On the Usage of Programming Languages in the iOS Ecosystem

Daniel Domínguez-Álvarez
IMDEA Software Institute
Madrid, Spain

Alessandra Gorla
IMDEA Software Institute
Madrid, Spain

Juan Caballero
IMDEA Software Institute
Madrid, Spain

***Abstract*—This paper studies how developers use different programming languages in the iOS ecosystem by examining 161,883 releases of 25,231 third-party libraries spanning 11 years available through CocoaPods, a popular iOS dependency manager. Our empirical study shows that since its release, Swift has been widely adopted, but most libraries, even recent ones, still use Objective-C as their primary programming language. Looking at a small set of 38 open-source iOS apps, instead, we observe that apps are instead predominantly written in Swift by now. We also observe significant C usage across both libraries and apps. Our results suggest that analysis tools for iOS apps should not only support Swift, but also Objective-C and C code.**

***Index Terms*—iOS, third-party libraries, mobile apps, Swift, Objective-C, CocoaPods**

I. Introduction

The Android ecosystem of mobile apps and third-party libraries has been widely studied. There exist many tools to statically and dynamically analyze Android apps and their behavior, and to automatically identify third-party libraries used by apps [1]–[3]. Moreover, many empirical studies provide insights, among others, on the usage of programming languages by Android developers [4]–[6], and their release and test engineering practices [7]–[10].

This is not the case for iOS. Its ecosystem is quite closed and obscure from many points of views. This is primarily due to the many challenges that researchers have to face when analyzing iOS apps. For example, it is challenging to collect iOS apps from the iTunes store and downloaded app binaries must be decrypted before they can be analyzed. More importantly, apps and third-party libraries can be developed in different programming languages. Native iOS apps and libraries were originally developed using Objective-C, but in June 2014 Apple released the Swift programming language to replace Objective-C. On Apple platforms such as iOS, Swift uses the Objective-C runtime library, which allows Swift, Objective-C, C++, and C code to run within the same program. Thus, iOS apps and libraries may be developed using a single programming language, but can also use a combination of Swift, Objective-C, C++, and C. However, while Swift code can fully interoperate with Objective-C code, the opposite is not true, e.g., pure Objective-C apps cannot not use pure Swift libraries. Furthermore, not all Swift versions are compatible, e.g., Swift 5 and Swift 4 are compatible, but Swift 3 is not.

In this paper we investigate how library and app developers use different programming languages in the iOS ecosystem. We analyze the adoption of Swift over time; the use of Swift, Objective-C, C, C++, and other programming languages; and the adoption of Swift major versions. These questions are fundamental to understand what language analysis tools for iOS should support, e.g., whether there is still a need to support Objective-C and old Swift versions.

Our investigation examines the source code of both iOS libraries and iOS apps. To obtain iOS libraries we leverage CocoaPods, a popular iOS dependency manager. The CocoaPods library repository acts a central place for app developers to identify third-party libraries to use in their apps. Our study examines 25,231 libraries, comprising 161,883 releases spanning over 11 years. These libraries have public GitHub repositories, which allow us to fully access their development history. We also use a small dataset of 38 open-source iOS apps to compare the usage of programming languages between iOS libraries and apps.

This work answers the following research questions:

RQ1: What is the predominant programming language used for developing iOS libraries and apps? We expect Objective-C and Swift to be the predominant languages, with Swift quickly growing since its introduction in June 2014. Indeed, the results show that Swift has been widely adopted. However, as of December 2021 (more than seven years after the release of Swift), 57% of third-party libraries still use Objective-C as their primary programming language, compared to 41% for Swift. On the other hand, 73% of apps use Swift as their main language by now. Thus, Swift adoption by apps has been quicker than adoption by libraries. This is likely explained by the fact that Objective-C libraries can be used by apps written in Swift and Objective-C, while libraries written in Swift can only be used by Swift apps.

RQ2: What programming languages, beyond the predominant one, are used by iOS libraries and apps? iOS allows projects to mix multiple programming languages. Thus, we also analyze all languages (beyond the predominant one) used in the development of libraries and apps. We observe 68 languages in use. Surprisingly, while C rarely is the predominant language in libraries and apps, it is frequently used in both, being second in popularity across apps after Swift, and surprisingly in front of Objective-C. However, C

usage is largely due to C libraries being copied into the source code of libraries and apps and is slowly decreasing over time as those vendored libraries are turned into external dependencies.

RQ3: Is the presence of Objective-C code in libraries a consequence of legacy code? We study the adoption of Swift in libraries created before and after its introduction. The results show that 95% of the libraries released before the Swift introduction never adopted it at all, highlighting the lack of incentive for Objective-C library developers to change language. Surprisingly, 53% of the libraries released after the Swift introduction do not use Swift at all, compared to 45% written purely in Swift. Thus, developers of new libraries still prefer to code them fully in Objective-C. Again, this is likely due to apps written in Swift being able to use Objective-C libraries, while the opposite is not true.

RQ4: Do developers update their libraries according to new Swift releases? Last, we analyze if developers update their libraries to support new Swift releases. We expect well-maintained projects to update to the most recent version of Swift. The results show that on December 2021, 52% of libraries target Swift 5, the latest Swift major version. The 48% libraries still using older Swift versions are largely not maintained, e.g., 88% of libraries using Swift 4 on December 2021 do not have commits in the previous two years. Of the libraries that update Swift, 73% upgrade at major releases.

We conclude that iOS analysis tools should not only support Swift, but also Objective-C and C code, and that it may take long time for Objective-C to be fully replaced by Swift, due to the limited incentive for language replacement in libraries.

II. The CocoaPods Ecosystem

The usage of third-party libraries by iOS apps is often handled by dependency managers. There are three main iOS dependency managers: *CocoaPods*, *Carthage*, and *SwiftPM*. Of these, CocoaPods is the only one using a centralized repository of available library versions, which app developers can search for to use in their apps. To distribute a library version through CocoaPods, the library developer commits to the CocoaPods repository a *podspec*, i.e., a text file containing metadata such as the library name, the version, the source from where the library version can be obtained, and the target iOS and Swift versions. For the purpose of this work we care about the following podspec fields illustrated in Figure 1:

- The *name* field uniquely identifies the library.
- The *source* field contains the location from where the library version can be downloaded such as a Git repository and a specific release tag in the repository.
- The *source_files* field captures the path to the library source files. We use this field to ignore other code (e.g., scripts, test cases) that may also be available in the repository of a library.
- The *pushed_with_swift_version* field optionally specifies the target Swift version for Swift-based libraries.

Most iOS developers use the official XCode GUI to develop their apps. To use CocoaPods, an iOS app developer adds a *podfile* to the app's XCode project. The podfile is a text file listing the libraries the app depends on (with optional constraints on specific library versions or version ranges). CocoaPods selects the library versions that will be installed, i.e., those that satisfy the constraints in the podfile and any additional dependencies those libraries may introduce. Then, it adds a compilation script to the XCode project so that the dependencies are included in the build process.

```
{name: "SwiftTransition",
  source: {
    git: "https://github.com/...",
    tag: "1.4"
  },
  source_files: "SwiftTransition/Classes/**/*",
  pushed_with_swift_version: "3.0"
}
```

Fig. 1. Excerpt of a podspec for the SwiftTransition 1.4 library version.

III. Dataset Collection

For our study, we collect the source code of iOS libraries and iOS apps.

Libraries. We collect metadata on third-party libraries from CocoaPods. As of December 2021, the CocoaPods repository contained 82,785 libraries and 528,409 library versions, From that initial list, we select only library versions whose source field in the podspec is a Git repository with an associated release tag. The use of a Git repository allow us to examine the full development history of the library, not possible for the 71,589 library versions whose source is an HTTP endpoint. The use of non-Git repositories is marginal: 111 library versions for SVN and 41 for Mercurial. Requiring a release tag allows us to obtain the source code snapshot for the specific version from the library repository. This filtering leaves 77,480 libraries and 450,684 library versions.

We try to clone each Git repository hosted in GitHub, filtering out unclonable repositories (e.g., private, dead). Finally, we apply PyDriller [11] on each successfully cloned repository to check out the snapshot for each release tag that corresponds to a library version in CocoaPods. Our final library dataset comprises 25,231 libraries and 161,883 library versions, each version with its own source code snapshot.

We apply cloc[1] to each source code snapshot to extract statistics about the number of lines written in each programming language. We apply cloc only to the folders listed in the source_files field of the library version podspec (see Figure 1) to exclude non-source files (e.g., scripts, examples, test cases).

To analyze Swift version adoption, we build a subset dataset with the library versions that contain a target Swift version in their podspec. This subset comprises 8,276 libraries and 52,375 library versions, 32% of the full library dataset.

iOS apps. We start our app collection from a curated list of open-source iOS apps[2]. From that list, we select apps that

[1]https://github.com/AlDanial/cloc

[2]https://github.com/dkhamsing/open-source-ios-apps

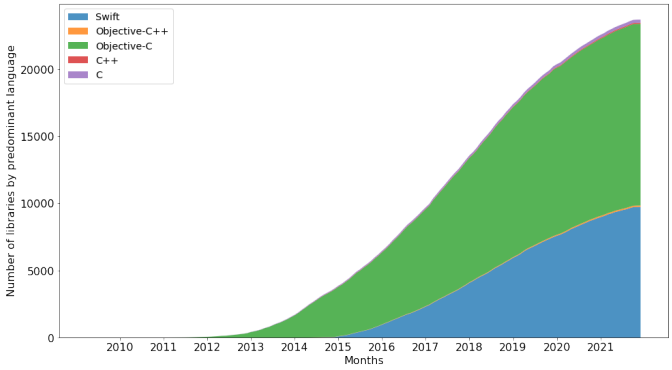

Fig. 2. Predominant programming language in CocoaPods libraries.

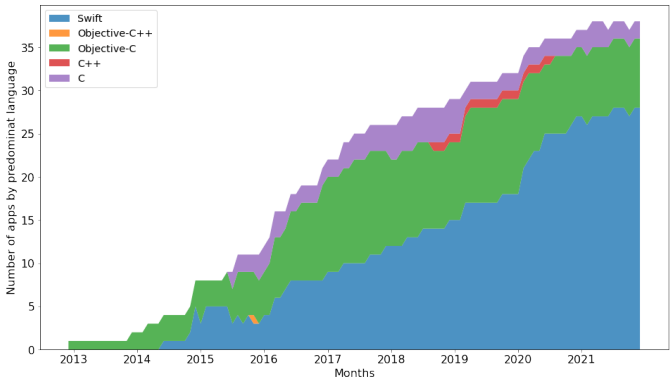

Fig. 3. Predominant programming language in apps dataset.

are hosted in GitHub, have at least 100 GitHub stars, were last updated throughout 2021, are predominantly written in Swift or Objective-C, and for which we can successfully clone their repository. This selection process leaves 38 apps that are popular, maintained, have not been written using frameworks (e.g., React Native, Dart, Cordova), and for which we can obtain their source code. We apply PyDriller to each cloned repository to check out each commit. To obtain statistics about programming languages used, we apply cloc to each commit, ignoring well-known directories from dependency managers in order to exclude the source code of third-party libraries. One limitation is that if the source code of a third-party library was copied into the app's repository instead of using a dependency manager (what we call a *vendored library*), its code will be considered part of the app. The final app dataset contains 204,846 commits for 38 apps.

IV. Empirical Study

This Section answers our four research questions.

A. RQ1: Predominant Programming Language

We first measure the usage of different programming languages over time focusing on the predominant language for each project. In particular, we use the cloc statistics to determine, for every library release, or app commit, the programming language with largest LoC. Figures 2 and 3 show

TABLE I
Top 10 CLOC entries by cumulative LoC across all versions in the library dataset.

Name	Releases	LOC
Objective-C	84,309	214,727,108
Swift	69,216	115,231,160
C/C++ Header	107,104	61,885,536
C	3,272	24,834,241
C++	1,343	19,312,751
JavaScript	281	5,445,147
Objective-C++	2,274	5,101,405
XML	11,368	3,582,887
JSON	1,573	1,218,541
CSS	141	576,513

for each month between April 2009 and December 2021, the number of libraries and apps, respectively, whose most recent release (commit for apps) at the beginning of that month had a specific predominant language. The plots show the following:

- At any point, over 98% of libraries are predominantly written using Objective-C and Swift. Very few libraries are predominantly written using other programming languages such as C and C++. On December 2021, out of 25,231 libraries, 57.12% use Objective-C as the main language, 41.12% use Swift, 0.98% use C, 0.43% use Objective-C++, and 0.35% use C++.
- In contrast, Swift dominates on the apps dataset since mid-2018. On December 2021, Swift is the predominant language for 73.68% apps, followed by Objective-C (21.05%), and C (5.26%). C++ and Objective C++ do not predominate in any app.
- Swift immediately starts to get adopted after its release on June 2014. Swift adoption by libraries grows slowly over time. On December 2021, more than seven years after its first release, Objective-C is still predominant in more than half of the libraries. We believe that the reasons why Swift has not yet surpassed Objective-C usage are twofold. First, library developers may still prefer to develop in Objective-C, since apps written in Swift can use Objective-C libraries, while the opposite is not true. We further analyze this aspect in Section IV-B. Second, developers who published their library before Swift was introduced may not want to take the effort to port the library to a new programming language, given Objective-C is still supported. We explore this research question in Section IV-C.

B. RQ2: Programming Languages Used

Since developers may use different programming languages in the same project, we study the prevalence of each language beyond the predominant one. We thus compute the cumulative LoC for each language for the most recent library version, or app commit, at the beginning of each month. Figures 4 and 5 show the results for libraries and apps, respectively. We observe the following:

- While C was rarely the predominant language in libraries and apps in Section IV-A, it is frequently used in both

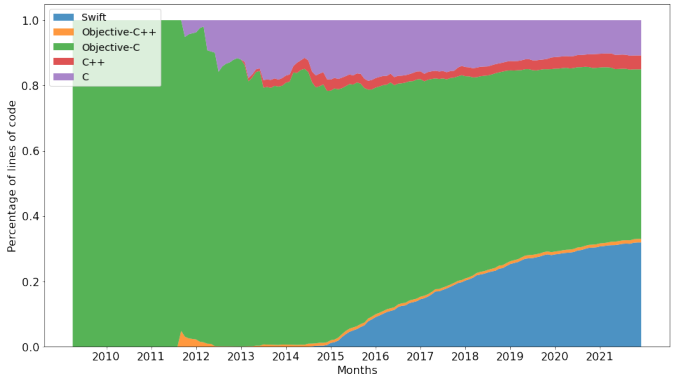

Fig. 4. Programming language LoC across CocoaPods libraries.

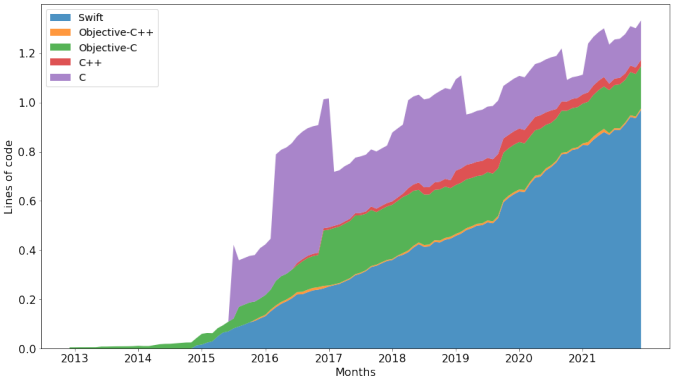

Fig. 5. Programming language LoC across apps.

libraries and apps, being more popular than Objective-C in apps. Still, its usage is slowly decreasing over time.

- Objective C++ had a peak of usage in 2012. After the introduction of Swift, its usage stabilizes: it does not increase, but it does not disappear either.
- On December 2021, 51.92% of library code is written in Objective-C, 31.96% in Swift, 10.81% in C, 4.24% in C++, and 1.07% in Objective-C++.
- Beyond those five languages, there are another 63 languages that developers use in their libraries. Table I shows the top 10 cloc entries in libraries.

These results show that library development and app development follow different strategies in terms of what programming languages to use. They indicate that iOS analysis tools should support both Swift and Objective-C. Furthermore, analysis tools should also support C code, which still has significant use in both libraries (10.81%) and apps (12.84%). The presence of C seems mostly due to the bad practice of including entire externally developed components as vendored libraries in the codebase of some iOS apps, rather than adding them as dependencies. For example, Figure 5 shows a couple of sudden drops in the use of C code in apps. The first drop on early 2017 is due to the removal of the SQLite source code from the codebase of the Firefox iOS app, which was replaced with a pre-compiled SQLite binary instead. Another drop around the end of 2018 is due to a bunch of vendored C libraries that were also removed from the Firefox iOS app codebase, but were this time added as external dependencies.

C. RQ3: Legacy vs New Libraries in CocoaPods

One hypothesis, which would explain why Objective-C is still so prevalent among libraries, is that CocoaPods may include many old, though still usable, libraries originally written in Objective-C, which developers never bothered to port to Swift. To analyze this, we split the library dataset in two: libraries first released before Swift's first release and libraries first released after Swift's introduction.

We classify libraries first released before the introduction of Swift into four disjoint groups. The vast majority of libraries (95.79%) do not adopt Swift at all (i.e., cloc reports 0% Swift LoC). This is likely due to Objective-C libraries being fully compatible with new Swift apps. A second group of 3.48% libraries fully replace old code with Swift over time (i.e., cloc reports versions with 100% Swift LoC). These libraries decided that full rewriting was worth the effort. The remaining libraries introduce Swift, but do not fully replace the old code. We split them in two. 0.53% libraries introduce Swift as a minor language (i.e., cloc reports some Swift, but less than Objective-C) while 0.21% libraries introduce Swift as the predominant language (i.e., cloc reports more Swift than Objective-C).

We similarly classify libraries first released after the introduction of Swift. Of those, 52.96% do not use Swift at all, 44.84% are purely written in Swift, 1.58% use more Swift than Objective-C, and 0.62% use less Swift than Objective-C. These numbers capture a rather surprising result. Developers of new libraries, released after Apple introduced Swift as the official language for the iOS ecosystem, still prefer to develop fully in Objective-C. The main hypothesis regarding this result is that Objective-C libraries can be used by apps written in Swift and Objective-C, while libraries written in Swift can only be used by Swift apps. Thus, in an attempt to make their component compatible with as many apps as possible, developers may have preferred to stick to Objective-C. The opposite happens in apps, where only 7.89% of the apps created after the release of Swift do not use any Swift at all, while 47.37% of them all fully written in Swift. In apps with both languages, Swift tends to be the primary language (34.21%) as opposed to Objective-C (10.53%).

D. RQ4: Update to new Swift Releases

Our last research question focuses on whether libraries are updated whenever there are new Swift releases, expecting maintained libraries to do so. Swift has five major versions, with the 5.6.1 being the current latest version. Figure 6 shows how the Swift target version in the podspec of the libraries in the Swift subset dataset changes over time. We can see that:

- On December 2021, 51.88% CocoaPods libraries are up to date targeting Swift 5.x (i.e., major or minor releases). Among the rest, 28.95% target Swift 4.x and 18.93% target Swift 3.x.

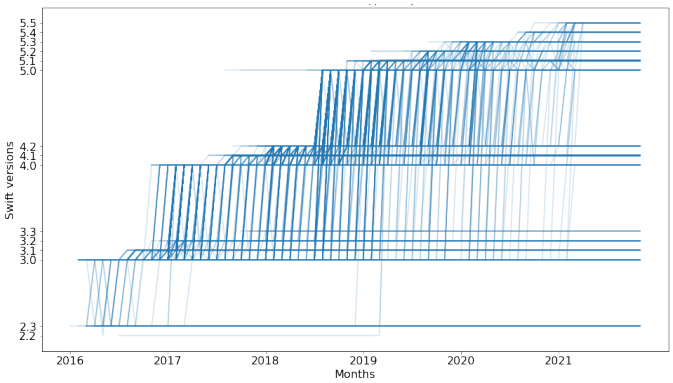

Fig. 6. Support of Swift versions

- 72.82% of libraries that upgrade the target Swift during their lifetime tend to do it upgrading among major releases (e.g., upgrading from Swift 4.0 to 5.0). A quarter of libraries (27.18%) upgrade Swift across minor versions.
- We see many horizontal lines in the plot, which represent libraries that keep the same Swift version for the whole timeline. Libraries stuck with Swift 3 are mostly not maintained (97.70% of these libraries do not have commits in the last 4 years). Similarly, 87.75% of libraries stuck with Swift 4 are not maintained (i.e. they do not have commits in the last 2 years).
- Finally we observe a few jumps back, i.e. libraries that target Swift 5.1 and then change it to 5. We examined the commit messages, but could not find useful insights on why developers did this. We believe it is because they realize that a newer version of Swift breaks the building process due to some dependencies. In fact, most of the times the rollbacks happen within one day.

Given these findings, we conclude that analysis tools should support Swift 4.0 onward, since it is compatible with Swift 5, and some libraries still use old releases of Swift.

V. Related Work

Similarly to what Apple did with Swift, Google in 2019 announced that "Android development would be Kotlin-first". Several studies have examined the adoption of Kotlin over Java in the Android ecosystem [4]–[6], showing is that there has been a rapid adoption of Kotlin, especially among the most popular apps. Our results do not show this fast adoption in the iOS ecosystem, instead, at least among third-party libraries. Rahkema and Pfahl previously studied the CocoaPods ecosystem [12]. However, they focused on studying the dependencies among components and how known vulnerabilities may have an impact on CocoaPods and other package managers for iOS. Finally, other researchers studied the iOS ecosystems from other points of view. Cassee et al. looked at how iOS developers handle errors using the Swift programming language [13], showing that developers tend to react to issues rather than preventing them with proper error handling mechanisms. Rahkema et al. [14] and Habchi et al. [15] both study the prevalence of code smells in iOS apps, showing that iOS apps are quite affected by code smells, although in general they seem less prone to smells than Android apps.

VI. Conclusion and Future Work

This paper presented the first study on how developers use different programming languages in the iOS ecosystem. Our empirical study shows that since its release, Swift has been widely adopted, but developers still use Objective-C as the predominant programming language to develop recent third-party libraries. On a small dataset of 38 open source iOS apps, we instead observe that apps are predominantly written in Swift by now. We also observe significant usage of C code on both libraries and apps, as a secondary language. The main takeaway message of our study is that analysis tools for iOS apps should support not only Swift, but also Objective-C and C code to provide high coverage of the codebase. One caveat of our work is that our app dataset is fairly small. In the future we would like to confirm our results in a larger app dataset. Moreover, we would like to analyze other iOS dependency managers to confirm our observations on CocoaPods libraries.

Code and dataset are available at
https://github.com/0xddom/cocoapods-evolution

Acknowledgments

This work was partially supported by the Spanish Government's SCUM grant RTI2018-102043-B-I00, Grant RYC2020-030800-I funded by MCIN, and the Madrid Regional project BLOQUES S2018/TCS-4339, and gifts from Facebook.

References

[1] Z. Ma, H. Wang, Y. Guo, and X. Chen, "Libradar: Fast and accurate detection of third-party libraries in android apps," in *ICSE*, 2016.
[2] M. Li, P. Wang, W. Wang, S. Wang, D. Wu, J. Liu, R. Xue, W. Huo, and W. Zou, "Large-scale third-party library detection in android markets," *IEEE TSE*, vol. 46, no. 9, pp. 981–1003, 2020.
[3] M. Backes, S. Bugiel, and E. Derr, "Reliable third-party library detection in android and its security applications," in *CCS*, 2016, pp. 356–367.
[4] G. Hecht and A. Bergel, "Quantifying the adoption of kotlin on android stores: Insight from the bytecode," in *MobileSoft*, 2021, pp. 94–98.
[5] M. Peters, G. L. Scoccia, and I. Malavolta, "How does migrating to kotlin impact the run-time efficiency of android apps?" in *SCAM*, 2021.
[6] R. Coppola, L. Ardito, and M. Torchiano, "Characterizing the transition to kotlin of android apps: a study on f-droid, play store, and github," in *WAMA*, 2019, pp. 8–14.
[7] X. Xia, E. Shihab, Y. Kamei, D. Lo, and X. Wang, "Predicting crashing releases of mobile applications," in *ESEM*, 2016, pp. 29:1–29:10.
[8] D. Domínguez-Álvarez and A. Gorla, "Release practices for ios and android apps," in *WAMA*, 2019, pp. 15–18.
[9] M. Nayebi, B. Adams, and G. Ruhe, "Release practices in mobile apps –- users and developers perception," in *SANER*, 2016, pp. 552–562.
[10] D. Domínguez-Álvarez, D. Tomic, and A. Gorla, "Rechan: An automated analysis of android app release notes to report inconsistencies," in *MobileSoft*, 2022, pp. 73–83.
[11] D. Spadini, M. F. Aniche, and A. Bacchelli, "Pydriller: Python framework for mining software repositories," in *ESEC/FSE*, 2018.
[12] K. Rahkema and D. Pfahl, "Dataset: Dependency networks of open source libraries available through cocoapods, carthage and swift PM," in *MSR*, 2022, pp. 393–397.
[13] N. Cassee, G. Pinto, F. Castor, and A. Serebrenik, "How swift developers handle errors," in *MSR*, 2018, pp. 292–302.
[14] K. Rahkema and D. Pfahl, "Empirical study on code smells in ios applications," in *MobileSoft*, 2019, pp. 61–65.
[15] S. Habchi, G. Hecht, R. Rouvoy, and N. Moha, "Code smells in ios apps: How do they compare to android?" in *MobileSoft*, 2017, pp. 110–121.