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 classification. It includes questions in which clients 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 clients perform minor changes in the code sample, however, the modifications do not behave as expected. In Question 7, the Spring modification resulted in an exception,[23] whereas in Question 8, the clients reports a deformed image in Android.[24] In both cases, the answers are trivial, and it seems that the clients do not have enough experience.

Question 7: "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."

Question 8: "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?"

Besides performing minor changes, clients may also create applications based on the code samples. In the following examples, Question 9 and Question 10, the clients are building custom cameras based on code samples provided by Android.[25][26]

[23] https://stackoverflow.com/questions/23366226
[24] https://stackoverflow.com/questions/34638651
[25] https://stackoverflow.com/questions/39044494
[26] https://stackoverflow.com/questions/39022845

Question 9: “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.”

Question 10: “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 client aims to expand the code sample considerably, taking into account security issues.[27]

Question 11: “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 the modification, we assess their tags. In Stack Overflow, a tag is a word or phrase that describes the topic of the question.[28] 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 4.4 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,[29] 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[30] and provides a framework for finding objects in photos and video as face, barcode, and text detection.[31] The next tag is setup (29), which is a merge of two other tags: ndk and studio.

[27]https://stackoverflow.com/questions/40355743
[28]https://stackoverflow.com/help/tagging
[29]https://developer.android.com/guide/topics/media/camera?hl=en_us
[30]https://developers.google.com/ml-kit
[31]https://developers.google.com/vision/introduction

The Android NDK[32] 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 official IDE for building Android apps.[33] The next tag is dagger (31), which is a dependency injection framework for Java, Kotlin, and Android.[34] 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.[35]

Table 4.4: 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.[36] The second tag is data (14): Spring Data's goal is to provide a familiar and consistent, Spring-based programming model for data access.[37] 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.[38] The next tag is cloud (4): Spring Cloud provides tools for clients to build some of the common patterns in distributed systems quickly.[39] 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.[40]

Overall, these results suggest that clients have issues modifying distinct types of code samples, as presented by the variation of detected tags. In both platforms, 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 Question 12, the client is simply illustrating his problem with

[32] https://developer.android.com/ndk
[33] https://developer.android.com/studio/intro
[34] https://dagger.dev
[35] https://developer.android.com/jetpack/guide#recommended-app-arch
[36] https://spring.io/projects/spring-security
[37] https://spring.io/projects/spring-data
[38] https://projects.spring.io/spring-social/
[39] https://spring.io/projects/spring-cloud
[40] https://maven.apache.org/

reference to an Android code sample.[41]

Question 12: "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 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 client is curious about the design of the code sample and looks for explanations about it.[42]

Question 13: "Does anybody know why the Spring Boot Guide includes two different types of integration tests? [...]"

Finally, in the following question, the client references the code sample to illustrate his doubt with a concrete example better.[43]

Question 14: "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: Clients 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.

### 4.2.3 (RQ3) What are the most common issues that impact code samples on GitHub?

Figure 4.5 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 clients who use code samples want to improve them, and, at the same time, maintainers care about improvements and clients' requests. In both platforms, 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*

[41]https://stackoverflow.com/questions/28355353
[42]https://stackoverflow.com/questions/46732371
[43]https://stackoverflow.com/questions/40832882

(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.

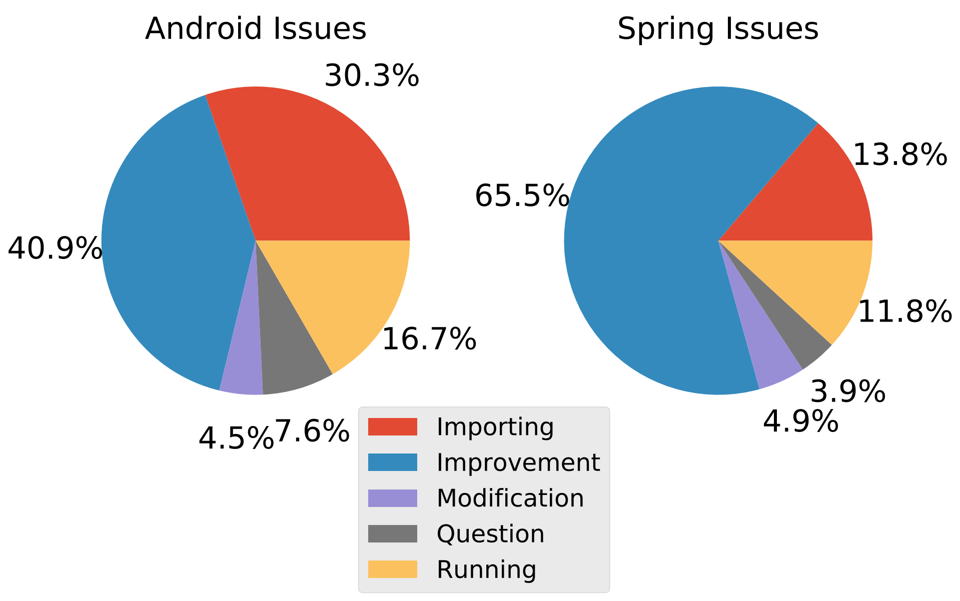


Figure 4.5: Distribution of issues' categories.

Figure 4.6 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.

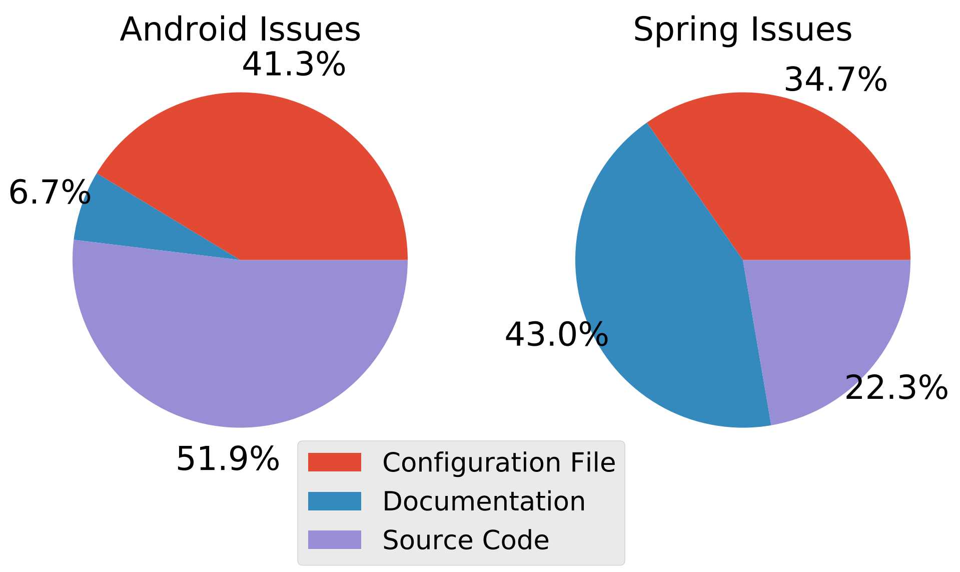


Figure 4.6: Distribution of modification types.

Lesson Learned 6: Clients 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.

## 4.3 Implications

**Many hands make work lighter**. We found that clients frequently try to modify or improve the code samples, but face some problems, for instance, expanding the sample with new features. We also detect that clients may even suggest the improvement of code samples via GitHub issues. Experienced clients can help newcomers deal with code samples problems. In that way, they may feel part of the community.

**From code sample to beyond**. We find that many questions are created, for example, when clients try to use camera API on Android and security features on Spring Boot. Maybe clients 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 a canvas drawing. This could help to spread the technology and also to support clients.

**Each client counts**: We found that code sample clients don't usually use the copy/paste approach to reuse code sample code in their own repositories. Besides that, there is a minority that does and it must not be ignored. Code sample guides also stated that clients will copy and paste code sample code into their own projects and may put it in production [20]. For that, code samples need to follow generally accepted best practices and does not do anything that will cause an application to be insecure, grossly inefficient, bloated, or inaccessible.

## 4.4 Threats to Validity

This section discusses the study limitations based on the four categories of validity threats described by Wohlin et al. [106]. 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 *low reliability of measures*: we conduct a manually classification of Stack Overflow questions and GitHub issues. This is a risk since we could have bias to this classification. To mitigate this risk, we used thematic analysis [21] and Cohen Kappa test [19].

*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 have specific characteristics about their usage, understanding and demands, different from another programming languages and could affect this study. This risk cannot be avoided since we analyzed only code samples 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 four platforms might present specific aspects compared to other platforms. This risk cannot be avoided because our focus is on this platforms. However, we argue that they are relevant and worldwide adopted platforms that have millions of end-users. Therefore, we believe the results extracted can be the first step towards the generalization of the results.

## 4.5 Related Work

Frameworks are used to support development, provide source code reuse, improve productivity, and decrease costs [44, 68, 80]. 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 [91]. Moreover, with the popularity and relevance of the Question and Answer (Q&A) sites such as Stack Overflow, some studies propose approaches and tools to search and/or retrieve source code samples and explore the 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 [34, 50, 78, 85, 91]. For instance, Holmes and Murphy [34] proposed a technique that recommends source code examples from a repository by matching structures of given code. FuzzyCatch [72] provides a code recommendation tool, based on fuzzy logic, for handling exceptions.

XSnippet [85] 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 [34]). 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.

*Assessing Q&A code snippets.* Nasehi et al. [90] 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. [108] 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.* [23] 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.* [56] 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 [33, 84, 109]. For instance, Roy and Cordy [84] 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.

## 4.6 Conclusion

In this study, we explore the code sample usage through the copy/paste approach. We also assess the most common issues related to code samples. Finally, we analyzed the most common Stack Overflow questions related to code samples. Based on our results on this study, we stand out the following results: (1) the copy/paste approach had low frequency of usage in repositories of code sample watchers; (2) clients typically face problems when trying to modify the code samples; (3) The main need raised by code sample clients are suggestions to improve the code sample. From these findings, some questions arose. For example, if the most common problem faced by clients was when trying to modify the code sample, then how experienced are these clients? And if the most common suggestion was to improve the code sample, are organizations paying enough attention when releasing new updates to their code samples? These and some other questions helped in the development of the studies of the other chapters.

# CHAPTER 5

# Actors of Code Samples

**Context:** As already stated in Chapter 2, to a health SECO context is important to exist interaction between the actors [52]. In Chapter 2 and Chapter 4, we show that Stack Overflow (SO) is an environment where newcomers clients, experienced clients, and code sample maintainers can interact with each other. In general, we believe that newcomers clients, when facing problems or raising questions about the platform or code samples, turn to SO for a solution. On the other hand, more experienced clients or maintainers, help newcomers by answering these questions in a detailed and accurate way.

**Problem:** Few studies that explore the most common characteristics of code samples and similar projects. [59, 112]. There is a gap in knowledge about the profile of actors that interact with code samples. In Chapter 4, we found that the motivation behind most code sample questions on SO is related to trying to modify the code sample. That is, clients face problems when trying to extend the code sample to another context, probably more complex. This may indicate that the clients are less experienced than organizations believe and should receive assistance through more complete artifacts and code sample extension guides. Through the assessment of experience level of actors that interact with the code sample can be useful to (1) organizations that seek to disseminate knowledge to the community and accelerate contributions (including evolution) around its products; and (2) actors that seek to contribute to the "curatorship" of the repository, as they will understand the profiles of other actors.

**Purpose:** As a way of moving towards a characterization of actors that interact with code sample, In this chapter we seek to achieve the following goal: Explore characteristics from actors that interact with code samples, via

SO and GitHub information, to fill the gap of knowledge and help organizations to create more suitable code samples.

To do so, we built three research questions. (RQ1) What is the reputation of questioner about code samples? (RQ2) What is the reputation of answerer about code samples? (RQ3) What are the characteristics of code sample maintainers? To answer these questions, we conduct a study that explores the degree of experience of questioners and answerer through reputation on SO. In addition, we also explore the characteristics of code sample maintainers via the information provided by GitHub. We explore actors' information from four different platforms: Android, AWS, Azure, and Spring Boot.

**Structure:** Section 5.1 shows methods and steps used to conduct the study. Section 5.2 shows the results obtained after we performed steps of Section 5.1. Section 5.3 presents the implications of results. Section 5.4 shows the threats of validity from this study and how we mitigated them. Section 5.5 presents the related work and differences to our study. Finally, Section 5.6 presents the conclusion of this study.

## 5.1 Study Design

This section presents how we design and conduct the exploratory study. Section 5.1.1 presents the code sample selection. Section 5.1.2, Section 5.1.2 and Section 5.1.2 present the research question, their rationale and method to be answered.

### 5.1.1 Code Sample Selection

In Chapter 3 and Chapter 4 we explored information from code samples of only two platforms: Android and Spring Boot. For this chapter and Chapter 6, we decide to explore information from code samples of four platforms: Android, AWS, Azure, and Spring Boot. In that way, we could expand the number of analyzed code samples, increasing the reliability and generality of our results. For the study presented in this chapter we selected 176 code samples from Android, 111 from AWS, 263 from Azure, and 57 from Spring Boot, totaling 607 code samples. We detail each selected platform and motivation behind in Section 2.4.

### 5.1.2 (RQ1) What is the reputation of questioner about code samples?

This research question aims to explore the experience degree of the clients who ask questions about code samples (questioners). To do so, we use the

SO reputation as one of the indicators of the experience and knowledge of actors [9, 67]. Reputation is a SO metric that indicates how much the community believes in a user [38]. The idea of this research question is to have a threshold that can indicate the experience level of questioners. In other words, it would be possible to know the reputation of clients who have doubts about the code samples.

*Rationale*: In Chapter 4, we found that most of the code sample questions on SO are related to the situation when the client tries to make changes to the code sample. This may indicate that clients may not have enough experience to be able to use the code sample in this way and may need support tools. By answering this research question, we can advance the characterization of clients and organizations can better define the target audience of their code samples.

Since the absolute reputation score may not be that representative, we decided to extract the reputation of the average of SO community to be compared. We obtained this information through the Stack Exchange platform[1] which provides a database referring to the reputation of all the 12,485,155 SO users. The data statistics of SO users were: Minimum (1), Maximum (1,185,733), Average (119), Median (1), and Mode (1).

To identify SO questions related to code samples, we used the following approach: a question is related to a code sample when a reference (URL) to the GitHub repository is found in the body of the question. The entire procedure for collecting the questions was conducted through the official Stack Exchange API[2] and from StackAPI library[3]. We extracted questions created from November 2013 to June 2020.

### 5.1.3 (RQ2) What is the reputation of answerer about code samples?

While RQ1 is interested in evaluating the experience of who ask about code samples, this research question aims to evaluate the experience of who answer questions about code samples (answerer). We believe that answerer should be more experienced than questioner about code samples on SO.

*Rationale*: Thus, answering this research question helps us to characterize more about the profile of actors around code samples. This will fill the gap of knowledge about these actors, and may assist organizations to create code samples most appropriate to their target audience.

The answers used in this research question are the answers to the ques-

[1]https://archive.org/download/stackexchange/stackoverflow.com-Users.7z
[2]https://api.stackexchange.com/
[3]https://stackapi.readthedocs.io/

tions found in RQ1 and presented in Section 5.1.2. In the same way, we extracted answers from November 2013 and June 2020. In addition, we collected the data through the official Stack Exchange API and from StackAPI library. In SO, there is a concept of accepted answer. The SO user that ask a question has the option to accept an answer. Accepting an answer is not meant to be a definitive and final statement indicating that the question has now been answered perfectly. It simply means that the author received an answer that worked for them personally. [4]

### 5.1.4 (RQ3) What are the characteristics of code sample maintainers?

In RQ1 we aim to explore who ask about code samples, in RQ2 we aim to explore who answer about code samples, and in this research question we aim to find characteristics of code sample maintainers. To cover another gap of knowledge in code sample maintenance, we explore characteristics like popularity, experience, and location, provided by GitHub API.

*Rationale*: The interaction between the actors is essential to keep a health SECO [52]. For these interactions to occur properly, it is necessary that the actors know the characteristics of each other. For example, cultural and linguistic aspects can influence how interactions occur. Another example, geolocation factors can affect the communication time between actors.

First of all, we need to know which GitHub users are maintainers of code samples. To do so, we consider a code sample maintainer the user that (1) performed at least one commit directly to the main code sample branch or (2) accepted (merged) at least one Pull Request (PR) to the code sample main branch. Second, we extract all information via GitHub API wrapped in python library PyGitHub[5]. To assess maintainer popularity we use the *following* metric, which means the number of GitHub users following the maintainer. To assess maintainer experience we use *GitHub time*, this metric is the number of days since the maintainer creates his account on GitHub. For location information, GitHub provides a field that the user can fill where he is located, but it is optional. As this field is free to be filled, the user can write anything, so we removed information that is not interesting to us. For example, we exclude *127.0.0.1*, *Earth*, *XYY-IA*, *somewhere*, and others.

[4]https://stackoverflow.com/help/accepted-answer
[5]https://pygithub.readthedocs.io/en/latest/

## 5.2 Results

In this section, we presents the results for each research question. Section 5.2.1 shows results of experience degree questioners (RQ1). Section 5.2.1 presents the results of experience degree of answerer (RQ2). Finally, Section 5.2.3 presents characteristics of code sample maintainers (RQ3).

### 5.2.1 (RQ1) What is the reputation of questioner about code samples?

We obtained 549 SO questions, from which 453 were related to Android code samples, 36 questions related to code samples from Azure, 6 of the code samples from AWS, and 54 related to Spring Boot code samples. Figure 5.1 presents the reputation score of questioners about code samples, for each platform. On the median, Spring Boot ones presented higher reputation, with 430.5, followed by AWS with 388.5, Android with 323, and Azure with 57. Compared with the reputation of the average of SO community, we can note that questioners are most experienced for most of the platforms. Azure ones seem to be more inexperienced them other platforms and the average of SO community. This may indicate that each platform and its code samples have different target audiences and organizations must create code samples suitable for each one.

Still on Figure 5.1, we can see that there is a wide distribution among reputations, especially for Android and Spring Boot. Looking for the data, we could find clients with 1 until over 100,000. This may indicate that, although most clients are a little more experienced than the average of SO community (119), the target audience for code samples ranges from the inexperienced clients, who need simple code samples and even auxiliary content to help them understand. them better even the most experienced clients who need more complex code samples to meet their demands.

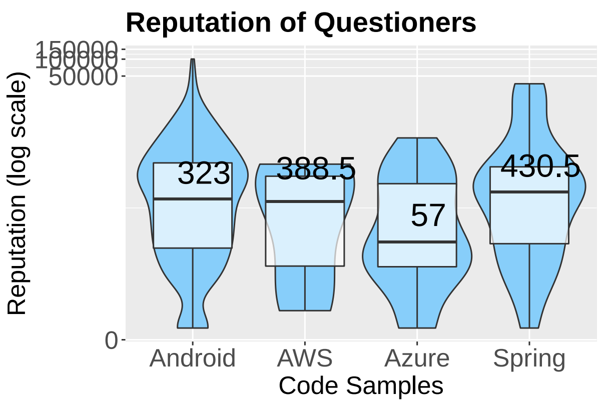


Figure 5.1: Reputation of questioners about code samples on SO.

Lesson Learned 1: The target audience of code samples can range from the inexperienced clients to the most experienced clients. Despite this, each organization should evaluate its target audience in a specific way, to create suitable code samples.

### 5.2.2 *(RQ2) What is the reputation of answerer about code samples?*

From the set of questions obtained in Section 5.2.1, we have 639 answers, with 538 answers related to Android code samples, 62 related to Spring Boot code samples, 5 to AWS code samples and 34 to Azure. Of these 639 answers, 145 are accepted answers. Section 5.2 (left) presents the reputation of SO users that answer questions about code sample. We can note that the Spring Boot platform has the most experienced answerers, with 1,391 on the median. Followed by AWS with 1,387, Android with 885, and Azure with 646 of reputation on the median. Figure 5.2 (right) shows the reputation of who have accepted answer. We note that AWS ones present 3,970 of reputation on the median, followed by Android with 1,514, Azure with 1,226, and Spring with 1,211.5. Comparing these results with the average of SO community (119), we can note that answerers are, in general, much more experienced. In addition, answerers also are much more experienced than questioners.

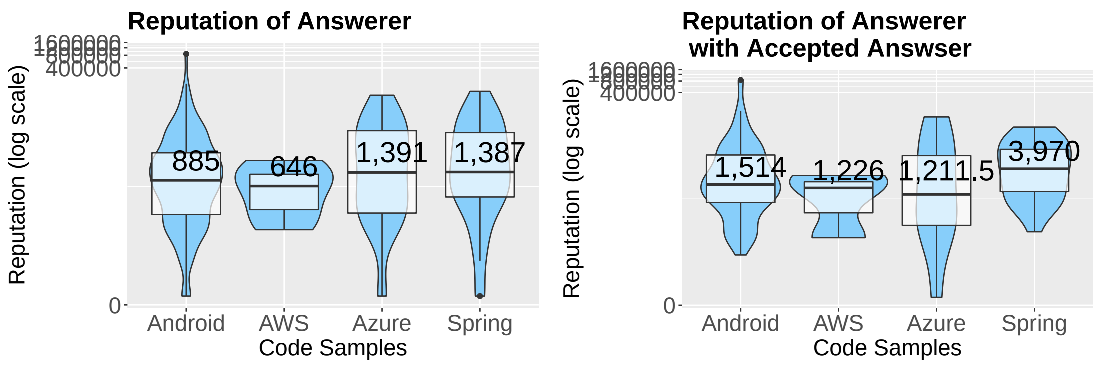


Figure 5.2: Reputation of answerers about code sample (left) and reputation of answerers with accepted answers (right).

Lesson Learned 2: Answerers of code samples questions are very experienced compared with questioners and the average of SO community.

### 5.2.3 *(RQ3) What are the characteristics of code sample maintainers?*

After carrying out the selection of maintainers described in the Section 5.1.4, we obtained a set of 740 code samples maintainers, 186 from Android, 228 from AWS, 253 from Azure, and 74 from Spring Boot. Figure 5.3 shows the number of followers and following of code sample maintainers. Figure 5.3 (left) present the followers, and we can see that in Android and Spring Boot, the dispersion of the number of followers is greater compared with AWS and Azure. In Android and Spring Boot the median is 29.5, while Azure presents 10 and AWS with 3 followers. Besides this difference between organizations, it is noted that the majority of code sample maintainers are not as popular on GitHub as we could think. Figure 5.3 (right) presents the number of followings of code samples maintainers. As in the number of followers, the number of followings presents a high dispersion of the that, where some users follow more than 200 GitHub users, while some maintainers follow zero users. In Spring, code sample maintainers follow four users on the median, while Android presents one follow, AWS and Azure zero on the median. There are some reasons for this low number of followers, for example, code sample maintainers can use a different GitHub account for their personal repositories and the organization repositories.

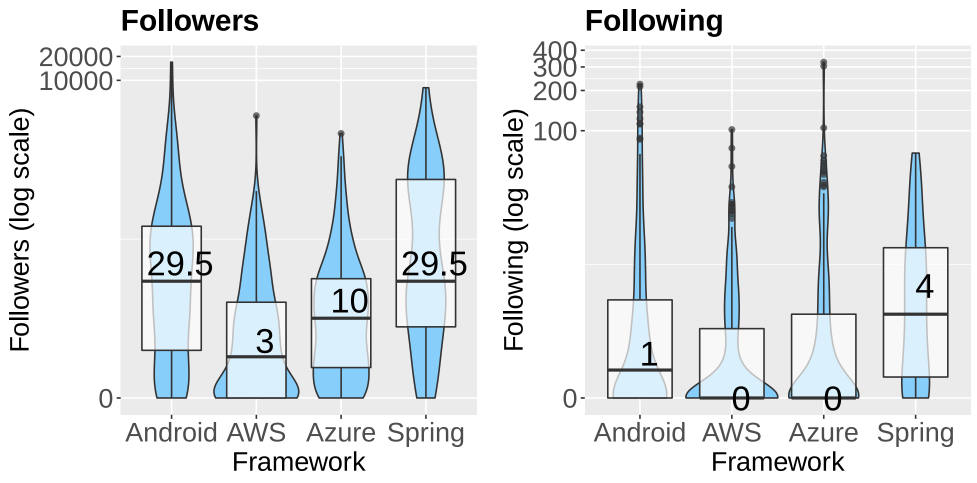


Figure 5.3: Followers (left) and Following (right) of code sample maintainers.

Figure 5.4 presents the time in days since the code sample maintainers create their GitHub account. We can note that maintainers of Spring and Android present on the median of 3,665.5 and 3,360.5 days respectively, which means around 10 years of GitHub usage. In Azure and AWS the time is smaller, Azure maintainers have 2,662 on the median, while AWS maintainers present 2,182. It means around 7 years for Azure maintainers and around 6 years of GitHub usage to AWS maintainers. This result may be explained by the niche of each platform. The development of mobile and web applications has been popular longer than the development of cloud computing, which has been popularized more recently.

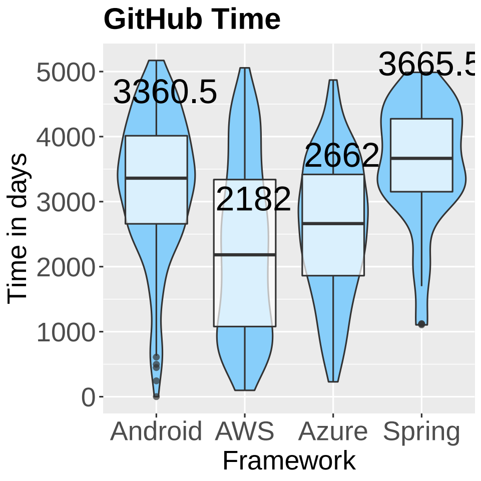


Figure 5.4: Number of days from code sample maintainers on GitHub.

Table 5.1 presents the results about maintainers location for each platform. We can note that for all platforms, the majority of maintainers are located in the United States of America, representing 55 (30%) of Android maintainers, 74 (33%) in AWS, 97 (38%) in Azure, and 19 (25%) in Spring. A high part of them is in cities inside or around Silicon Valley, such as San Jose, Sunnyvale, Mountain View, Palo Alto, and others. However, there is a percentage of maintainers distributed in other regions, countries, and continents. For example, in Android, we found 11 (6%) maintainers in the United Kingdom, 4 (2%) in Australia, 4 (2%) in Japan, and 2 (1%) in Germany. Also, others 8 (4%) maintainers are distributed in Spain, Canada, Finland, India, Russia, Switzerland, and Italy. In AWS, besides the US, we found the United Kingdom with 5 (2%) maintainers, Brazil with 4 (2%), Germany with 3 (1%), and France with 3 (%1). And others with 21 (9%) maintainers in countries such as the Netherlands, Canada, India, China, Singapore, Australia, Japan, Switzerland, Belgium, Norway, Sweden, Taiwan, Ireland, Luxembourg, and Argentina. For Azure platform, besides the US, we found 11 (4%) maintainers in China, 8 (3%) in France, 7 (3%) in Canada, and 7 (3%) in India. We also found 13 (5%) maintainers in other countries such as UK, Germany, Netherlands, Singapore, Belgium, Poland, New Zealand, Mexico, Ukraine, Israel, and Iceland. Finally, in Spring, we also found 8 (11%) maintainers in Germany, 5 (7%) in UK, 3 (4%) in Canada, and 3 (4%) in Poland. Spring also presented 10 (13%) maintainers located in France, Belgium, China, India, Netherlands, Russia, Lithuania, and Turkey.

Also in Table 5.1, we found a high number of maintainers without location information. In Android, 97 (53%) of maintainers had location information empty in their GitHub accounts. It happened in other platforms as well, in AWS, 116 (51%) of the maintainers had empty locations, Azure presented 106 (42%) and Spring with 23 (30%).

Table 5.1: Maintainers location

| Android | | AWS | | Azure | | Spring | |
|---|---|---|---|---|---|---|---|
| Region | # | Region | # | Region | # | Region | # |
| US | 55 (30%) | US | 74 (33%) | US | 97 (38%) | US | 19 (25%) |
| UK | 11 (6%) | UK | 5 (2%) | CN | 11 (4%) | DE | 8 (11%) |
| AU | 4 (2%) | BR | 4 (2%) | FR | 8 (3%) | UK | 5 (7%) |
| JP | 4 (2%) | DE | 3 (1%) | CA | 7 (3%) | CA | 3 (4%) |
| DE | 2 (1%) | FR | 3 (1%) | IN | 7 (3%) | PL | 3 (4%) |
| Others | 8 (4%) | Others | 21 (9%) | Others | 13 (5%) | Others | 10 (13%) |
| Ignored | 3 (2%) | Ignored | 1 (0%) | Ignored | 5 (2%) | Ignored | 2 (3%) |
| Empty | 97 (53%) | Empty | 116 (51%) | Empty | 106 (42%) | Empty | 23 (30%) |
| Total | 184 (100%) | Total | 227 (100%) | Total | 254 (100%) | Total | 76 (100%) |

Lesson Learned 3: Code sample maintainers are not the most popular or influential users on GitHub. Besides that, they seem to be experienced users on GitHub. A wide part of them located in USA but there is a set of maintainers distributed in other countries as United Kingdom, China and Germany.

## 5.3 Implications

**From beginners to experts.** In general, we identify a wide range of reputations among clients. We found inexperienced clients with a reputation close to one to more experienced clients with a reputation greater than 100,000. This indicates that, contrary to common sense, code samples, despite their simplicity, can target even more experienced clients.

**For each target audience, a strategy.** We found that clients who ask, in general, are relatively experienced. Despite this, we noticed that there is a difference in the degree of experience of who ask between platforms. In this way, each organization must identify the target audiences for its code samples and develop code samples and auxiliary tools that best suit each of them.

**Simplicity is not all.** In this chapter, we found that newcomers are more experienced than the community. but, in Chapter 3, we found that code samples are smaller and simpler than conventional projects. And in Chapter 4, we found that clients typically face problems when they try to modify the code sample. In this case, even with the code samples following guidelines and best practices, they were not sufficiently explanatory for the target audience with above-average reputation. Organizations should provide code samples that meet complex usage scenarios, as real-world projects demand.

**Community matters.** We noticed that who answer questions about code samples are actually more experienced than the community average and also more experienced than who ask. This may indicate that the community itself can help to solve this questions. Thus, it is interesting that organizations provide incentives for this type of interaction between their actors. Whether encouraging the most experienced clients themselves or with professionals in the field such as evangelists or developer relations. This can strengthen the community and imply the strengthening of SECO as well.

**Old is not cool**: We found that code sample maintainers have some age on the GitHub, in general, around six years. Besides that, they are not so popular on GitHub. Perhaps more influential maintainers in the community could use its popularity to attract more clients to code samples, helping to spread knowledge about the platform and helping to further consolidate SECO as well.

## 5.4 Threats to Validity

This section discusses the study limitations based on the four categories of validity threats described by Wohlin et al. [106]. 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.

*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 *sample selection*: on the Stack Overflow platform, any user can create question without any restriction. In other words, there is no filter to have only relevant questions, and poor questions could be a risk. To mitigated this we only select questions with positive score.

*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 *measurement metrics*: we evaluated the today's reputation on Stack Overflow and this could not represent the reputation when the user created the question. This threat can not be avoid since Stack Overflow API only provides today's reputation.

*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 four platforms might present specific aspects compared to other platforms. This risk cannot be avoided because our focus is on this platforms. However, we argue that they are relevant and worldwide adopted platforms that have millions of end-users. Therefore, we believe the results extracted can be the first step towards the generalization of the results.

## 5.5 Related Work

**Code samples.** Menezes et al. [59] carries out an exploratory study, seeking to clarify characteristics found in code samples, as well as defining a set of lessons learned. Zhang et al. [112], an approach to API documentation enrichment is proposed, mapping code samples into corresponding usage scenarios. Although the work cited focuses on code samples, they do not address aspects of clients who seek or use these projects. In this work, developers who use code samples are explored, evidencing their degree of expertise in the OS.

**Reputation score in Q&A platforms.** Unlike this work, which analyzes the target audience of code samples by reputation, Movshovitz-Attias et al. [69], focuses on the behavior pattern of users of high and low reputation, concluding that, although most of the questions asked come from users with a low reputation, on average, a user with a high reputation asks more questions than a user with a low reputation. In the study carried out by Bosu and colleagues [9] an empirical evaluation of the OS user reputation is presented, guiding new users to obtain high reputation scores in an agile way. Differently, in this work, the reputation of developers who use several code samples in the OS is observed, exposing possible contributions to organizations in improving their repositories and making them more adaptable to the characteristics of the developers who use them.

## 5.6 Conclusion

Based on the results obtained from conducting the exploratory study, the following results stand out: (1) Contrary to what was imagined, clients who ask or answer about code samples are more experienced than the community in general; (2) Code samples are likely to be used by a wide variety of clients ranging from novices to the most experienced; (3) Organizations should look for characteristics in their code samples compatible with the average community experience to avoid a high number of questions; (4) Organizations should analyze code samples individually in some situations to identify specific target audiences (eg, less or more experienced than average), and thus adapt them to these clients; (5) Organizations should engage experienced clients in their code samples so they can answer other clients' questions on Q&A sites.

CHAPTER 6

# Maintenance of Code Samples

**Context:** Since code samples belong to a SECO context, it is important to exist interaction between the actors since it is essential for the survival of the SECO [52]. As previously stated in Chapter 2, clients can interact with organizations through GitHub. Clients can related bugs or claim for new usage scenarios via GitHub issues. In addition, clients can also contribute to the code sample code via Pull Request (PR). Whether coming from external contributors or from within the organization itself, it is vital that code samples preserve or even improve their quality [20, 59].

On the other hand, Lehman's laws of software report that evolution makes software projects more complex and harder to maintain [48]. In a similar way Jacobson *et al.* [39] defined software entropy as a closed system's disorder that cannot be reduced, it can only remain unchanged or increase. We know that code samples are software projects, made available by organizations, for educational purposes, to assist clients with platform features [20, 59].

**Problem:** As code samples are software projects, it is natural that they also evolve [59]. For example, a code sample evolves to update to a new platform version, otherwise, it becomes outdated. But it's not strict as was stated in Chapter 3, a code sample may change to accommodate new usage scenarios to clients' needs. However, the uncontrolled evolution of code samples can make their quality decay. They may become large in code size and project size, less readable, and more complex. Low-quality code samples can implicate a barrier to the learning process, and raise doubts for clients. They may lose their educational purpose and become useless. Even worse, they may be another source of questions, confusing clients. Therefore, instead of being the solution to a problem, they may be another problem that clients must overcome to understand or keep up to date on an organization's product.

So, it is essential for both organizations and academia to investigate questions such as what are the most common code sample maintenance activities? how organizations are evolving their code samples how code sample evolution impacts their clients. In this way, we can raise points that need more attention from organizations that are already evolving their code samples, but also help new organizations that would like to create and evolve their own code samples.

**Purpose:** Since there are still few studies that evaluate how organizations evolve their code samples, the goal of this chapter is Evaluate how organizations maintain their code samples over time and the impact of this maintenance on clients, analyzing their maintainers' activities, source code evolution, and clients' doubts on Stack Overflow (SO), to provide insights for other organizations.

To do so, we built three research questions. (RQ1) How are maintenance activities distributed in code sample repositories? (RQ2) How do code samples evolve over time? (RQ3) Does the evolution of code samples may impact clients' questions? To answer these questions, we conduct an exploratory study on top of 166 code samples implemented in Java and provided by Android, AWS, Azure, and Spring Boot. We assess information about maintenance activities on GitHub. In addition, we assess code sample evolution of code size, readability, project size, OO design, and code complexity. Finally, we explore questions about code samples on SO and compare their raise with code sample evolution.

**Structure:** Section 6.1 shows methods and steps used to conduct the study. Section 6.2 shows the results obtained after we performed steps of Section 6.1. Section 6.3 presents the implications of results. Section 6.4 shows the threats to validity of this study and how we mitigated them. Section 6.5 presents the related work and differences to our study. Finally, Section 6.6 presents the conclusion of this study.

## 6.1 Study Design

This section presents how we design and conduct the exploratory study. Section 6.1.1 presents the code samples selection. Section 6.1.2, Section 6.1.3 and Section 6.1.4 present the research questions, their rationale and method to be answered.

### 6.1.1 Code Sample Selection

We selected code samples written in Java from four different organizations: Google, Microsoft, Amazon, and Spring. More specifically, we analyzed code samples of the following products: Android, Azure, AWS, and Spring Boot. Details about each platform and motivation are described in Section 2.4. The extraction of the code samples was conducted via GitHub [29]. It's worth noticing that we only selected a code sample if it was mentioned in at least one question on SO. Details of questions selection is presented in Section 6.1.4.

### 6.1.2 (RQ1) How are maintenance activities distributed in code sample repositories?

In this research question, we aim to explore the maintenance process by looking at the division of maintenance activities on code samples repositories in GitHub. *Rationale*: The results of this research question can help us understand of the maintenance process of the code samples. Code samples are created to help clients in the learning process of platform features, and for that, it is interesting that they get and maintain high code quality. In that way, we want to know if the code sample maintainers use PR approach, using a review process, that can be helpful to maintain code quality. On the other hand, they could insert new code directly into the main branch and that could be a factor to introduce new bugs or bad code.

We use GitHub events, provided by GitHub Activity API, to assess how organization distribute maintenance activities. The GitHub Activity API allows us to list events and feeds, and also manage notifications, starring, and watching for the authenticated users [1]. As a part of Activity API, GitHub make available the event API to the user's events. Trough events, we can track activities performed by GitHub users as create a branch, fork a repository, create a pull request or an issue, and among others. There are a few types of events and we will present they and they definition as well. **CommitCommentEvent** is triggered when a commit comment is created. **CreateEvent**: when user create a reference to a repository, as a Git branch or tag. **DeleteEvent**: in the same way as create, delete is when user delete a reference to a repository, branch or tag. **Fork**: this event is triggered when a user fork a repository. **IssueCommentEvent**: Activity related to an issue or pull request comment, can be *created*, *edited* or *deleted*. **IssuesEvent**: Activity related to an issue, and can be *opened*, *edited*, *closed*, *reopened*, *assigned*, *unassigned*, *labeled*, or *unlabeled*. **MemberEvent**: Activity related to repository collaborators and Can be *added* to indicate a user accepted an invitation to a repository. **PublicEvent**: When

[1]https://docs.github.com/en/rest/reference/activity

a private repository is made public. **PullRequestEvent**: Activity related to PR, and the action that was performed. Can be one of *opened*, *edited*, *closed*, *reopened*, *assigned*, *unassigned*, *review_requested*, *review_request_removed*, *labeled*, *unlabeled*, and *synchronize*. **PullRequestReviewEvent**: Activity related to pull request reviews, and only can be created. **PullRequestReview-CommentEvent**: Activity related to pull request review comments in the pull request's unified diff. Can be *created*. **PushEvent**: One or more commits are pushed to a repository branch or tag. **ReleaseEvent**: Activity related to a release and can be *published*.

Since the GitHub API does not provide us with a list of repository maintainers, we need to find a way to get it. For that, we consider a code sample maintainer a GitHub user that: (1) performed at least one commit directly to the code sample main branch or (2) accepted at least one PR to the code sample main branch. Then we got a list of 740 maintainers, 186 from Android, 228 from AWS, 253 from Azure, and 74 from Spring. After maintainer selection, we extract from them, all GitHub events related to code samples, but it is important to highlight that GitHub API only provides the events performed in the last 90 days ago.

### *6.1.3 (RQ2) How do code samples evolve over time?*

Aiming to answer RQ1, we evaluated historical changes in code samples and extracted the trend of four groups of metrics: source code size, source code readability, project size, OO design and code complexity.
1. Source code size: We first explored the evolution of the code sample size by computing metrics related to their source code size. We focused on 16 metrics: Physical Lines (PL), Lines of Code (LOC), Declarative Lines of Code (DLOC), Executable Lines of Code (ELOC), Commented Lines of Code (CLOC), Blank Lines (BL), Number of Class Variables (NV), Number of Instance Variables (NIV), NUmber of All Methods (AM), Public Methods (NPRM), Private Methods (NPM), Protected Methods (PM), Local Methods (LM), Statements (STMT), Declarative Statements (DSTMT), Executable Statements (ESTMT) [87]. Rationale: Small code with simple structures may improve code understanding [54]. So, it is important to assess whether *source code size* remains as small and concise as possible throughout the evolution of code samples.
2. Source code readability: The metrics selected to cover code readability are Buse and Weimer's Readability (BWR), Scalabrino *et al.*'s Readability (SAR), and Comment to Code Ratio (CCR). The readability metric (BWR) defined by Buse and Weimer [15], is based primarily on local, line-by-line features such as line length, identifier length, and indentation. The value of the metric varies between 0 and 1, with higher values indicating higher readability. In a similar

way, Scalabrino *et al*'s [86] readability (SAR) uses source code lexicon analysis, as consistency between code and comments, specificity of the identifiers, textual coherence, and comments, to assess the code readability. The Comment to Code Ratio (CCR) measures the ratio between code and comment into a code sample to analyze how commented it is. Rationale: Code samples can bridge technical resources and clients who want to learn or be updated about these resources. Therefore, it is vital to analyze if code samples are not becoming difficult to read and hard to understand.

3. Project size: We also investigated the evolution of the code sample size by computing metrics related to project size. We extracted the following three metrics: Number of Java Files (JAVA), Number of Classes (CLASSES), and Number of Packages (PKG). Rationale: In addition to the source code size, it is important to observe the *project size*. An increasing number of *source code files* (i.e., java files), *classes* and/or *packages* may demand more effort to be driven by clients in order to understand the code sample.

4. OO design: To explore the OO design of code samples, we extracted the so-called CK metrics suite [17]. The CK metrics suite comprises the following metrics: Weighted Methods per Class (WMC), Coupling Between Object Classes (CBO), Depth of the Inheritance Tree (DIT), Number of Children (NOC), Response for a Class (RFC), and Lack of Cohesion of Methods (LCOM). Rationale: The authors of this suite of metrics claim that these measures can aid users in understanding design complexity [17]. In addition, there are several studies using these metrics to analyze the design of OO projects [47, 74, 97, 111]. Since we are analyzing Java projects, it is important to understand how the OO design changes during its evolution. A worsening in the value of these metrics can indicate a worsening in the design along the evolution making it difficult for the clients to understand code samples.

5. Code Complexity: Finally, we also investigated aspects of code complexity. To do that we select 9 metrics selected to measure code complexity: FANIN, FANOUT, Cyclomatic Complexity (CC), Cyclomatic Complexity Modified (CCM), Cyclomatic Complexity Strict (CCS), Essential Complexity (EC), Knots (KNTS), Number of Paths (NPATH) and Number of Paths in Logarithm Scale (NPLOG). Rationale: It is worth analyzing code complexity since an increasing complexity of source code may lead to questions, and even drive off clients from the organization's products.

To extract the analyzed metrics, we used the Understand [87] tool to gather metrics related to source code size, project size, and OO design. The code readability is calculated using the implementation of Buse and Weimer's [15], which is largely adopted by the literature [5, 66]. In a similar way, Scalabrino et al's readability metric is computed using the authors' tool [86]. To retrieve evolutionary data, we implemented a script using GitPython [100] library to handle commit information. It is worth mentioning that the whole historical data available in the official repository of each code sample was analyzed, which comprises from the first commit of the project to the last commit performed until the end of August 2021.

The process of data aggregation was performed as follows: (1) at each commit of each code sample, the analyzed metrics were extracted. File-based metrics (source code size, source code readability, and OO design) were aggregated by averaging the metric of the code sample's Java files. In this way, we had for each commit of each code sample a value for each metric. (2) To organize the data over time, we consider the average of the metrics of the commits carried out each month. In this way, we were able to have the evolutionary data of each code sample per month.

The metrics trend over time were calculated by applying the Mann-Kendall test [43, 53] with *p-value* $\leq 0.05$ for each analyzed metric. The Mann-Kendall trend test (also known as the M-K test) is used to analyze data collected over time for increasing, decreasing, or no trends [32]. We used the pymannkendall package in Python[36] to compute it.

### 6.1.4 (RQ3) Does the evolution of code samples may impact clients' questions?

For this research question, we mined questions from the Q&A website SO. We considered that a question is related to a selected code sample when a reference to the official code sample repository is found in a question. This reference should be made by including the URL of the code sample in the question body. The questions were selected using the official Stack Exchange repository [Inc]. By using the described steps, we found 1,188 questions, 939 of these questions are related to Android code samples, 149 from Spring Boot, 84 for Azure, and 15 for AWS.

To assess whether careless evolution is related to a raise of questions related to code samples, we used the number of SO questions related to the code samples analyzed and associated them to the metrics extracted in RQ2 (Section 6.1.3). To obtain the correlation between each metric and the SO questions, we used two different correlation tests. If the data presented a normal distribution, we used the Pearson's correlation test [6], otherwise, we applied the Spearman's correlation test [92]. To find whether the data presented normal distribution, we used Shapiro-Wilk's test [88]. To automate this step, we used the scipy package in Python to compute all statistical tests. It is important to highlight that all data were normalized before we run the statistical tests.

Rationale: A correlation may mean that the metric (alone or together with other metrics) may have led to a difficulty in understanding the code sample. In addition, with this analysis, it would be possible to identify metrics (i.e., properties of code samples) that are important to avoid clients' questions.

## 6.2 Results

This section presents the results obtained during the extraction of data presented in Section 6.1. More specifically, Section 6.2.1 presents the result found on maintenance activities of code samples (RQ1). Section 6.2.2 shows the result obtained from evolution metrics of code samples (RQ2). And Section 6.2.3 presents the results of SO questions and their correlation to evolution metrics (RQ3).

### 6.2.1 (RQ1) How are maintenance activities distributed in code sample repositories?

Table 6.1 presents the GitHub events of code sample maintainers. We can note that the Push event is the most common event with 1,264 (36.47%) on code samples. This means that code sample maintainers dedicate more than 1/3 of their activities on GitHub to uploading new versions of the code sample to the repository, adding new features, fixing bugs, or updating it to a new platform version, also in the source code, documentation or configuration files.

The second most common event is Pull Request with 802 (23.14%). This event is related to Pull Request management, as Pull Request create or edit, request a review, labeling, and other actions into Pull Requests. Pull Request Review is the third most common event on code samples maintainers with 546 (15.75%). This event is triggered when a review is created. A code review is an important approach to improving or maintaining code quality and it is essential to code samples since they have an educational purpose. The fifth most common event is Pull Request Review Comment with 190 (5.48%) events. This event is triggered when the reviewer creates a comment into a Pull Request. As a part of code review, this communication between reviewer and code sender is essential because the reviewer can comment directly on code, providing different approaches or improvements to get a better code quality.

Table 6.1: GitHub Events of Code Sample Maintainers.

| Event | Code Samples |
|---|---|
| Push | 1,264 (36.47%) |
| Pull Request | 802 (23.14%) |
| PR Review | 546 (15.75%) |
| Delete | 212 (6.11%) |
| PR Review Comment | 190 (5.48%) |
| Others | 451 (14.96%) |
| Total | 3,465 (100%) |

This means that among the most performed maintenance activities is the insertion of changes in the code samples repository. These changes can be updated to a new platform version, add of new usage scenarios, improvement of documentation, and others. We know that the increase in size and complexity is directly related to the evolution of conventional software [48]. But we also know that code samples have an educational purpose and should keep their smallness and simplicity. [20, 59]. Thus, organizations should use good practices for maintaining code samples, such as the use of CD/CI tools to control these characteristics.

Although the Push events are the most common activity performed by code sample maintainers, exists a relevant part related to the Pull Request management, being PR creation, review or comment. As said before, it is worth mentioning that Pull Requests is a way that code samples receive contributions from internal members of the organization and external contributions, for example, from code sample clients as well. On the one hand, the Pull Request development approach helps clients to keep code bugs safe, avoid code smells, and improve code quality. This occurs because one of the steps on the PR approach is code review, where a highly experienced client assesses the PR code, to reject, accept or suggest alternatives. In addition to code quality benefits, this approach also has a social one. PR approach also can encourage code sample clients to feel free to contribute with code sample, creating PRs to fix a bug, remove a code smell, or improve documentation, feeling part of the community and contributing to the SECO health. On the other hand, if this approach is not carried out correctly, it means with a complete and thorough review, it may have a reverse effect and generate a technical debt in the code samples code, with the insertion of bugs and code smells.

Lesson Learned 1: The code sample modification is the most common activity performed by maintainers. Followed by PR activities. Due to the high amount of modifications and possible modifications from outside organization contributors, organizations should pay attention to these modifications, and may use CI/CD tools, to keep code samples small, simple, readable, and of high code quality.

### 6.2.2 (RQ2) How do code samples evolve over time?

Table 6.2 presents the results of the trend test (Mann-Kendall Test) for each analyzed metric. We present the number of code samples which statistically increased the value of a metric ($\nearrow$), decreased ($\searrow$), and presented no trend ($\longleftrightarrow$).

Table 6.2: Results

| Category | Shortname | Trend | Correlation |
|---|---|---|---|
| Code Size | PL | $\nearrow$ | + |
| | ELOC | $\nearrow$ | + |
| | LOC | $\nearrow$ | + |
| | BL | $\nearrow$ | + |
| | STMT | $\nearrow$ | + |
| | ESTMT | $\nearrow$ | + |
| | DLOC | $\nearrow$ | + |
| | DSTMT | $\nearrow$ | ? |
| | CLOC | $\nearrow$ | ? |
| | AM | $\nearrow$ | ? |
| | LM | $\longleftrightarrow$ | ? |
| | PM | $\longleftrightarrow$ | ? |
| | NV | $\longleftrightarrow$ | ? |
| | NPM | $\longleftrightarrow$ | ? |
| | NIV | $\longleftrightarrow$ | ? |
| | NPRM | $\longleftrightarrow$ | ? |
| Readability | SAR | $\searrow$ | - |
| | CCR | $\searrow$ | ? |
| | BWR | $\longleftrightarrow$ | ? |
| Project Size | CLASSES | $\nearrow$ | + |
| | JAVA | $\nearrow$ | ? |
| | PKG | $\longleftrightarrow$ | ? |
| OO Metrics | WMC | $\nearrow$ | + |
| | CBO | $\nearrow$ | ? |
| | RFC | $\nearrow$ | ? |
| | NOC | $\longleftrightarrow$ | ? |
| | DIT | $\longleftrightarrow$ | ? |
| | LCOM | $\longleftrightarrow$ | ? |
| Complexity | EC | $\searrow$ | - |
| | CC | $\searrow$ | ? |
| | CCM | $\searrow$ | ? |
| | CCS | $\searrow$ | ? |
| | NPATH | $\searrow$ | ? |
| | FANIN | $\searrow$ | ? |
| | FANOUT | $\searrow$ | ? |
| | NPLOG | $\longleftrightarrow$ | ? |
| | KNTS | $\longleftrightarrow$ | ? |

*Source code size*

The majority of code samples presented an upward trend over time in metrics related to source code size as expected. The metrics related to lines of code and their average increase over time are PL (+153%), ELOC (+131%), LOC (+124%), BL (+46%), DLOC (+65%), and CLOC (+65%) and statements as STMT (+89%), ESTMT (+84%), and DSTMT (+60%) presented a growth behavior in most of the code samples. Metrics related to the number of variables as NV and NIV and the number of methods as PM, NPM, and NPRM, unlike lines of code, surprisingly do not present a trend during the evolution in most of the code samples. No metric related to source code size showed a decreasing behavior in most of the code samples analyzed.

<u>Discussion:</u> Analyzing the code samples evolution to understand what causes the increase in source code size, we can highlight that they often occur by (i) adding new functionalities and usage scenarios to the code sample, and (ii) updating the code sample to support newer versions of frameworks and libraries. Our analysis shows there are recurring cases where there is an increase in the complexity related to the use of the product presented by the code sample. This situation requires more lines of code to implement the same functionalities previously existing in the code sample. Another situation that may be causing an increasing trend in the source code size is the maintainers' attempt to take advantage of existing code throughout the evolution and adapt it to new features, as well as update it to newer versions of frameworks and libraries. In this case, we believe there is an upward trend as the organization may not have given due importance to the quality of code during maintenance tasks of code samples, causing clients to quickly implement an evolution of the code sample without proper quality criteria. One point that, in principle, can be positive in our data is the upward trend in the number of comment lines and, proportionally, an increase in that number in relation to the PL. This, on one hand, may indicate a concern by the organization to insert comments in parts of the source code as a way to assist clients' learning. On the other hand, it is known that the increase in the number of comment lines can cause problems for the maintenance of the code sample, as these comments may be out of date, and not useful for learning, among others [54]. Thus, this trend, although beneficial in a first analysis, should be better explored and understood. Figure 6.1 shows an example of a method added during the evolution of an Azure code sample. This method contains both source code and comments, thus increasing PL, LOC, CLOC, AM, NPRM, among others.

```
//Utils.java
+/**
+* Print management lock.
+* @param lock a management lock
+*/
+public static void print(ManagementLock lock)
+{
+     StringBuilder info = new StringBuilder();
+     info.append("\nLock ID: ").append(lock.id())
+          .append("\nLocked resource ID: ")
+          .append(lock.lockedResourceId())
+          .append("\nLevel: ").append(lock.level());
+     System.out.println(info.toString());
+     }
+}
```

Figure 6.1: New method added in compute-java-manage-vm (Azure code sample).

### *Source code readability*

Readability is a measure that tries to indicate how readable a source code is. The value of the metrics analyzed in our study varies between zero and one, with higher values indicating higher readability. It is worth mentioning that, for a code sample that has an educational side, it is important that source code readability remains high and preferably does not decrease over the evolution. However, in our data, we noted a downward trend both in SAR and CCR metrics with -20% and -1% of decrease on average, while BWR does not present a trend.

Discussion: Readability metrics are based primarily on local, line-by-line features such as line length, identifier length, and indentation. Figure 6.2 illustrates the evolution of a code snippet from code sample *cognitive-services-face-android-detect* provided by Azure. In the previous version, SAR value for this file is around 0.72, while in the last version (after code addiction) the value for SAR metric goes to 0.69. It means a decrease in the readability of SAR metrics. Despite being a controversial measure, its importance is hard to be contested when it comes to code samples. The downward trend present is not a good indicator of how organizations are dealing with code samples evolution. It is common in software development to use tools to check the style of the source code based on good programming practices. This action may impact readability. This point is important, as it can show that organizations are probably not using tools that aim to improve the quality of the source code, at least for code samples. Another point that we highlight is the lack of maturity of the maintainers of the code samples. There are several commit authors in the life cycle of a code sample. This shows that occasional maintainers are assigned to code samples. We believe that more experienced programmer which is familiar with the source code of the code samples may help increase the readability.

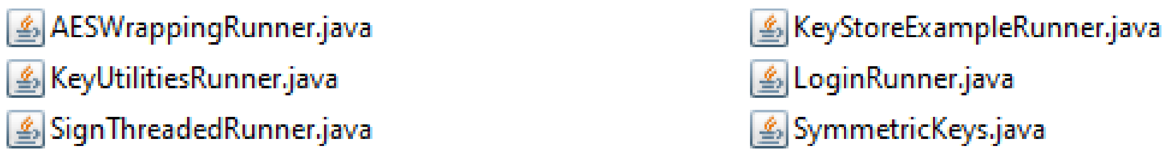
```
//MainActivity.java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
−}
+Button button1 = findViewById(R.id.button1);
+button1.setOnClickListener(new +View.OnClickListener() {
+     @Override
+     public void onClick(View v) {
+          Intent intent = new +Intent(Intent.ACTION_GET_CONTENT);
+          intent.setType("image/*");
+          startActivityForResult(Intent.createChooser(
+               intent, "Select Picture"), PICK_IMAGE);
+               }
+          }) ;
+
+          detectionProgressDialog = new ProgressDialog(this);
+     }
+}
```

Figure 6.2: Comparing two versions (from July 2018 to September 2019) of cognitive-services-face-android-detect (Azure code sample).

### *Project size*

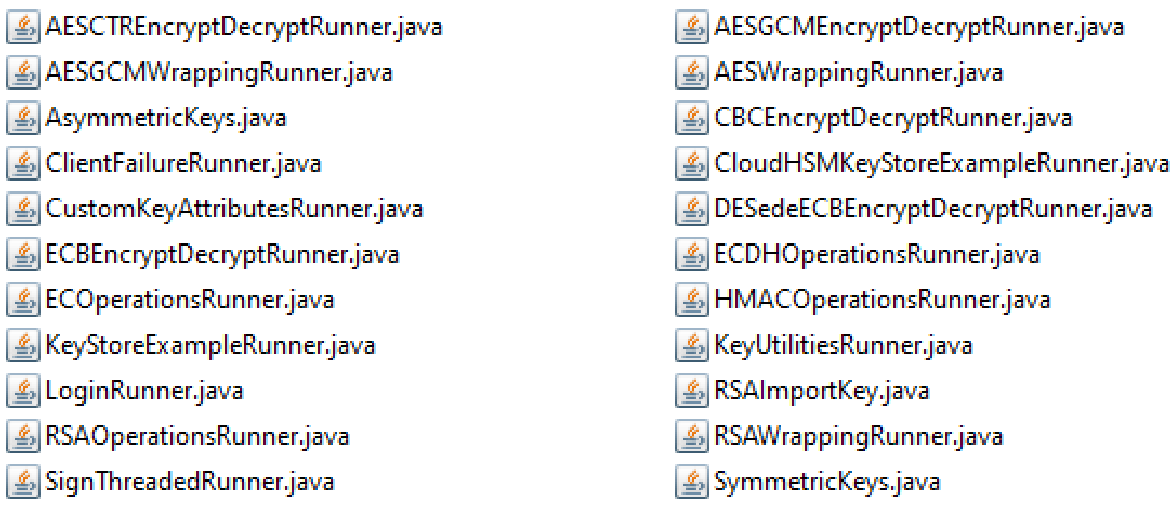
Besides the source code size, we are also interested in the project size. Table 6.2 also presents the results of the metrics number of Java files (Java), the number of classes (Classes), and the number of packages (PKG). We noted that in the case of Java and Classes, the number of code samples with an upward trend is higher than downward and code samples with no trend. On average, Java increases +208% and Classes +299% since their creation until now. PKG metric does not present a trend in most of the code samples analyzed. Considering these results with the source code size, we can say there is a real increase in code sample size since the content of files (e.g., PL, LOC, CLOC, etc.) and the number of files/classes are increasing. Figure 6.3 presents an example of the number of files increasing over time. More specifically, in this AWS code sample, there were six files in 2018 which became 22 in 2020.

<u>Discussion:</u> The first conclusion with this data is that we have a real increase in code sample size since both source code size and project size, in general, since most of the metrics in both categories showed an upward trend during the evolution. So, the increase in both number of classes and number of files are not result of the splitting of existing classes into different classes and files. Regarding the increase in the number of classes and files, and based on analysis made of the source code of the analyzed code samples, we believe



AESWrappingRunner.java
KeyStoreExampleRunner.java
KeyUtilitiesRunner.java
LoginRunner.java
SignThreadedRunner.java
SymmetricKeys.java

(a) Files in 2018.



AESCTREncryptDecryptRunner.java
AESGCMEncryptDecryptRunner.java
AESGCMWrappingRunner.java
AESWrappingRunner.java
AsymmetricKeys.java
CBCEncryptDecryptRunner.java
ClientFailureRunner.java
CloudHSMKeyStoreExampleRunner.java
CustomKeyAttributesRunner.java
DESedeECBEncryptDecryptRunner.java
ECBEncryptDecryptRunner.java
ECDHOperationsRunner.java
ECOperationsRunner.java
HMACOperationsRunner.java
KeyStoreExampleRunner.java
KeyUtilitiesRunner.java
LoginRunner.java
RSAImportKey.java
RSAOperationsRunner.java
RSAWrappingRunner.java
SignThreadedRunner.java
SymmetricKeys.java

(b) Files in 2020.

Figure 6.3: Files from aws-cloudhsm-jce-examples in 2018 and 2020 (AWS code sample).

that the reason for the upward trend is mainly due to the addition of new features and new usage scenarios, as pointed out in the discussion about source code size. It is interesting to note that code sample maintainers try to implement these new features and scenarios by avoiding to touch existing code (i.e., existing classes and files). The problem is that this attempt by maintainers to not change existing code causes the number of classes and files to increase. These new classes and files may increase the effort of clients resorting on code samples to understand them. This is not a good practice in code samples. This increase may show that organizations probably do not have a process for creating and maintaining code samples. Maintainers, based on their knowledge of good programming practices and organizational culture, carry out the evolution of code samples but do not pay attention to especially important properties when it comes to code samples. Therefore, this attempt to improve modularization and avoid changes may be good for conventional projects but it also may be an anti-pattern to code samples. For instance, the code sample aws-cloudhsm-jce-examples provided by AWS, starts with six Java files in 2018 and presents 22 in 2020. Besides that, of 22 files, only four remained throughout the evolution in these two years.

### *OO metrics*

We analyze the results obtained in six CK metrics for OO design [17]. Analyzing the results of the Mann-Kendall trend test presented in Table 6.2, we can observe that only WMC (+49%) has an upward trend in the majority of the code samples analyzed, while NOC, DIT, RFC, CBO, and LCOM have no trend in most of the code samples.

Discussion: Analyzing the results, we can initially observe that the metrics related to the class hierarchy did not present a definite trend for most code samples. For code samples, it is interesting that the DIT value remains stable, as greater depth in the inheritance tree means greater complexity of the project. This lack of trend is believed to have occurred because code samples have simpler designs and do not require complex hierarchies. Regarding the NOC, although an increasing number indicates greater reuse of the code, but it is expected that this will not occur in code samples. A higher NOC value may also indicate greater complexity in understanding the code sample, as more classes will be created. That is, a larger NOC could be an anti-pattern in the context of code samples. Regarding the values of LCOM and CBO metrics, it should be noted that most code samples do not show a trend. Through our observations, we found many classes with few or no class attributes making the metric obtain a value of zero over time. Finally, the WMC metric has an upward trend in its values for most code samples. Such metrics are related to the class complexity of a software project. Similar to points already presented in previous discussions, it is believed that this increase is related to the code sample maintainers' attempting not to alter existing code when evolution is made, as well as the lack of specific tasks for code samples to verify the quality of the code within organizations. In summary, some of the CK metrics showed an increasing trend, thus indicating a software decay [76]. This decay may hinder code samples maintenance as well as the impact on clients' understanding of code samples. Figure 6.4 shows a representative example of a method's WMC increasing in Android's code sample android-testdpc during an evolution.

### *Code Complexity*

According to Lehman's law of software evolution [48] and Jacobson *et al.*'s software entropy [39], software projects become more complex and harder to maintain. However, our results, unexpectedly, show a decreasing trend in the complexity for 7 out of 9 metrics in the majority of the code samples.

Discussion: The unexpected result in which a decreasing trend in the complexity is noted for most metrics shows that code samples are types of software

```
//KioskModeActivity.java
@Override
public void onItemClick(AdapterView<?> parent, View view,
int position, long id) {
    if (getPackageName().equals(getItem(position))){
        onBackdoorClicked();
        return;
    }
    PackageManager pm = getPackageManager();
-   startActivity(pm.getLaunchIntentForPackage(getItem(position)));
+   Intent launchAppIntent;
+   String appPackage = getItem(position);
+
+   if (Util.isRunningOnTvDevice(getContext())){
+       launchAppIntent = pm
+           .getLeanbackLaunchIntentForPackage(appPackage);
+   } else {
+       launchAppIntent = pm.
+           getLaunchIntentForPackage(appPackage);
+   }
+   startActivity(launchAppIntent);
}
```

Figure 6.4: Change affecting WMC in android-testdpc (Android code sample.

projects with particularities and deserve a different look. As code samples have an educational purpose, perhaps the main characteristic that comes to mind when thinking about the source code of these projects is that they should not be complex. Therefore, we believe that organizations focus on keeping controlled or decreasing code complexity as a way to ensure the quality of code samples. This kind of concern contrasts with the form code samples evolve if we look at other categories of metrics in the previous analyses. Furthermore, it is essential to emphasize that code samples never present complex examples of functionality usage. That is, if we look at the implementation logic, the complexity tends to get controlled. And as new scenarios or new functionality are required, it is expected that other code samples will be generated instead of using the same.

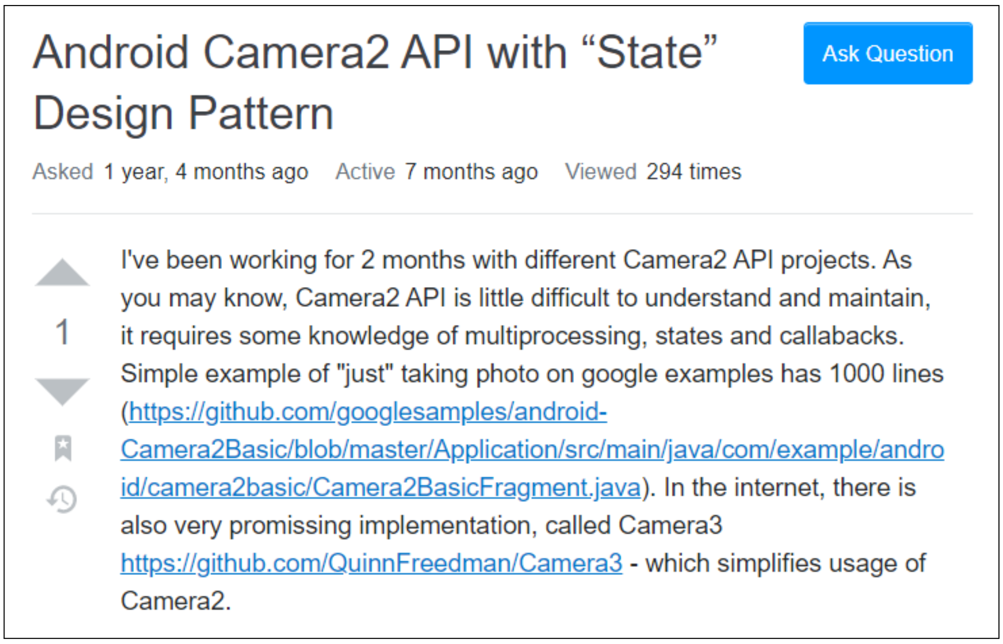
Lesson Learned 2: Unlike conventional software projects, code complexity of code samples is decreasing throughout evolution. However, they are becoming larger and less readable.

### 6.2.3 *(RQ3) Does the evolution of code samples may impact clients' questions?*

In this section, we present the results obtained by correlating the evolution metrics analyzed in RQ1 and the Stack Overflow questions about code samples. The results of the correlation for each of the metrics are presented in Table 6.2. The number of code samples presenting positive (+), negative (-), and inconclusive (?) correlations are shown.

*Source code size*

It is known that there is an upward trend in 10 out of 16 metrics of source code size (Section 6.2.2). As these results go against guidelines and good practices of code samples, we hypothesize there is also an increase in clients' questions. Looking at Table 6.2 we notice that the number of code samples with a positive correlation (i.e., trends of metrics and questions in the same direction) happens in 7 metrics showing a strong correlation with the increase in the number of SO questions related to code samples. The other 9 metrics presented an inconclusive correlation with clients' questions.


Android Camera2 API with "State" Design Pattern

Asked 1 year, 4 months ago Active 7 months ago Viewed 294 times

I've been working for 2 months with different Camera2 API projects. As you may know, Camera2 API is little difficult to understand and maintain, it requires some knowledge of multiprocessing, states and callbacks. Simple example of "just" taking photo on google examples has 1000 lines (https://github.com/googlesamples/android-Camera2Basic/blob/master/Application/src/main/java/com/example/android/camera2basic/Camera2BasicFragment.java). In the internet, there is also very promissing implementation, called Camera3 https://github.com/QuinnFreedman/Camera3 - which simplifies usage of Camera2.

Figure 6.5: Question complaining about the high LOC number.

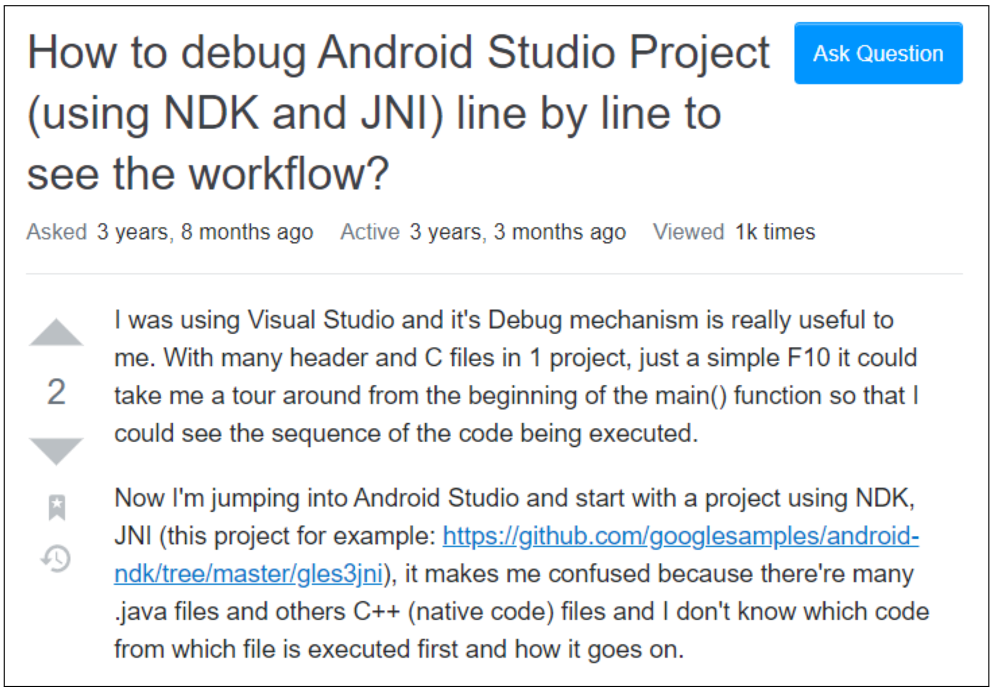
Discussion: The results obtained correlating source code size and clients' questions are somehow expected. An increase in the source code size is likely to increase the clients' efforts to understand the code sample. Figure 6.5 shows an example of a question where the client complains about the number of lines of code in an Android code sample. An interesting behavior is that almost all metrics related to lines of code (exception made to CLOC) presented a positive correlation. Considering this fact with the no trend of the metrics related to the number of methods (LM, PM, NPM, and MPRM), we can say that methods are becoming longer throughout evolution. In conventional projects, a long method is a well-known code smell that brings collateral effects such as software decay, increases in the maintenance effort, and difficulty in understanding. We believe that this may be one of the causes of questions related to code samples. In addition, considering the results obtained, code size metrics related to lines of code have proven to be an important indicator to be tamed during the evolution.

### *Source code readability*

The SAR and CCR metrics related to the source code readability presented a downward trend in our analysis (Section 6.2.2). This behavior may indicate that the code samples became harder to read and understand over time. So, in this case, we expected a negative correlation with the number of questions. In other words, as the readability decreases, the number of questions should increase. The correlation results showed this behavior only in the SAR metric. Both CCR and BWR showed an inconclusive correlation with the clients' questions.
<u>Discussion:</u> It is interesting to notice that one of the readability metrics SAR showed the expected negative correlation with the number of questions. As in conventional projects, it is expected that the readability decreases as the software evolves. This means that code samples evolution is showing the same behavior as conventional software projects in terms of readability. Probably, organizations are not giving due attention to a vital indicator when it comes to code samples. It is worth noting that there is criticism regarding readability metrics. For instance, Posnett *et al.* assessed this metric BWR and identified several weaknesses in its statistical modeling [79]. Despite that, we believe, based on our analysis, that some properties considered by these metrics are indeed relevant to clients referring to code samples.

### *Project size*

There is an upward trend in the values of 2 out of 3 metrics related to project size over time (Section 6.2.2). This upward trend is not advisable for conventional projects as it may increase maintenance effort [17]. In the context of code samples, such an increase in project size (number of Java files and number of classes) could mean an increase in questions related to code samples due to the number of files and/or classes that should be analyzed.
<u>Discussion:</u> Considering the results, it is possible to notice that the number of classes is the only metric that has a correlation (positive) with questions of most of the code samples. It is interesting to notice that code samples often present more than one class per file. This result must be analyzed in other studies to understand the need (or not) to implement some classes in the code samples. It is important to understand if good practices of conventional software projects fit in code samples. For instance, a modularization of a code into


How to debug Android Studio Project (using NDK and JNI) line by line to see the workflow?

Asked 3 years, 8 months ago Active 3 years, 3 months ago Viewed 1k times

2

I was using Visual Studio and it's Debug mechanism is really useful to me. With many header and C files in 1 project, just a simple F10 it could take me a tour around from the beginning of the main() function so that I could see the sequence of the code being executed.

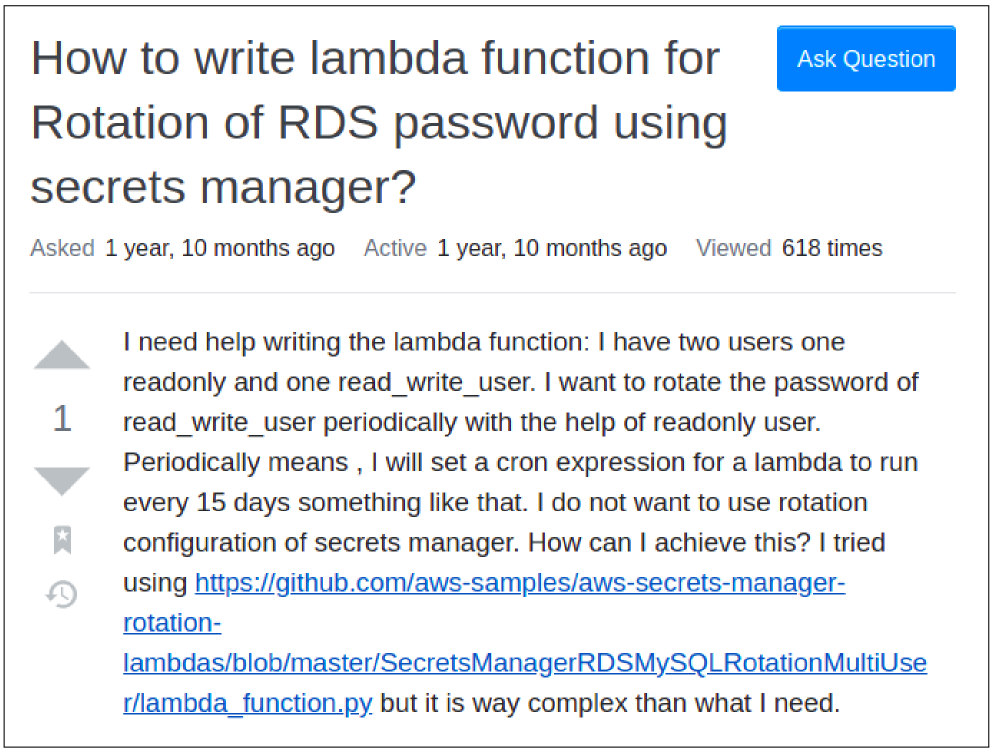
Now I'm jumping into Android Studio and start with a project using NDK, JNI (this project for example: https://github.com/googlesamples/android-ndk/tree/master/gles3jni), it makes me confused because there're many .java files and others C++ (native code) files and I don't know which code from which file is executed first and how it goes on.

Figure 6.6: Client complaining about the number of files.

a new class in the context of a conventional project to improve maintainability may be more harmful to a code sample, where the source code should be as simple as possible.

### *OO metrics*

The authors of the CK metrics suite claim that the proposed OO metrics can indicate code complexity, detect design flaws and predict external software qualities such as software defects, testing, and maintenance effort in conventional software projects [17]. In our results about code samples trying to correlate them with clients' questions, all metrics but WMC presented an inconclusive result. In other words, just the WMC has proven to be a good indicator of clients' questions when it comes to code samples since most of the code samples presented a positive correlation with the number of questions.
Discussion: The CK metrics measure various properties related to the project's OO design. Code samples have different properties when compared to conventional software projects. For example, they must be simple and not present high complexity. As we observed in the results presented in Section 6.2.1, some metrics showed an upward trend meaning a code sample decay over time. However, when analyzing the relationship between these metrics and the clients' questions, we noticed that only the WMC metric had a correlation with clients' questions. Such a metric is of paramount importance in the context of code samples, as it represents the complexity of a class. It is believed that a more complex class can generate more clients' questions. Added to this


How to write lambda function for Rotation of RDS password using secrets manager?

Asked 1 year, 10 months ago Active 1 year, 10 months ago Viewed 618 times

I need help writing the lambda function: I have two users one readonly and one read_write_user. I want to rotate the password of read_write_user periodically with the help of readonly user. Periodically means , I will set a cron expression for a lambda to run every 15 days something like that. I do not want to use rotation configuration of secrets manager. How can I achieve this? I tried using https://github.com/aws-samples/aws-secrets-manager-rotation-lambdas/blob/master/SecretsManagerRDSMySQLRotationMultiUser/lambda_function.py but it is way complex than what I need.

Figure 6.7: Client complaining about the code complexity.

is the fact that there is an increasing number of classes during the evolution of code samples. In other words, the number of classes is growing and they are becoming more complex. Figure 6.7 illustrates an example of a question where a client complains about the complexity of an AWS code sample. Analyzing the mentioned file, we found more than 25 decision points which may be one of the main reasons for the question to be made by the client. Regarding the RFC and CBO metrics that showed an upward trend in the results of RQ1, but did not present a conclusive correlation in the results related to RQ2, we believe that such metrics may interfere more in code samples maintenance tasks than in clients' understanding.

### *Complexity metrics*

Our results show that in the set of complexity metrics considered in our study, only the EC metric showed an unexpected negative correlation with the clients' questions related to code samples, while the other metrics showed an inconclusive correlation.

Discussion: Complexity is one of the first properties that comes to mind when one thinks about code understandability. The results presented in RQ1 showed that organizations may be driving efforts to this property when they evolve code samples since 7 out of 9 metrics showed a decreasing trend. However, the correlation test showed that only one of these metrics is correlated to clients' questions. Even worse, a negative correlation is shown with the EC metric, while the complexity decreases, the questions increase. This fact showed us that (1) organizations may be driving efforts to the wrong property if one considers only clients' questions, and (2) only taming complexity is not enough to minimize clients' questions; this property should be considered with other metrics such as code size, project size, and readability.

Lesson Learned 3: Careless evolution may be causing clients' questions, especially when indicators of size and readability are not observed.

## 6.3 Implications

**More code, less samples:** Our results show that code samples seem to evolve like any other conventional software. Most of the analyzed metrics presented an upward trend. However, a key difference between conventional software projects and code samples is that conventional projects aim at the correct execution of their functionalities at the expense of code quality. That is, the main goal when a conventional project is implemented is that it runs the required functionalities in a correct way. In the case of code samples, although it is important to execute the required functionalities, code quality plays a key role. Putting aside code samples peculiarities can make clients give up on using code samples and look for other sources of information. Or even worse, clients can give up on using the organization's products and migrate to competing products. Thus, it is important that the development of code samples is not adapted to the routine of development of conventional projects in organizations. In fact, code samples should have specific activities in the development life cycle within organizations so that the focus can be on the quality of the project and, especially, the source code.

**Our differences are only skin deep, but our same goes down to the bone:** In this exploratory study, we analyzed code samples from four different organizations. Although it is outside our scope to consider the differences between organizations when analyzing the data we noticed differences in the way of dealing with code samples within the same organization and between organizations. For example, Spring Boot's code samples appear to be more standardized than each other while Android's are very different from each other. When comparing the code samples from different organizations, we notice that some indicators that seem to be important for one organization are not important for another. There are several factors that can lead to these differences, for example, the number of different clients working on the code samples, acceptance policies for pull requests from external clients (outside of the organization), the complexity of the products presented by the code samples, the target audience of each code sample, organization priorities, among others. Thus, it is important (i) to understand how organizations deal with code samples in practice, and (ii) to define a set of good practices that must be followed so that there is at least standardization between code samples from the same organization. In this way, it is possible to have code samples more similar in terms of quality as well as characteristics and priorities defined by organizations.

**Size does matter (in code samples):** When we look at the possible impacts of code samples evolution on clients' questions, we noticed that not all the metrics analyzed are important. Metrics related to OO design as well as the readability metric were not conclusive as to their results (except the WMC metric). However, the code size and project size metrics (except the number of files) were found to be related to the number of clients' questions. That is, especially when the metrics related to the size increased, the questions of the clients also increased. Thus, although more specific studies are needed, it appears that the increase in the size of the code sample, as well as the increase in cyclomatic complexity, can have an impact on the growth of clients' questions. Knowing this, organizations can direct their efforts to keep these metrics under control in their tasks related to maintenance/evolution in case the goal of organizations is to minimize the clients' questions.

**Natural selection of code samples:** Our results show that code samples can be very different. In specific analyzes of some code samples, it was noted that even with evolution the number of questions from clients regarding these code samples remained unchanged. This behavior can be desirable when the evolution is well done and does not raise questions. However, this behavior can also indicate that code samples have lost their usefulness and are no longer a source of information for clients. And one of the reasons for the second case may be the way the code sample has evolved. It should be highlighted that code samples are strategic for organizations as a way to attract clients to use their products. Therefore, organizations should be aware of how their code samples evolve or their risk to become outdated and useless.

**Pull Request is the approach, maintenance is the game:** We found that a wide part of code sample maintenance is the management of Pull Request. Due to the review process, the PR approach helps maintainers to maintain high code quality, and avoid bugs and code smell insert. This approach can encourage code sample clients to contribute with code sample, proposing bug fixes or improvements via PRs and feeling part of the community.

**Sympathy to the devil**: Besides the wide usage of PR approach, we also found a representative part of code pushed directly to the main branch, which means, without a code review. This kind of practice can be a factor to decrease the code sample quality in terms of code.

## 6.4 Threats to Validity

This section discusses the study's limitations based on the four categories of threats to validity described by Wohlin et al. [106]. Each category has a set of possible threats to the validity of an experiment. The following are the main threats and measures taken to reduce the risk.

*Conclusion Validity*: It concerns the relation between treatment and outcome. Here, potential threats arise from *violated assumptions of statistical tests*: the statistical tests used to support the conclusions may have been improperly chosen. To mitigate this threat, we perform a test of normality to identify whether the set of metrics comes from a normal distribution or not. From this result, it was decided which correlation test would be used.

*Intern Validity*: It is the degree to which it is possible to conclude the causal effect of independent variables on dependent variables. A major threat to internal validity is related to *ambiguity about the direction of causal influence*: the number of questions in SO is not necessarily an indication that code samples evolution caused these questions. However, like conventional software projects, we believe that metrics analyzed in this study can lead clients to have problems in code understanding.

*Construct Validity*: It refers to the degree to which inferences can legitimately be made from operationalizations in the study for the theoretical constructs on which those operationalizations were made. A possible threat related to *restricted generalization between constructs* was detected: Java might present specific characteristics when compared to other programming languages and affects our results. This risk cannot be avoided, since we analyzed only Java code. However, it is argued that Java is an important programming language and comprises a large number of code samples on GitHub. Another possible threat to the study validity is related to the use of CK metrics. Despite the criticism, we argue that several fresh studies use CK metrics in similar studies [1, 99, 110].

*External Validity*: Threats associated with external validity concern the degree to which the results can be generalized to the wider classes of subjects from which the experimental work has drawn a sample. A risk related to *the interaction between selection and treatment* was identified: the use of code samples provided by four organizations may have specific aspects when compared to other code samples' organizations. This risk was mitigated by using four relevant organizations.

## 6.5 Related Work

Code samples. Menezes *et al.* [59], which evaluates 233 code samples for Android and Spring Boot, and evaluates aspects of source code, evolution, popularity, and use by customers. Zhang *et al.* [112] propose a novel approach towards enriching API documentation with high-quality code samples and corresponding usage scenarios by leveraging crowd knowledge from Stack Overflow. Zhou *et al.* [114] propose a context-aware code-to-code recommendation tool to automatically analyze the intention of the incomplete code and recommend relevant and reusable code samples in real-time. Picard [77] describes an architecture to manage code samples in documentation, it involves documentation associated with code samples and a testing module. In our study, we are focusing on code sample evolution. The mentioned related studies investigate code sample acquiring, quality or documentation.
Stack Overflow. In [70] is identified which makes the Stack Overflow code examples effective for user learning. Through a qualitative analysis of posts, it was possible to realize that, some artifacts such as, for instance, a good explanation of the code, are fundamental for a solution to be easily understood. In our study, the search for the characteristics of the users of the stack overflow differs from the approach used by [70], here, the information raised about the users is incorporated into the evolutionary analyzes of the code samples generating data on the evolutionary trend of the observed projects. Through the results obtained, it is possible to improve the availability of code samples by organizations. Meldrum *et al.* [57] by performing a systematic mapping study on crowdsourced knowledge on Stack Overflow indicate the need to research quality aspects of the code. There is no study that performs analysis involving code samples evolution and Stack Overflow.
Software evolution. There are countless studies that address the impact of software systems evolution on several quality attributes and code understanding. For example, [41] brings a 10-year analysis of JHot Draw and Rhino software versions, in which object-oriented metrics and their behavior over time are investigated. In addition, they also make an assessment of Lehman's Laws on the systems analyzed. Considering the clients' perceptions about code samples, Breivold *et al.* [11] indicates that there is a lack of precise definition or explanation of the authors' perception of software evolvability.

## 6.6 Conclusion

Finally, we concluded that modification is the most common maintenance activity, followed by PR management. Indeed code samples tend to increase mainly in size, and become less readable and less complex. In addition, there is an impact of the code samples' evolution on clients. In other words, the larger the code sample becomes, the more questions arise from clients. We also noticed that the organizations may be evolving code samples like conventional software projects. It seems that they are not considering code samples in a preventive maintenance plan. Based on this we suggest to practitioners and researchers: (1) the code samples project must have a specific development life cycle within organizations so that the focus can be on the quality of the project and, especially, the source code. Organizations may benefit from using CI/CD tools to control the size, complexity, readability, and design of code samples; (2) establishing strategies to share and communicate about code samples since the way to support the code sample awareness can be impacted by the level of knowledge of the public that will use the code sample; (3) applying software engineering disciplines to design, develop, test, deploy and maintain adapted to code samples goals; (4) using metrics provided in our study as indicators of problems in order to support the "code sample engineering". In future work, the idea is to expand the database of code samples by including new platforms, frameworks, or libraries. In addition, the inclusion of new metrics increases understanding and reinforces the data presented here. Another point we are interested in is conducting qualitative studies with clients and organizations that maintain code samples in order to understand the decisions made about the maintenance and evolution of code samples.

CHAPTER
7

# Conclusion and Future Work

In this work, we seek to explore characteristics of code samples and aspects related to their context in a Software Ecosystem (SECO). In Chapter 3, we evaluate the main similarities and differences between code samples and conventional projects, in terms of source code, evolution and usage. We can conclude that code samples are smaller and simpler than conventional projects. Also, code samples, as conventional projects, rely on tool to automate build and integration, and provide a working environment to their clients. Besides code samples change less frequently, they update to a new platform version faster than conventional projects. In addition, we found that code samples has low usage trough fork approach. In Chapter 4, we go deep to explore code sample usage. We found that the copy and paste approach has low usage in code sample clients. In addition, we noted that clients typically face problems when trying to modify the code sample. And the most client need is code sample improvement. In Chapter 5, we assess the actors that interact with code samples. We highlight that the target audience of code samples can range from inexperienced to experienced client. Also, platforms of different organizations seem to have different target audience. Also, code sample maintainers are aged and unpopular on GitHub, and the majority of them are located in USA, but a set of them are distributed in United Kingdom, China and Germany. Finally, in Chapter 6 we go deep on code sample evolution assessing their maintenance and impact on clients. We found that the most common maintenance activity is modify the code sample repository, but PR management plays a important part of maintenance time. We also found that code samples, become less complex but larger and less readable over time. In addition, careless evolution of code samples may be causing clients questions, especially when size and readability are not observed by organizations.

As this work is a exploratory study about code samples and their relation to SECO context, we need more in-deep studies to complete it. About code sample usage, will be interesting if we assess code sample usage through a survey to understand clients needs and how they use code samples. In addition, we believe that a survey is necessary to better understand the profile of code sample clients. Finally, to better understand the process of code sample maintenance, is necessary a survey with organizations to assess how they handle code sample maintenance.

We provide publicly all data and scripts used in this research in `https://github.com/researchgroupsoma/DISSERTATION-ExploringCodeSamplessAndImpactOnSECO`

# Bibliography

[1] Alsolai, H., Roper, M., e Nassar, D. (2018). Predicting software maintainability in object-oriented systems using ensemble techniques. In 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), páginas 716–721. IEEE. Citado na página 85.

[2] Amazon (2021). Aws samples. Citado nas páginas 8 e 11.

[3] Andersson, J., Larsson, S., Ericsson, M., e Wingkvist, A. (2015). A study of demand-driven documentation in two open source projects. In 2015 48th Hawaii International Conference on System Sciences, páginas 5271–5279. IEEE. Citado na página 8.

[4] Barnaby, C., Sen, K., Zhang, T., Glassman, E., 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, páginas 1353–1364. Citado na página 29.

[5] Bavota, G. e Russo, B. (2016). A large-scale empirical study on self-admitted technical debt. In Working Conference on Mining Software Repositories (MSR), páginas 315–326. IEEE. Citado na página 67.

[6] Benesty, J., Chen, J., Huang, Y., e Cohen, I. (2009). Noise reduction in speech processing, volume 2. Springer Science & Business Media. Citado na página 68.

[7] Borges, H., Hora, A., e Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In International Conference on Software Maintenance and Evolution, páginas 334–344. Citado na página 20.

[8] Bosch, J. (2009). From software product lines to software ecosystems. In SPLC, volume 9, páginas 111–119. Citado nas páginas 1 e 5.

[9] Bosu, A., Corley, C. S., Heaton, D., Chatterji, D., Carver, J. C., e Kraft, N. A. (2013). Building reputation in stackoverflow: An empirical investigation. In _2013 10th Working Conference MSR_, páginas 89–92. Citado nas páginas 53 e 61.

[10] Braga, W., Menezes, G., Fontao, A., Hora, A., e Cafeo, B. (2020). Quero lhe usar! uma análise do público alvo de code samples. In _Anais do VIII Workshop de Visualização, Evolução e Manutenção de Software_, páginas 33–40. SBC. Citado na página 3.

[11] Breivold, H. P., Crnkovic, I., e Larsson, M. (2012). A systematic review of software architecture evolution research. _Information and Software Technology_, 54(1):16–40. Citado na página 86.

[12] Brito, A., Valente, M. T., Xavier, L., e Hora, A. (2020). You broke my code: Understanding the motivations for breaking changes in APIs. _Empirical Software Engineering_, 25:1458–1492. Citado na página 34.

[13] Brito, G., Hora, A., Valente, M. T., e Robbes, R. (2018). On the use of replacement messages in API deprecation: An empirical study. _Journal of Systems and Software_, 137:306–321. Citado na página 15.

[14] Brown, P. (2017). State of the union: npm - linux.com. Citado na página 1.

[15] Buse, R. P. e Weimer, W. R. (2008). A metric for software readability. In _Proceedings of the 2008 international symposium on Software testing and analysis_, páginas 121–130. Citado nas páginas 66 e 67.

[16] Buse, R. P. L. e Weimer, W. (2012). Synthesizing api usage examples. In _International Conference on Software Engineering_, páginas 782–792. Citado na página 29.

[17] Chidamber, S. R. e Kemerer, C. F. (1994). A metrics suite for object oriented design. _IEEE Transactions on software engineering_, 20(6):476–493. Citado nas páginas 67, 77, 80, e 81.

[18] Cloud, S. (2021). Spring cloud samples. Citado nas páginas 8 e 11.

[19] Cohen, J. (1960). A coefficient of agreement for nominal scales. _Educational and psychological measurement_, 20(1):37–46. Citado nas páginas 35 e 48.

[20] Corporation, M. (2022). Code example guidelines. https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines. Citado nas páginas 2, 8, 9, 18, 47, 63, e 70.

[21] Cruzes, D. S. e Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. In International Symposium on Empirical Software Engineering and Measurement (ESEM), páginas 275–284. Citado nas páginas 34, 36, e 48.

[22] Duvall, P., Matyas, S. M., e Glover, A. (2007). Continuous Integration: Improving Software Quality and Reducing Risk. Addison-Wesley Signature Series. Addison-Wesley. Citado nas páginas 9 e 18.

[23] Firouzi, E., Sami, A., Khomh, F., e 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), páginas 1–6. Citado nas páginas 29 e 49.

[24] Fontão, A., Ábia, B., Wiese, I., Estácio, B., Quinta, M., Santos, R. P. d., e Dias-Neto, A. C. (2018). Supporting governance of mobile application developers from mining and analyzing technical questions in stack overflow. Journal of Software Engineering Research and Development, 6(1):1–34. Citado na página 1.

[25] Fontão, A., Cleger-Tamayo, S., Wiese, I., Santos, R. P. d., e Dias-Neto, A. C. (2020). On value creation in developer relations (devrel) a practitioners' perspective. In Proceedings of the 15th International Conference on Global Software Engineering, páginas 33–42. Citado na página 1.

[26] Foundation, P. S. (2021a). Pypi - the python package index. Citado na página 1.

[27] Foundation, T. A. S. (2021b). Apache maven project. Citado na página 1.

[28] German, D. M., Adams, B., e Hassan, A. E. (2013). The evolution of the R software ecosystem. In European Conference on Software Maintenance and Reengineering. Citado na página 15.

[29] GitHub, I. (2021). Example of extraction. Citado na página 65.

[30] Google (2021a). Android samples. Citado nas páginas 2, 8, e 11.

[31] Google (2021b). Google maps samples. Citado nas páginas 2, 8, e 11.

[32] Hamed, K. H. (2008). Trend detection in hydrologic data: the mann–kendall trend test under the scaling hypothesis. Journal of hydrology, 349(3-4):350–363. Citado na página 68.

[33] Heinemann, L., Deissenboeck, F., Gleirscher, M., Hummel, B., e 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, páginas 207–222. Citado nas páginas 29 e 49.

[34] Holmes, R. e Murphy, G. C. (2005). Using structural context to recommend source code examples. In International Conference on Software Engineering, páginas 117–125. Citado nas páginas 28, 29, 48, e 49.

[35] Hora, A., Robbes, R., Valente, M. T., Anquetil, N., Etien, A., e Ducasse, S. (2018). How do developers react to API evolution? a large-scale empirical study. Software Quality Journal, 26(1):161–191. Citado nas páginas 11 e 19.

[36] Hussain, M. e Mahmud, I. (2019). pymannkendall: a python package for non parametric mann kendall family of trend tests. Journal of Open Source Software, 4(39):1556. Citado na página 68.

[Inc] Inc, S. E. Stack exchange data explorer. Citado na página 68.

[38] Inc, S. E. (2022). What is reputation? how do i earn (and lose) it? https://stackoverflow.com/help/whats-reputation. Citado na página 53.

[39] Jacobson, I., Christerson, M., Jonsson, P., e Övergaard, G. (1992). Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley, Reading. Citado nas páginas 63 e 77.

[40] Jansen, S., Cusumano, M. A., e Brinkkemper, S. (2013). Software ecosystems: analyzing and managing business networks in the software industry. Edward Elgar Publishing. Citado nas páginas 5 e 6.

[41] Johari, K. e Kaur, A. (2011). Effect of software evolution on software metrics: an open source case study. ACM SIGSOFT Software Engineering Notes, 36(5):1–8. Citado na página 86.

[42] Keivanloo, I., Rilling, J., e Zou, Y. (2014). Spotting working code examples. In International Conference on Software Engineering, páginas 664–675. Citado na página 29.

[43] Kendall, M. e Gibbons, J. D. (1990). Rank Correlation Methods. A Charles Griffin Title, 5 edition. Citado na página 68.

[44] Konstantopoulos, D., Marien, J., Pinkerton, M., e Braude, E. (2009). Best principles in the design of shared software. In International Computer Software and Applications Conference, páginas 287–292. Citado nas páginas 1, 28, e 48.

[45] Kude, T., Huber, T., e Dibbern, J. (2018). Successfully governing software ecosystems: Competence profiles of partnership managers. IEEE software, 36(3):39–44. Citado nas páginas 1 e 7.

[46] Kula, R. G., German, D. M., Ouni, A., Ishio, T., e Inoue, K. (2018). Do developers update their library dependencies? Empirical Software Engineering, 23(1):384–417. Citado nas páginas 11 e 19.

[47] Kurmangali, A., Rana, M. E., e Ab Rahman, W. N. W. (2022). Impact of abstract factory and decorator design patterns on software maintainability: Empirical evaluation using ck metrics. In 2022 International Conference on Decision Aid Sciences and Applications (DASA), páginas 517–522. IEEE. Citado na página 67.

[48] Lehman, M. M. (1996). Laws of software evolution revisited. In European Workshop on Software Process Technology, páginas 108–124. Springer. Citado nas páginas 15, 63, 70, e 77.

[49] Lethbridge, T. C., Singer, J., e Forward, A. (2003). How software engineers use documentation: The state of the practice. IEEE Software, páginas 35–39. Citado na página 18.

[50] Mandelin, D., Xu, L., Bodík, R., e Kimelman, D. (2005). Jungloid mining: Helping to navigate the api jungle. In Conference on Programming Language Design and Implementation, páginas 48–61. Citado nas páginas 28 e 48.

[51] Manikas, K. (2016). Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software, 117:84–103. Citado nas páginas 1, 5, e 7.

[52] Manikas, K. e Hansen, K. M. (2013). Software ecosystems–a systematic literature review. Journal of Systems and Software, 86(5):1294–1306. Citado nas páginas 5, 7, 51, 54, e 63.

[53] Mann, H. B. (1945). Nonparametric tests against trend. Econometrica: Journal of the econometric society, páginas 245–259. Citado na página 68.

[54] Martin, R. C. (2009). Clean code: a handbook of agile software craftsmanship. Pearson Education. Citado nas páginas 17, 66, e 73.

[55] McDonnell, T., Ray, B., e Kim, M. (2013). An empirical study of API stability and adoption in the Android ecosystem. In International Conference on Software Maintenance, páginas 70–79. Citado nas páginas 11 e 19.

[56] Meldrum, S., Licorish, S. A., Owen, C. A., e Savarimuthu, B. T. R. (2020). Understanding stack overflow code quality: A recommendation of caution.

Science of Computer Programming, 199:102516. Citado nas páginas 29 e 49.

[57] Meldrum, S., Licorish, S. A., e Savarimuthu, B. T. R. (2017). Crowd-sourced knowledge on stack overflow: A systematic mapping study. In Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, páginas 180–185. Citado na página 86.

[58] Menai, M. E. B. e Al-Hassoun, N. S. (2010). Similarity detection in java programming assignments. In 2010 5th International Conference on Computer Science & Education, páginas 356–361. IEEE. Citado na página 33.

[59] Menezes, G., Cafeo, B., e Hora, A. (2019). Framework code samples: How are they maintained and used by developers? In 2019 ACM/IEEE International Symposium ESEM, páginas 1–11. IEEE. Citado nas páginas 2, 3, 8, 9, 11, 15, 16, 17, 18, 19, 20, 30, 31, 32, 51, 61, 63, 70, e 86.

[60] Menezes, G., Cafeo, B., e Hora, A. (2022). How are framework code samples maintained and used by developers? the case of android and spring boot. Journal of Systems and Software, 185:111146. Citado na página 3.

[61] Meyer, M. (2014). Continuous integration and its tools. IEEE Software, 31(3):14–16. Citado nas páginas 9 e 18.

[62] Microsoft (2021a). Azure samples. Citado nas páginas 8 e 11.

[63] Microsoft (2021b). Microsoft samples. Citado nas páginas 2 e 11.

[64] Montandon, J. E., Borges, H., Felix, D., e Valente, M. T. (2013). Documenting apis with examples: Lessons learned with the apiminer platform. In Working Conference on Reverse Engineering, páginas 401–408. Citado na página 29.

[65] Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., e Marcus, A. (2015a). How Can I Use this Method? In International Conference on Software Engineering, páginas 880–890. Citado na página 29.

[66] Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., e Marcus, A. (2015b). How can I use this method? In International Conference on Software Engineering, páginas 880–890. Citado na página 67.

[67] Morrison, P. e Murphy-Hill, E. (2013). Is programming knowledge related to age? In Companion to the Working Conference MSR, páginas 1–4. Citeseer. Citado na página 53.

[68] Moser, S. e Nierstrasz, O. (1996). The effect of object-oriented frameworks on developer productivity. Computer, 29(9). Citado nas páginas 1, 28, e 48.

[69] Movshovitz-Attias, D., Movshovitz-Attias, Y., Steenkiste, P., e Faloutsos, C. (2013). Analysis of the reputation system and user contributions on a question answering website: Stackoverflow. In 2013 IEEE/ACM International Conference on ASONAM 2013, páginas 886–893. IEEE. Citado na página 61.

[70] Nasehi, S. M., Sillito, J., Maurer, F., e Burns, C. (2012). What makes a good code example?: A study of programming q a in stackoverflow. In 2012 28th IEEE International Conference on Software Maintenance (ICSM), páginas 25–34. IEEE. Citado na página 86.

[71] Nguyen, P. T., Di Rocco, J., Di Ruscio, D., Ochoa, L., Degueule, T., e 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), páginas 1050–1060. IEEE. Citado na página 29.

[72] Nguyen, T., Vu, P., e 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, páginas 1027–1038. Citado nas páginas 29 e 48.

[73] Niu, H., Keivanloo, I., e Zou, Y. (2017). Learning to rank code examples for code search engines. Empirical Software Engineering, 22(1):259–291. Citado na página 29.

[74] Nuñez-Varela, A. S., Pérez-Gonzalez, H. G., Martínez-Perez, F. E., e Soubervielle-Montalvo, C. (2017). Source code metrics: A systematic mapping study. Journal of Systems and Software, 128:164–197. Citado na página 67.

[75] Nybom, K., Ashraf, A., e Porres, I. (2018). A systematic mapping study on api documentation generation approaches. In 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), páginas 462–469. IEEE. Citado na página 8.

[76] Olague, H. M., Etzkorn, L. H., e Cox, G. W. (2006). An entropy-based approach to assessing object-oriented software maintainability and degradation-a method and case study. In Software Engineering Research and Practice, páginas 442–452. Citado na página 77.

[77] Picard, A. (2014). Managing code samples in documentation. US Patent 8,694,964. Citado na página 86.

[78] Poshyvanyk, D. e and, A. M. (2006). Jiriss - an eclipse plug-in for source code exploration. In International Conference on Program Comprehension, páginas 252–255. Citado nas páginas 28 e 48.

[79] Posnett, D., Hindle, A., e Devanbu, P. (2011). A simpler model of software readability. In Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, pagina 73–82, New York, NY, USA. Association for Computing Machinery. Citado na página 80.

[80] Raemaekers, S., van Deursen, A., e Visser, J. (2012). Measuring software library stability through historical version analysis. In International Conference on Software Maintenance, páginas 378–387. Citado nas páginas 1, 28, e 48.

[81] Robillard, M. P. (2009). What makes apis hard to learn? answers from developers. IEEE software, 26(6):27–34. Citado nas páginas 1 e 8.

[82] Robillard, M. P. e DeLine, R. (2011). A field study of api learning obstacles. Empirical Software Engineering, 16(6):703–732. Citado nas páginas 1, 8, e 33.

[83] Romano, J., Kromrey, J. D., Coraggio, J., e Skowronek, J. (2006). Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In Florida Association of Institutional Research, páginas 1–33. Citado na página 20.

[84] Roy, C. K. e Cordy, J. R. (2010). Near-miss function clones in open source software: An empirical study. Journal of Software: Evolution and Process, 22(3):165–189. Citado nas páginas 29 e 49.

[85] Sahavechaphan, N. e Claypool, K. (2006). Xsnippet: Mining for sample code. In Conference on Object-oriented Programming Systems, Languages, and Applications, páginas 413–430. Citado nas páginas 28, 29, 48, e 49.

[86] Scalabrino, S., Linares-Vásquez, M., Oliveto, R., e Poshyvanyk, D. (2018). A comprehensive model for code readability. Journal of Software: Evolution and Process, 30(6):e1958. Citado na página 67.

[87] Scitools (2021). Understand documentation. Citado nas páginas 66 e 67.

[88] Shapiro, S. S. e Wilk, M. B. (1965). An analysis of variance test for normality (complete samples). Biometrika, 52(3/4):591–611. Citado na página 68.

[89] Sheskin, D. J. (2003). Handbook of parametric and nonparametric statistical procedures. Chapman and Hall/CRC. Citado na página 20.

[90] Sillito, J., Maurer, F., Nasehi, S. M., e Burns, C. (2012). What Makes a Good Code Example?: A Study of Programming Q&A in StackOverflow. In International Conference on Software Maintenance, páginas 25–34. Citado nas páginas 29 e 49.

[91] Sindhgatta, R. (2006). Using an information retrieval system to retrieve source code samples. In International Conference on Software Engineering, páginas 905–908. Citado nas páginas 28 e 48.

[92] Spearman, C. (1904). The proof and measurement of association between two things. The American Journal of Psychology, 15(1):72–101. Citado na página 68.

[93] Spring (2021a). Oracle. Citado nas páginas 2 e 8.

[94] Spring (2021b). Spring samples. Citado nas páginas 2, 8, e 11.

[95] Spring (2022). Spring | guides. Citado na página 8.

[96] Stylos, J. e Myers, B. A. (2006). Mica: A web-search tool for finding api components and examples. In Visual Languages and Human-Centric Computing (VL/HCC'06), páginas 195–202. IEEE. Citado nas páginas 1 e 8.

[97] Subramanyam, R. e Krishnan, M. S. (2003). Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. IEEE Transactions on software engineering, 29(4):297–310. Citado na página 67.

[98] Tian, Y., Thung, F., Sharma, A., e Lo, D. (2017). Apibot: question answering bot for api documentation. In 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), páginas 153–158. IEEE. Citado nas páginas 1 e 8.

[99] Trautsch, A., Herbold, S., e Grabowski, J. (2020). Static source code metrics and static analysis warnings for fine-grained just-in-time defect prediction. In 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), páginas 127–138. IEEE. Citado na página 85.

[100] Trier, M. (2021). Gitpython documentation. Citado na página 68.

[101] Twitter (2021). Twitter samples. Citado nas páginas 2, 8, e 11.

[102] Uddin, G., Khomh, F., e Roy, C. K. (2020). Mining api usage scenarios from stack overflow. Information and Software Technology, 122:106277. Citado na página 29.

[103] Uddin, G. e Robillard, M. P. (2015). How api documentation fails. IEEE Software, 32(4):68–75. Citado na página 1.

[104] Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., e 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, páginas 805–816. Citado nas páginas 9 e 18.

[105] Wareham, J., Fox, P. B., e Cano Giner, J. L. (2014). Technology ecosystem governance. Organization science, 25(4):1195–1215. Citado na página 7.

[106] Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., e Wessln, A. (2012). Experimentation in Software Engineering. Springer Publishing Company, Incorporated. Citado nas páginas 27, 47, 60, e 85.

[107] Xavier, L., Brito, A., Hora, A., e 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, páginas 138–147. Citado nas páginas 11 e 19.

[108] Yang, D., Hussain, A., e Lopes, C. V. (2016). From Query to Usable Code: An Analysis of Stack Overflow Code Snippets. In International Conference on Mining Software Repositories, páginas 391–402. Citado nas páginas 29 e 49.

[109] Yang, D., Martins, P., Saini, V., e Lopes, C. (2017). Stack Overflow in Github: Any Snippets There? In International Conference on Mining Software Repositories, páginas 280–290. Citado nas páginas 29 e 49.

[110] Yu, X., Bennin, K. E., Liu, J., Keung, J. W., Yin, X., e Xu, Z. (2019). An empirical study of learning to rank techniques for effort-aware defect prediction. In 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), páginas 298–309. IEEE. Citado na página 85.

[111] Zhang, F., Mockus, A., Zou, Y., Khomh, F., e Hassan, A. E. (2013a). How does context affect the distribution of software maintainability metrics?

In 2013 IEEE International Conference on Software Maintenance, páginas 350–359. IEEE. Citado nas páginas 15, 20, e 67.

[112] Zhang, J., He, J., Ren, Z., Zhang, T., e Huang, Z. (2019). Enriching api documentation with code samples and usage scenarios from crowd knowledge. IEEE Transactions on Software Engineering, PP:1–1. Citado nas páginas 9, 33, 51, 61, e 86.

[113] Zhang, J., Sagar, S., e Shihab, E. (2013b). The evolution of mobile apps: An exploratory study. In Proceedings of the 2013 International Workshop on Software Development Lifecycle for Mobile, páginas 1–8. Citado na página 15.

[114] Zhou, S., Shen, B., e Zhong, H. (2019). Lancer: Your code tell me what you need. In 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), páginas 1202–1205. IEEE. Citado na página 86.

[115] Zhu, Z., Hua, C., Zou, Y., Xie, B., e Zhao, J. (2017). Automatically generating task-oriented api learning guide. In Proceedings of the 9th Asia-Pacific Symposium on Internetware, páginas 1–10. Citado nas páginas 1 e 8.

[116] Zhu, Z., Zou, Y., Xie, B., Jin, Y., Lin, Z., e Zhang, L. (2014). Mining api usage examples from test code. In International Conference on Software Maintenance and Evolution, páginas 301–310. Citado na página 29.