Comparison of Code Smells in iOS and Android Applications

Kristiina Rahkema[a], Dietmar Pfahl[a]

[a]Institute of Computer Science, University of Tartu, Tartu, Estonia

Abstract

Code smells are patterns indicating bad practices that may lead to maintainability problems. For mobile applications most of the research has been done on Android applications with very little research on iOS applications. Our goal is to compare the variety, density, and distribution of code smells in iOS and Android applications. We analysed 273 open source iOS and 694 open source Android applications. We used PAPRIKA and GraphifySwift to find 19 object oriented code smells. We discovered that the distributions and proportions of code smells in iOS and Android applications differ. More specifically, we found: a) with the exception of one code smell (DistortedHierarchy) all code smells that could be observed in Android apps also occurred in iOS apps; b) the overall density of code smells is higher on iOS than on Android with LazyClass and DataClass particularly sticking out; c) with regards to frequency, code smells are more evenly distributed on iOS than on Android, and the distributions of code smell occurrences on class level are more different between the platforms than on app level.

Keywords

Mobile applications, Android, iOS, Code smells

1. Introduction

Code smells are patterns indicating bad practices that often lead to maintainability problems [1]. Code smells have been studied extensively for desktop applications (shortened to "apps" in the following). For mobile apps most of the analysis has been done on the Android platform.

Mannan et al. [2] analyzed 21 object oriented code smells in open source Android apps. They compared code smell occurrences on Android and Java desktop apps looking at differences in variety, density and distribution of code smells. They discovered that the variety of code smells is the same, but density and distribution of code smells in desktop Java and Android apps differ. They mention that other mobile platforms should have the same variety of code smells but do not discuss possible differences in density or distribution.

Habchi et al. [3] used the tool PAPRIKA [4] to analyse iOS and Android apps and compared proportions of code smells on these platforms. They analysed iOS apps for four object oriented, three iOS specific and Android apps for four object oriented and two Android specific code smells. They discovered that code smell proportions were higher in Android apps.

Our goal is to compare the variety, density and distribution of code smells in iOS and Android apps. First we will check if variety, density and distribution of code smells differ in iOS and Android apps to see if the results are similar to differences found between Android and desktop Java apps by Mannan et al. [2]. Second we extend the analysis done by Habchi et al. [3] by comparing the densities and distributions of more code smells in iOS and Android apps, to see if Android apps are in general more prone to code smells and if different platforms are more prone to different code smells. In this study we aim to answer the following research questions:

RQ 1: Are all types of object-oriented code smells present in both iOS and Android apps?

To answer this and the following research questions, we used the tool GraphifySwift[1] [5] to analyse iOS Apps and the tool PAPRIKA [4] to analyse Android apps. To make the code smell definitions (and calculations) comparable across plat-

[1]https://github.com/kristiinara/GraphifySwift

QuASoQ 2020: 8th International Workshop on Quantitative Approaches to Software Quality, December 1st, 2020, Singapore
kristiina.rahkema@ut.ee (K. Rahkema);
dietmar.pfahl@ut.ee (D. Pfahl)
0000-0003-2400-501X (D. Pfahl)

forms, we adapted the code smell queries defined by Rahkema et al. [5] when searching for code smells in Android apps. In total, we identified 19 code smell types that could potentially occur in apps on both platforms. We took under consideration that the variety of code smells depends on the programming language used. For example, the code smell RefusedParentBequest is not applicable to Swift because Swift lacks the *protected* keyword. Therefore, we did not include it in our analyses.

Our analysis showed that 18 of the 19 identified code smells occurred in apps on both platforms, i.e., Android and iOS. Code smell DistortedHierarchy never occurred in iOS apps.

To better understand whether the frequency of occurrence is similar, we formulated our second research question.

RQ 2: Do code smells occur with the same density in iOS and Android apps?

To answer this question, we calculated the overall density of all code smells and the densities of each of the 19 code smells over all apps on both iOS and Android. It turned out that, contrary to what Habchi et al. [3] expected, the overall density of code smells is higher in iOS apps than in Android apps. Code smells LazyClass, DivergentChange, PrimitiveObsession and DataClass had a particularly high density in iOS apps. On the other hand, code smells LongMethod, LongParameterList and ShotgunSurgery were clearly more frequent in Android apps. In addition, we found that the code smell densities per code smell type were sometimes higher and sometimes smaller in iOS apps as compared to Android apps. This might be explained by the fact that Android apps tend to have more of the code smells that correspond to more complex classes whereas iOS apps tend to have more of the code smells that correspond to more simple classes.

To better understand the distributions of code smells in apps on the two platforms iOS and Android, we formulated our third research question.

RQ 3: Do code smell distributions differ between iOS and Android apps?

To answer this question we first compared the proportions of code smell occurrences across all iOS and Android apps. The results confirmed what we had seen when we compared code smell densities: the proportions of code smells differ between platforms. In addition, we saw that code smells are more evenly distributed in iOS apps as compared to Android apps.

Then we analyzed how large the share of smelly apps on each platform is and how large the share of smelly classes is on each platform. We did these analyses for each code smell type separately. It turned out that the percentages of smelly apps are relatively similar between platforms. Only the code smell DataClass is much more prominent in iOS apps than in Android apps.

In addition, we found that the distributions of code smell occurrences on class level are more different between the platforms than on app level. This result might, again, be explained by the fact that Android apps usually have larger classes and, thus, tend to have more of the code smells that correspond to more complex classes whereas iOS apps tend to have more compact classes and, thus, tend to have more of the code smells that correspond to more simple classes. This effect is more prominent when doing the analysis on class level than on app level.

2. Related Work

Code smells in desktop applications: Fowler [1] defined 22 object oriented code smells and provided refactorings for these code smells. Khomh et al. [6] studied the impact of code smells. They found that code smells affect classes negatively and that classes with more code smells were more prone to changes [6]. Olbrich et al. [7] studied the evolution and impact of code smells based on two open source systems. Their findings confirmed that code smells affect the way how code changes in a negative way. They were also able to identify different phases of evolution in code smells [7]. Linares et al. [8] made a large scale analysis of Java Mobile apps and discovered that anti-patterns negatively impact software quality metrics such as fault-proneness [8].

Tufano et al. [9] studied the change history of 200 open source projects and found that most code smells are introduced when the corresponding code is created and not when it is changed. They also found that when code does become

smelly through evolution then it can be characterized by specific code metrics. Contrary to common belief [10] they discovered that most code smells are not introduced by newcomers, but by developers with high work loads and high release pressure [9].

Code smells in Android applications: Different kinds of code smells have been researched for Android, such as object-oriented, Android-specific, security-related and energy-related code smells. Gottschalk et al. proposed an approach to detect energy related code smells on mobile apps and validated this approach on Android and showed that it is possible to reduce energy consumption by refactoring the code [11]. Ghafari et al. [12] studied security-related code smells and discovered that most apps contain at least some security-related code smells.

Hecht [4] proposed an approach to detect code smells and anti-patterns on Android systems and implemented this approach in a tool called PAPRIKA. This tool analyses the Android APK, creates a model of the code and inserts this model into the neo4j database. Code smells are then defined as database queries which makes it possible to query code smells on a large number of apps at the same time. He analysed 15 popular apps for the occurrences of four object oriented and three Android code smells. Hecht et al. [13] tracked, the software quality of 106 popular Android apps downloaded from the Google Play Store along their evolution. They calculated software quality scores for different versions of these apps and tracked their evolution. There were different evolution graphs, such as constant decline, constant rise, stability or sudden change in either direction depending on the programming practices of the team [13]. This shows that code quality is not necessary linked to app size, but the programming practices of the developers. Mateus et al. [14] used PAPRIKA to analyze Android apps written in Java and kotlin. They compared code smell occurrences in both languages and concluded that apps that were initially written in Java and later introduced kotlin were of better quality than other Android apps [14]. They analysed a set of 2167 open source Android apps combining different databases of open source Android apps.

In these papers using PAPRIKA the number of code smells studied was limited due to the number of code smells PAPRIKA is able to detect and ranged from three to four object oriented code smells and four to six Android specific code smells [15][13][14]. Mannan et al. [2] decided to broaden this scope and studied 21 object oriented code smells using the commercial tool InFusion. They analyzed open source Android and Java desktop apps for these 21 code smells and compared their occurrences. Mannan et al. detected that the variety of code smells was the same and most code smells occur in both systems in a similar frequency with major differences only for a couple of code smells. They concluded that studying code smells on mobile platforms can be done with tools meant for desktop apps. They also found that the code smells that have been researched so far are not the same ones that occur most and that the focus should change to code smells that are more relevant [2]. They suggest that other mobile platforms will have the same code smells, but do not give any suggestions towards the possible differences in density or distribution. They analysed 500 Android and 750 Java desktop apps randomly selected from GitHub. Unfortunately, the tool Infusion used by Mannan et al. does not seem to be available anymore. Therefore a direct comparison using Infusion for code smell analysis on iOS is no longer possible.

Code smells in iOS applications: Habchi et al. [3] used PAPRIKA to detect code smells in iOS apps. They used ANTLR4 grammars to generate parsers for Swift and Objective-C code. They created the apps graphs that could then be used by PAPRIKA. They analysed 176 Swift and 103 Objective-C apps from a collaborative list of open source iOS apps. In their study they analysed four object oriented, three iOS specific and two Android specific code smells. They compared smell proportions in iOS and Android apps and discovered that the proportions of code smells were higher in Android apps. On the other hand proportions of code smells in Objective-C and Swift were similar [3].

Rahkema et al. [5] introduced a tool called GraphifySwift that analyses Swift code and detects 34 object oriented code smells. Similarly to PAPRIKA, GraphifySwift enters data about the analysed app into the neo4j database. The database structures used by PAPRIKA and GraphifySwift

are similar, but slightly different. In their analysis they used the same collaborative list of open source iOS apps but did not compare the results to Android.

In the following, we extend the research in [2, 3, 5]. We adapted the queries defined in [5], where possible, so that they could be applied to a database populated by PAPRIKA. We used PAPRIKA to analyse Android apps and GraphifySwift to analyse Swift apps. Then we compared the two platforms with regards to variety, density, and distribution of 19 code smells.

3. Methods

In Section 3.1, we present the tools used for code smell analysis. In Section 3.2, we cover the choice of apps and in Section 3.3 we describe the analysis performed.

3.1. Code Smell Analysis

In previous research a tool called PAPRIKA has been used to find code smells in Android applications [4, 15, 13, 3, 14]. PAPRIKA analyses the Android APK, enters data about the applications into a neo4j database and defines queries for each code smell. For analysing iOS applications Habchi et al. [3] used PAPRIKA to query code smells, but populated the neo4j database using ANTLR grammars. Rahkema et al. [5] introduced a new tool called GraphifySwift that extends the functionality of PAPRIKA. It analyses Swift code, enters data about the iOS applications into a neo4j database and defines database queries to find code smells. PAPRIKA is able to find four object oriented code smells. Since the queries for these four code smells are implemented identically in GraphifySwift, it produces the same results as PAPRIKA for them. In GraphifySwift additional code smell queries are defined. Overall, GraphySwift is able to find 34 object oriented code smells.

For the analysis of iOS apps we used the tool GraphifySwift. We used the same thresholds as in Rahkema et al. [5]. Note that we focused on Swift code as Swift has replaced Objective-C and not many differences between the two languages are to be expected according to Habtchi et al. [3].

For Android apps we used PAPRIKA to populate the neo4j database. We then took the queries defined by Rahkema et al. for GraphifySwift to find code smells. Since GraphifySwift was originally developed to analyse iOS apps we had to adapt the code smell queries so that they could be used on the database produced by PAPRIKA. We made the following changes to the code smell queries:

We removed references to Module nodes, i.e., the relationship

```
(app)-APP_OWNS_MODULE->(module)-
        MODULE_OWNS_CLASS->(class)
```

was substituted by the relationship

```
(app)-APP_OWNS_CLASS->(class)
```

We removed references to argument type or substituted them with argument name. Argument names are not accessible in Java bytecode and therefore the argument name provided by PAPRIKA is actually the argument type.

Finally, we added the relationship

```
(variable|argument)-IS_OF_TYPE
                    ->(class)
```

by finding classes whose name matched the argument name or variable type.

After these modifications of the database and queries, 19 of the 34 GraphifySwift code smell queries could be used on the Android app database produced by PAPRIKA.

The code smell queries that had to be excluded contained metrics or attributes that were not provided by PAPRIKA. We excluded for example queries referring to code duplication, maximum nesting depth, number of switch statements and number of comments.

For the analysis of Android apps we calculated new thresholds based on the apps that we analysed. The list of iOS and Android thresholds is included in the thresholds table[2].

3.2. Choice of Applications

For analysis of iOS apps we used the same collaborative list of open source iOS apps as was used by

[2] https://figshare.com/articles/conference_contribution/Thresholds_for_iOS_and_Android_code_smell_analysis/13102991

Rahkema et al. [5] and whose older version was used by Habchi et al. [3]. The final set of successfully analysed apps was the same as in [5] and included 273 open source iOS apps.

For analysis of Android apps we took the list of apps provided by Habchi et al. [3]. Since the list only included app package names, we queried AllFreeAPK api[3] to find and download these apps. We decided to search AllFreeAPK instead of GitHub, as PAPRIKA uses APKs for analysis and this way we were able to skip the step of compiling these apps. Later during the analysis we needed to discard some of the very big apps due to performance issues. In total we included 694 open source Android apps in our analysis.

3.3. Data Analysis

To answer RQ1, we checked whether any of the 19 identified code smells occurred in at least one app on each platform.

To answer RQ2, we calculated the densities of code smells for both iOS and Android apps and compared these. Code smell density was calculated by counting the number of code smells (total and per code smell type) and dividing by the number app instructions.

To answer RQ3, we had to perform several calculations. To calculate the relative frequencies of code smells per code smell type on each platform, we counted the code smells of a type in all apps and divided by the total code smell count. We did this per platform. To calculate the code smell distributions on app and class levels per platform, we counted how many apps (and classes) contain at least one code smell of a certain type and then divided by the total number of apps (and classes).

4. Results

We analysed 273 open source iOS apps using GraphifySwift and 694 open source Android apps using PAPRIKA and modified code smell queries from GraphifySwift to answer our research questions. We analyzed the apps with regards to 19 code smells: BlobClass, ComplexClass, CyclicClassDependency, DataClass, DataClumpFields, DistortedHierarchy, DivergentChange, InappropriateIntimacy, LazyClass, LongMethod, LongParameterList, MiddleMan, ParallelInheritanceHierarchies, PrimitiveObsession, SAPBreaker, ShotgunSurgery, SpeculativeGeneralityProtocol, SwissArmyKnife and TraditionBreaker.

Below, we present and discuss the results for each research question separately.

RQ 1: Are all types of object-oriented code smells present in both iOS and Android apps?

When comparing the occurrence of code smells on each platform, we found that 18 of the 19 identified code smells occurred in apps on both platforms, i.e., Android and iOS. Code smell DistortedHierarchy never occurred in iOS apps.

Our result does not fully support Mannan et al.'s expectation that mobile apps on other platforms than Android should exhibit the same code smells [2].

RQ 2: Do code smells occur with the same density in iOS and Android apps?

The results of our code smell density analysis is shown in Figure 1. Accumulated over all code smells it turned out that the apps on the iOS platform had a density of 41.7 smells/kilo-instructions while the apps on Android only had a density of 34.4 smells/kilo-instructions. This result is contrary to what Habchi et al. [3] expected.

Moreover, it can be seen from Figure 1 that the code smell densities differ between iOS and Android. Code smells LazyClass, DivergentChange, PrimitiveObsession and DataClass had a particularly high density in iOS apps. On the other hand, code smells LongMethod, LongParameterList and ShotgunSurgery were clearly more frequent in Android apps. The fact that code smell densities were sometimes higher and sometimes lower in iOS apps as compared to Android apps might be explained by the fact that Android apps tend to have more of the code smells that correspond to more complex classes whereas iOS apps tend to have more of the code smells that correspond to more simple classes.

RQ 3: Do code smell distributions differ between iOS and Android apps?

Figure 2 shows the relative frequency of code smell occurrences over all apps on the Android platform (blue bars) and the iOS platform (red bars). The results confirm what we had seen when

[3] https://m.allfreeapk.com/api/

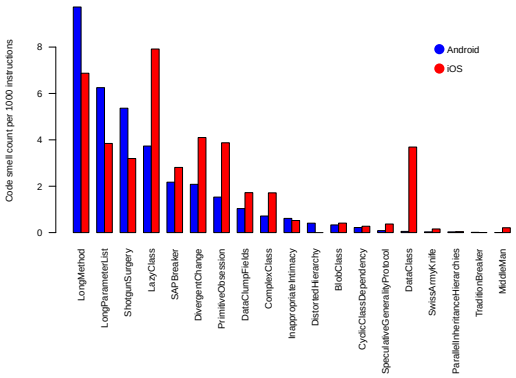

Figure 1: Comparison of code smell densities between Android (blue) and iOS (red) apps

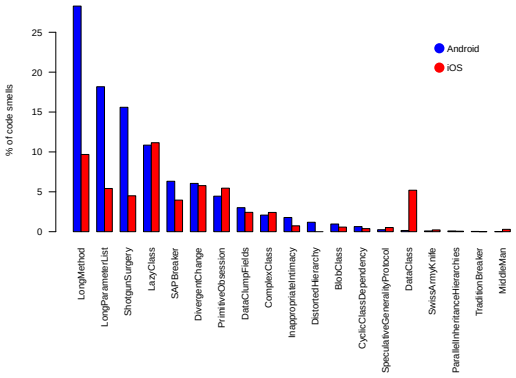

Figure 2: Code smell proportions on Android (blue) and iOS (red)

we compared code smell densities: the proportions of code smells differ between platforms. In addition, we see that code smells are more evenly distributed in iOS apps as compared to Android apps.

Then we analyzed how large the share of smelly apps on each platform is and how large the share of smelly classes is on each platform. We did these analyses for each code smell type separately.

Figures 3 and 4 show the percentages of apps and classes, respectively, containing code smells of a certain type.

We found that the percentages of smelly apps are relatively similar between platforms. The biggest differences occurs for code smell DataClass (79% of iOS apps have at least one affected class

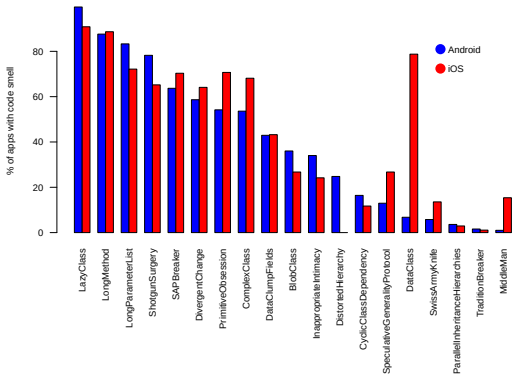

Figure 3: Comparison of code smell frequencies on app level between Android (blue) and iOS (red)

while only 7% of Android apps are affected), MiddleMan (15% of iOS apps are affected but only 1% of Android apps), and DistortedHierarchy (25 % of Android apps are affected but none of the iOS apps is).

We found that the distributions of code smell occurrences on class level are more different between the platforms than on app level. This result might, again, be explained by the fact that Android apps usually have larger classes and, thus, tend to have more of the code smells that correspond to more complex classes whereas iOS apps tend to have more compact classes and, thus, tend to have more of the code smells that correspond to more simple classes. This effect is more prominent when doing the analysis on class level than on app level.

In addition, we analyzed the occurrence of the method-based code smells LongMethod and LongParameterList separately. We found that in iOS apps 9% of methods are considered LongMethod while this is the case for 14% of the methods in Android apps. In iOS apps 5% of the methods have a LongParameterList while this is the case for 9% of methods in Android apps.

5. Threats to Validity

Internal Validity: In our case internal validity might be affected by how code smells are detected by the tools used. PAPRIKA has been used in multiple studies [15, 13, 3, 14]. GraphifySwift was in-

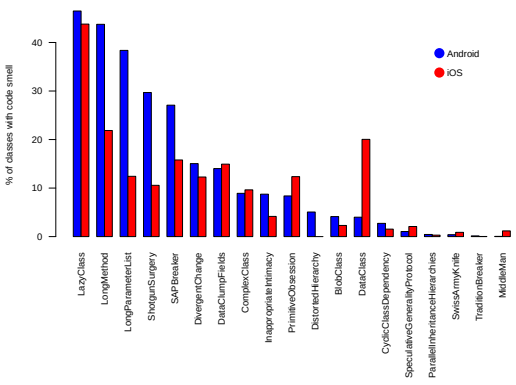

Figure 4: Comparison of code smell frequencies on class level between Android (blue) and iOS (red)

troduced in [5] and validated by replicating results in [3]. We adapted code smell queries defined in [5], but did so by not changing the code smell definitions themselves.

External Validity: We analysed open source apps. For swift the analysis can only be performed if the code of the app is accessible. For Android the analysis could be performed on apps from the app store. Therefore for both platforms open source apps were chosen. Previously [5] it was shown that although there are some differences between apps that are on the app store the differences are small. On the iOS platform we only analyzed apps written in Swift. Given that Objective-C and Swift code is quite similar, we assume our results extend to apps written in Objective-C.

Construct Validity: GraphifySwift uses standard definitions of code smells found in literature [5]. In code smell queries we use thresholds calculated based on the app set analysed. Using thresholds is a common approach for detecting code smells. We used the same method to determine thresholds as was used by Hecht et al. [13], Habchi et al. [3] and [5]. Thresholds might differ between languages, but since they are calculated based on the current set of apps analysed language specific differences should be resolved.

Reliability: For iOS analysis we used the same collaborative list of open source iOS apps written in Swift as was used in [5]. All these apps are available on GitHub. The list of successfully analysed apps can be found on the tool GitHub page. GraphifySwift is open source and also available on the tool GitHub page. For Android analysis we used the list of apps analysed by [3], the list of successfully analysed apps can be found in the list of apps[4]. PAPRIKA is open source and also available on GitHub. The adapted code smell queries used for Android analysis can be found in the list of Android code smell queries[5].

6. Conclusion

Mannan et al. [2] analysed the density and distribution of code smells in Android apps. We calculated a similar density and distribution for iOS and Android apps and saw that these densities and distributions were different. Additionally we discovered that one of the code smells analysed by Mannan et al. was not present in iOS apps.

Habchi et al. [3] compared ratios of code smell occurrences on iOS and Android. We extended their research by adding additional code smells to the analysis and found that code smell occurrences are not always higher in Android apps. For some code smells they were higher in iOS apps. This shows that Android apps are not necessarily smellier, but different kinds of code smells are more prevalent depending on the platform.

These results can be interesting for developers moving from one platform to the other. It can also be useful for developers of tools for these platforms. We see that the emphasis on which code smells to look at is different depending on the platform.

Acknowledgments

This research was partly funded by the Estonian Center of Excellence in ICT research (EXCITE), the IT Academy Programme for ICT Research Development, the Austrian ministries BMVIT and BMDW, and the Province of Upper Austria under the COMET (Competence Centers for Excellent Technologies) Programme managed by FFG,

[4] https://figshare.com/articles/dataset/iOS_and_Android_app_analysis_data/13103012

[5] https://figshare.com/articles/conference_contribution/GraphifySwift_queries_adapted_for_PAPARIKA_for_Android_code_smell_analysis/13102994

and by the group grant PRG887 of the Estonian Research Council. We thank Rudolf Ramler for the thorough review of a previous version of this paper.

References

[1] M. Fowler, Refactoring: improving the design of existing code, Addison-Wesley Professional, 2018.

[2] U. A. Mannan, I. Ahmed, R. A. M. Almurshed, D. Dig, C. Jensen, Understanding code smells in android applications, in: 2016 IEEE/ACM Int'l Conf. on Mobile Softw. Eng. and Systems (MOBILESoft), IEEE, 2016, pp. 225–236.

[3] S. Habchi, G. Hecht, R. Rouvoy, N. Moha, Code smells in ios apps: How do they compare to android?, in: 2017 IEEE/ACM 4th Int'l Conf. on Mobile Softw. Eng. and Systems (MOBILESoft), IEEE, 2017, pp. 110–121.

[4] G. Hecht, An approach to detect android antipatterns, in: Proc. of the 37th Int'l Conf. on Software Engineering-Volume 2, IEEE Press, 2015, pp. 766–768.

[5] K. Rahkema, D. Pfahl, Empirical study on code smells in ios applications, in: Proc. of the IEEE/ACM 7th Int'l Conf. on Mobile Softw. Eng. and Systems, MOBILESoft '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 61–65.

[6] F. Khomh, M. Di Penta, Y.-G. Gueheneuc, An exploratory study of the impact of code smells on software change-proneness, in: 2009 16th Working Conf. on Reverse Engineering, IEEE, 2009, pp. 75–84.

[7] S. Olbrich, D. S. Cruzes, V. Basili, N. Zazworka, The evolution and impact of code smells: A case study of two open source systems, in: 2009 3rd Int'l symposium on empirical software engineering and measurement, IEEE, 2009, pp. 390–400.

[8] M. Linares-Vásquez, S. Klock, C. McMillan, A. Sabané, D. Poshyvanyk, Y.-G. Guéhéneuc, Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps, in: Proc. of the 22nd Int'l Conf. on Program Comprehension, ACM, 2014, pp. 232–243.

[9] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, D. Poshyvanyk, When and why your code starts to smell bad, in: Proc. of the 37th Int'l Conf. on Software Engineering-Volume 1, IEEE Press, 2015, pp. 403–414.

[10] T. Sharma, Extending Maintainability Analysis Beyond Code Smells, Ph.D. thesis, 2019.

[11] M. Gottschalk, J. Jelschen, A. Winter, Saving energy on mobile devices by refactoring., in: EnviroInfo, 2014, pp. 437–444.

[12] M. Ghafari, P. Gadient, O. Nierstrasz, Security smells in android, in: 2017 IEEE 17Th Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM), IEEE, 2017, pp. 121–130.

[13] G. Hecht, O. Benomar, R. Rouvoy, N. Moha, L. Duchien, Tracking the software quality of android applications along their evolution (t), in: 2015 30th IEEE/ACM Int'l Conf. on Automated Softw. Eng. (ASE), IEEE, 2015, pp. 236–247.

[14] B. G. Mateus, M. Martinez, An empirical study on quality of android applications written in kotlin language, Empirical Software Engineering (2018) 1–38.

[15] G. Hecht, R. Rouvoy, N. Moha, L. Duchien, Detecting antipatterns in android apps, in: Proc. of the Second ACM Int'l Conf. on Mobile Softw. Eng. and Systems, IEEE Press, 2015, pp. 148–149.