Poster: Kotlin Assimilating the Android Ecosystem - An Appraisal of Diffusion and Impact on Maintainability

Riccardo Coppola
riccardo.coppola@polito.it
Politecnico di Torino, Turin, Italy

Tommaso Fulcini
tommaso.fulcini@polito.it
Politecnico di Torino, Turin, Italy

Marco Torchiano
marco.torchiano@polito.it
Politecnico di Torino, Turin, Italy

ABSTRACT

Kotlin is a language alternative to Java, introduced in 2011. It promises to address many of Java's limitations and lead to better application maintainability. In 2017, it became a first-class language for Android development with full tool support. We mined a dataset of 2708 Android applications on which we based our study. Our empirical assessment of the diffusion of Kotlin in Android app development shows that it is now used in around 40% of projects. Kotlin adoption has a significant positive effect on code maintainability metrics and in popularity among end-users and developers. Overall, Kotlin appears to be successfully fulfilling its promise of being a better Java for Android development.

KEYWORDS

Software Maintainability, Android Development, Kotlin

ACM Reference Format:
Riccardo Coppola, Tommaso Fulcini, and Marco Torchiano. 2024. Poster: Kotlin Assimilating the Android Ecosystem - An Appraisal of Diffusion and Impact on Maintainability. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643071

1 INTRODUCTION

Kotlin has been developed to address several limitations of the Java language: the handling of *null* values possibly leading to NullPointerExceptions [2]; maintainability, understandability and conciseness; avoiding several other common Java coding pitfalls [2].

Many developers claim that using Kotlin results in better quality code. However, it is unclear how Kotlin code affects the maintainability of codebases in comparison to traditional Java Android apps. The objectives of this paper are manifold: (i) we extend a preliminary analysis of the state of popular repositories of Android applications in terms of Kotlin adoption, building up on a dataset that was originally mined in 2019 [3]; (ii) we compute a set of state-of-the-art software quality and maintainability metrics for Kotlin; (iii) we conducted an empirical analysis to determine the impact of Kotlin adoption on popularity among developer, user ratings of the application, and maintainability of software project.

2 APPROACH

The study answered four research questions: (RQ1) What is the percentage adoption of Kotlin on Android apps on OS (open-source) repositories? (RQ2) What is the pace of the integration of Kotlin? (RQ3) What is the impact of the presence of Kotlin on the popularity and feedback from users and developers? (RQ4) What is the impact of the presence of Kotlin on code maintainability?

2.1 Repository mining

As the source of projects for our study, we selected the F-Droid repository of OS applications because of its wide use in related literature. The repository mining was performed on April 30, 2023, and allowed the collection of an initial set of 3889 Android OS projects. All OS projects extracted from this mining were then paired with the corresponding OS project on GitHub or application on the PlayStore (if existing). We then filtered out for our study the projects that were developed with hybrid non-native frameworks and those that were not updated for more than 45 days.

The final set of repositories consisted of a total of 2708 OS Android projects. 1369 projects were also released on the Google Play Store, 1578 were also published as GitHub repositories, 805 apps were available on all three repositories.

2.2 Analysis procedure

The following metrics were extracted for each project to answer the RQs of the study:

- **Diffusion (RQ1)** : to evaluate diffusion we defined the *Kotlin Presence* ordinal variable with four levels based on the relative quantity of Kotlin code in the whole project: *No Kotlin*; *Kotlin* < 50%; *Kotlin* > 50%; *Only Kotlin.* The lines of code in the Kotlin language were computed using the cloc tool[1].
- **Evolution (RQ2)** : To analyze the evolution of the projects, we took monthly snapshots of each project, by considering the last commit in each frame. Each snapshot was analyzed with the cloc tool. We then divided the projects into different Kotlin adoption groups and reported the ratio of projects in each group with respect to the total number of projects available at any given monthly snapshot.
- **Popularity (RQ3)** : To analyze popularity, we considered two distinct sets of projects: (i) the projects that were present on the PlayStore (regardless of their presence on GitHub), and that were not abandoned (a total of 912 projects): for them, we considered as a popularity metric the average rating (i.e., the *Stars*); (ii) the projects that were available on GitHub (regardless of their presence on the PlayStore), and that were

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

not abandoned: for them, we considered as popularity metrics the number of stars averaged by the project lifespan months and the number of watching accounts.

Maintainability (RQ4) : To analyze maintainability, we considered the set of projects that were updated after October 2017, and we considered the most newly updated repository between the tarball hosted on F-Droid and the last commit on GitHub (if present). We applied the rca (Rust Code Analysis)[2] tool to the codebase. We selected the six metrics that are most related to code maintainability and understandability: CC (McCabe's Cyclomatic Complexity) [4]; Halstead Difficulty [5]; the logarithm of Halstead Effort; MI (Maintainability Index); WMC (Weighted Method per Class) [6]; Cognitive Complexity [7]. For the details of the metrics the interested reader may refer to Ardito et al. [1].

3 RESULTS

By analyzing the total amount of projects, it emerges that 1023 out of 2,708 (37.8%) featured lines of code written in Kotlin. 551 (20.3%) were instead entirely written in Kotlin. The fraction of projects with the presence of Kotlin increases if the most recent update of the project is on F-Droid (38.9%) rather than on GitHub (35.8%).

When Kotlin is adopted, on average the majority of the code is written with it (75.90% LOCs, 77.80% files) hinting at a preference of the developers in using Kotlin rather than Java when the two languages are used in the same project. The interested reader can refer to the online appendix for additional details[3].

Answer to RQ1: 38% of analyzed apps contain Kotlin code.

To answer RQ2, we considered the set of projects not abandoned on GitHub, with last update after October 2017 (967 projects).

For each month, we discarded the projects that were no longer updated in the previous year. The amount of projects with Kotlin steadily increased over the surveyed period: from 6.8% (39 out of 574) to 48% (241 out of 501), whilst there is a visible contraction in the number of projects featuring only Java (-52.2% since October 2017, and -13.4% only on the first four months of 2023).

Restricting the analysis of Kotlin evolution only to the 342 projects with Kotlin that were most recently updated on GitHub, excluding projects that were not updated for the last 12 months a clear trend of increase in the relative amount of Kotlin code inside Android projects can be seen.

Answer to RQ2: since 2017 the proportion of active projects using Kotlin raised from 8% to 48%.

We notice a significant positive effect of having a project fully developed in Kotlin on the average number of stars on the PlayStore platform (average 4.4 mean rating, against average 4.14 mean rating, p-value equal to 1.35e-04). There is no statistically significant difference between the groups "Kotlin minority" and "Kotlin majority" and the reference group "Only Java".

Regarding GitHub stars per project, a statistically significant positive difference in normalized GitHub stars was verified for projects fully developed in Kotlin. Regarding GitHub Watching per project, there is no statistically significant difference between any group and the reference "Only Java" group.

Answer to RQ3: end-users on PlayStore assign ratings 1/4 of star higher to projects with Kotlin; on GitHub, pure Kotlin projects get 8 stars per month more than pure Java ones.

The metrics to answer RQ4 were collected on all the projects that were updated after October 2017. Regarding WMC, Kotlin Majority and Only Kotlin had a statistically significant difference compared to the reference group "No Kotlin". This suggests that the applications of Kotlin tend to have a lower number of weighted methods per class, with an average decrease from 17.03 to 7.47 when only Kotlin is used. Regarding CC, the difference between the groups "Kotlin Majority" and "Only Kotlin" against the reference group "No Kotlin" proved to be statistically significant, with an average decrease from 1.88 to 1.74 and 1.70, respectively. Concerning the log10 of the Halstead Effort, the difference between the groups "Kotlin Majority" and "Only Kotlin" against the reference group "Only Java" proved to be statistically significant, with an average decrease from 5.43 to 4.79 and 4.42, respectively. We observe a similar significant difference for the Halstead difficulty, with a decrease from 29.33 to 11.49 from "No Kotlin" to "Only Kotlin". Regarding MI, all the groups had a statistically significant positive effect compared to the "No Kotlin" reference group. For projects with Kotlin only, we observe an increase from 30.64 to 38.14. Finally, regarding Cognitive Complexity, we observe a statistically significant negative difference only for the "Kotlin Majority" (reduction of 0.44) and "Only Kotlin" (reduction of 0.56) compared to the 1.42 average value for the "No Kotlin" reference group.

Answer to RQ4: consistent improvement of all maintainability related metrics: McCabe CC $\downarrow$ 9.6%, WMC $\downarrow$ 56%, Halstead Effort $\downarrow$ 18.6%, Halstead Difficulty $\downarrow$ 60.6%, Cognitive Complexity $\downarrow$ 39.4%, Maintainability Index $\uparrow$ 24.5%.

REFERENCES

[1] Luca Ardito, Luca Barbato, Marco Castelluccio, Riccardo Coppola, Calixte Denizet, Sylvestre Ledru, and Michele Valsesia. 2020. rust-code-analysis: A Rust library to analyze and extract maintainability information from source codes. *SoftwareX* 12 (2020), 100635.

[2] Subham Bose, Madhuleena Mukherjee, Aditi Kundu, and Madhurima Banerjee. 2018. A comparative study: java vs kotlin programming in android application development. *International Journal of Advanced Research in Computer Science* 9, 3 (2018), 41–45.

[3] Riccardo Coppola, Luca Ardito, and Marco Torchiano. 2019. Characterizing the transition to kotlin of android apps: a study on f-droid, play store, and github. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*. 8–14.

[4] Christof Ebert, James Cain, Giuliano Antoniol, Steve Counsell, and Phillip Laplante. 2016. Cyclomatic Complexity. *IEEE Software* 33, 6 (2016), 27–29. https://doi.org/10.1109/MS.2016.147

[5] T Hariprasad, G Vidhyagaran, K Seenu, and Chandrasegar Thirumalai. 2017. Software complexity analysis using halstead metrics. In *2017 International Conference on Trends in Electronics and Informatics (ICEI)*. 1109–1113. https://doi.org/10.1109/ICOEI.2017.8300883

[6] Wei Li. 1998. Another metric suite for object-oriented programming. *Journal of Systems and Software* 44, 2 (1998), 155–162. https://doi.org/10.1016/S0164-1212(98)10052-3

[7] Jingqiu Shao and Yingxu Wang. 2003. A new measure of software complexity based on cognitive weights. *Canadian Journal of Electrical and Computer Engineering* 28, 2 (2003), 69–74.

[2]https://github.com/mozilla/rust-code-analysis
[3]doi.org/10.6084/m9.figshare.24999962