Postprint: An Empirical Study on Code Smell Co-occurrence in Android Applications

Authors: Bian Yixin, Wang Luying, Zhao Song, Zhu Xiao

Date: 2022-05-10T00:00:00+00:00

Abstract

The coexistence of multiple code smells is more detrimental than single-type code smells. Existing empirical studies have primarily focused on analyzing the coexistence of code smells in desktop applications, with a notable lack of research on code smell coexistence in Android applications. To investigate the coexistence of code smells in Android applications and compare it with that in desktop applications, we detected 285 Android applications and 30 desktop applications, respectively, and analyzed the 10 types of smells identified. The analysis methodology is as follows: First, we calculate the percentage of classes affected by multiple smells based on the detection results. Then, we compute the frequency of code smell coexistence using formulas. Finally, we employ Spearman correlation coefficient to analyze the relationship between code smell coexistence and application size. The conclusions are as follows: a) In Android applications, classes affected by more than one code smell account for 31.04% of the total number of classes with smells; b) In applications across both platforms, the two pairs of code smells Brain Class–Brain Method and God Class–Brain Method exhibit relatively high frequencies of coexistence; c) Single smell, two-smell coexistence, and three-smell coexistence are strongly correlated with the size of Android applications.

Full Text

Abstract

Code smell co-occurrences are more harmful to systems than individual smell types. Although multiple research works have focused on code smell co-occurrences in traditional desktop applications, researchers pay less attention to code smell co-occurrences in Android applications. To study the phenomenon of object-oriented code smell co-occurrences in Android applications and compare it with the phenomenon in desktop applications, we analyzed 10 object-oriented code smell types detected in 285 Android apps and 30 desktop apps.

First, we calculated the percentage of classes affected by various kinds of smells according to the detection results. Second, we analyzed the frequency of code smell co-occurrences using a formula. Finally, we used the Spearman correlation coefficient to analyze the relationship between code smell co-occurrences and application size. The results show that: (a) in Android applications, 31.04% of smelly classes are affected by more than one type of code smell; (b) in applications on both platforms, two pairs of code smells—Brain Class—Brain Method and God Class—Brain Method—frequently co-occur; and (c) single smells, co-occurrences of two smell types, and co-occurrences of three smell types are strongly correlated with the size of Android applications.

Key words: code smell co-occurrences; Android applications; desktop applications; empirical study

0 Introduction

Code smells, also known as code bad smells, are code segments with design defects in software systems [1]. Code smell co-occurrences refer to the presence of more than one code smell in a single class. Research indicates that code smell co-occurrences pose a greater threat to systems than single-type code smells [2]. Palomba et al. [3] conducted an empirical study on code smell co-occurrences in traditional desktop applications, revealing that such co-occurrences are widespread and that affected classes contain at most three different types of code smells.

In recent years, with the rapid development of mobile communication technology, mobile applications have become the dominant force in the software industry. Since 2016, global mobile application downloads have continued to grow, exceeding 200 billion in 2019 and reaching 230 billion in 2021—an increase of 12 billion compared to 2020. Android applications account for over 80% of the mobile app market, while iOS applications represent less than 20%. Research shows that code smells can affect any software system [4, 5]. However, existing empirical studies have primarily focused on analyzing code smell co-occurrences in desktop applications, with little research specifically targeting Android applications. Consequently, it remains unclear whether code smell co-occurrences exist in Android applications, and if so, to what extent they impact systems. Furthermore, due to numerous differences between traditional desktop and Android applications—including program structure, API calls, memory, CPU, network, and battery usage—the differences in code smell co-occurrence patterns between these platforms are also unknown.

To address these questions, this paper employs an empirical research method to analyze code smell co-occurrences in Android applications and compare them with those in traditional desktop applications. Our findings are as follows: (a) Code smell co-occurrences exist in Android applications, with 31.04% of classes affected by code smells experiencing interference from more than one smell type. Among these classes with co-occurring smells, 0.05% suffer from six simultane-

ous code smells. Thus, while the proportion of classes affected by multiple co-occurring smells is not large, the harm to systems is severe. In contrast, code smell co-occurrences are more prevalent in desktop applications, posing an even greater threat. (b) In Android applications, two pairs of code smells—Brain Class—Brain Method and God Class—Brain Method—co-occur more frequently than others. In desktop applications, three pairs show higher co-occurrence frequencies: Brain Class—Brain Method, God Class—Brain Method, and Refused Parent Bequest—Brain Method. This indicates that Brain Class—Brain Method and God Class—Brain Method tend to co-occur in applications on both platforms, though the co-occurring smells are not in one-to-one correspondence across platforms. (c) Code smell co-occurrences correlate with Android application size; as application size increases, so do the numbers of classes containing one, two, or three smells, though classes with four, five, or six co-occurring smells show no such correlation.

These conclusions help Android application researchers and tool developers deepen their understanding of code smells, enabling more effective handling of the negative impacts of code smell co-occurrences, reducing maintenance costs, and improving Android application quality. For researchers, understanding smell interdependencies can inform detection and refactoring studies, necessitating consideration of not only individual smell detection methods but also dependencies between smells when developing co-occurrence detection approaches. Additionally, when refactoring Android applications, researchers must consider these dependencies to optimize refactoring order, saving time and improving accuracy and efficiency. For tool developers, these dependencies can guide refactoring efforts by prioritizing frequently co-occurring smells, thereby reducing development time costs.

1.1 Code Smells in Android Applications

Fowler et al. [1] first proposed 22 code smells with corresponding refactoring methods. As mobile devices have proliferated, many scholars have shifted focus from traditional desktop to Android applications. Mannan et al. [6] compared code smells between Android and desktop applications through empirical methods, while Rahkema et al. [7] found that, except for Distorted Hierarchy, code smells present in Android applications also occur in iOS applications.

Research shows that Android applications exhibit some Android-specific code smells distinct from traditional desktop applications. Reimann et al. [8] proposed 30 Android-specific code smells that primarily affect non-functional attributes. Subsequent research has explored these smells from various perspectives: Habchi et al. [9] analyzed eight Android-specific smells to study their causes, evolution patterns, and removal methods, while Rasool et al. [10] developed DAAP, a detection tool for 25 Android-specific smells. This study focuses on object-oriented code smell co-occurrences in Android applications, excluding Android-specific smells.

1.2 Code Smell Co-occurrences

Code smell co-occurrences refer to the presence of two or more code smells in the same class or method [11]. Pietrzak et al. [12] first identified this phenomenon, defining six co-occurrence relationships and examining dependencies between co-occurring smells. Yamashita [13] and Sjøberg [14] argued that such co-occurrences reduce system maintainability, requiring refactoring. Palomba et al. [2, 3] conducted a large-scale empirical study of 395 versions of 30 desktop applications, finding that code smell co-occurrences are widespread and that classes affected by multiple smells are more prone to changes and defects than those with single smells. Martins et al. [11] analyzed 11 versions of three closed-source Java systems, showing that refactoring to eliminate co-occurrences reduces system complexity. These studies, however, focused exclusively on desktop applications.

Hamdi et al. [15] empirically studied code smell co-occurrences in Android applications, analyzing 15 Android-specific and 10 object-oriented smell types. They found co-occurrences to be common, identifying 14 frequently co-occurring pairs. However, their study has limitations: they did not manually review detection tool outputs, making results dependent on potentially imperfect tool accuracy (which typically suffers from false positives and negatives [15]); they did not examine correlations between co-occurrences and application size; they did not compare co-occurrence patterns across platforms despite significant differences between Android and desktop applications (a comparison Mannan et al. [6] demonstrated as meaningful); and they did not clarify whether co-occurring smells have simple one-to-one relationships or more complex dependencies—a critical factor for refactoring research, as detection aims ultimately at removal and quality improvement.

Addressing these gaps, this paper empirically analyzes code smell co-occurrences in Android applications and compares them with desktop applications. Unlike single smells, co-occurrences reflect interdependencies that deepen understanding of smell origins, enabling developers to avoid introducing smells during development and informing detection and refactoring methods that minimize system harm during maintenance.

2.1 Research Questions

By analyzing Android application source code, this study investigates code smell co-occurrences, proposing three research questions:

RQ1: Do code smell co-occurrences exist in Android applications? If so, to what extent, and how does this differ from desktop applications?

RQ2: Which code smells tend to co-occur in Android applications, and how does this differ from desktop applications?

RQ3: Is there a correlation between code smell co-occurrences and Android application size?

2.2.1 Application Selection

We selected 285 Android applications and 30 desktop applications for our empirical study. The Android selection process involved three steps: (a) We first identified 539 open-source Android applications from AndroZooOpen [16], a dataset containing 76,466 Android apps. AndroZooOpen includes apps developed in various languages; we selected Java-based programs because Java is a mainstream mobile development language with numerous accessible detection tools. (b) We then used the open-source tool RepoReapers [17] to filter out low-quality projects. RepoReapers scores program quality across eight dimensions (architecture, community, continuous integration, documentation, commit history, license, issues, and unit testing). We retained programs scoring above zero in at least seven dimensions, yielding 321 candidate applications. (c) Finally, to ensure diversity, we categorized candidates according to Google Play Store classifications [18, 19] into six categories: Development, Audio/Video, Social/Communication, Home/Education, Security/Utilities, and Others, with proportions shown in [Figure 1: see original paper].

[Figure 1: see original paper]

After this filtering process, we collected 285 Android applications comprising 34,617 classes, 315,634 methods, and 2,701,826 lines of code. For comparison, we selected the same 30 open-source desktop applications used by Palomba et al. [3] in their study of traditional desktop applications, totaling 43,257 classes, 436,299 methods, and 4,948,368 lines of code.

2.2.2 Code Smell Detection Tools

We used iPlasma for code smell detection. Mainstream tools include iPlasma, inFusion, Checkstyle, JDeodorant, PMD, DECOR, and Stench Blossom [20]. Fontana et al. [20] found iPlasma and inFusion detect the most smell types. Mannan et al. [6] showed that desktop application detection tools work for Android apps. Although inFusion claims to detect Fowler' s 22 smells, Rahkema et al. [7] noted it is no longer available. Reis et al. [21] reported that 5.8% of studies use inFusion while 15.4% use iPlasma. Therefore, we selected iPlasma.

2.2.3 Code Smell Selection

We analyzed 10 code smells (Table 1) commonly used in related research [6] and detectable by iPlasma.

The experimental process is illustrated in [Figure 2: see original paper].

[Figure 2: see original paper]

After selecting experimental subjects, we executed the following steps: (a) **Code smell detection:** We used iPlasma to detect smells in the 285 Android and 30 desktop applications selected in Section 2.2.1. (b) **Manual review:** Despite iPlasma' s high accuracy, its detection rules have limitations that produce

false positives. Therefore, we manually reviewed tool outputs. Four volunteers (one master' s student and three undergraduates in computer science) were recruited and divided into two pairs: one pair reviewed Android results, the other desktop results. The review process involved: (i) training volunteers to deeply understand the 10 code smells; (ii) having each volunteer independently examine all source files against smell definitions and record results; (iii) using Cohen' s kappa [22] to measure inter-reviewer agreement (= 0.63); and (iv) reconciling disagreements through re-examination and recording final data. (c) **Recording results:** We recorded smell information (e.g., affected class/method names) and separately documented co-occurring classes/methods for further analysis.

2.3 Experimental Design

Experiment for RQ1: We first calculated the number of code smells per class from detection results, then computed percentages of classes affected by one versus multiple smells.

Experiment for RQ2: We analyzed co-occurrence frequency using Palomba et al.' s formula [3]:

$$cooccurrences_{i,j} = \frac{|CS_i \wedge CS_j|}{|CS_i|}, \quad i \neq j$$

where CS_i and CS_j represent different code smell types, the numerator counts total co-occurrences of smells i and j, and the denominator counts total occurrences of smell i. Swapping i and j yields different results.

Experiment for RQ3: We used Spearman correlation coefficients to analyze relationships between co-occurrences and Android application size (measured by class count, method count, and lines of code). We applied Cohen' s guidelines [20] to interpret correlation strength.

3 Results and Analysis

RQ1: Table 2 shows co-occurrence patterns in both platforms. In Android applications, 31.04% of smelly classes suffer from multiple smell types. Among these, 21.73% contain two smells (the largest proportion), while 6.64%, 2.21%, 0.37%, and 0.09% contain three, four, five, and six smells respectively. Android applications show 1.34% higher co-occurrence rates than desktop applications (31.04% vs. 29.70%). While Palomba et al. [14] found desktop applications affected by at most three co-occurring smells across 395 versions of 30 applications, we observed four, five, and six smell co-occurrences in the same 30 desktop applications.

RQ2: Table 3 shows co-occurrence frequencies in Android applications. Two pairs co-occur most frequently: Brain Class–Brain Method and God Class–Brain Method. Specifically, 61% of classes with Brain Class also contain Brain

Method, and 17% of classes with God Class also contain Brain Method. Co-occurring smells are not in one-to-one correspondence; for example, one class in the Android application Toposuite contains eight Feature Envy instances and eight Intensive Coupling instances.

Table 4 shows co-occurrence frequencies in desktop applications. Brain Method tends to co-occur with other smells, with three pairs showing higher frequencies: Brain Class–Brain Method, God Class–Brain Method, and Refused Parent Bequest–Brain Method. As in Android applications, these co-occurring smells lack one-to-one correspondence.

Combined, Tables 3 and 4 show that Brain Class–Brain Method and God Class –Brain Method co-occur frequently in both platforms.

RQ3: As Table 5 shows, five- and six-smell co-occurrences affect less than 1% of Android applications and are excluded from correlation analysis. Single smells, two-smell co-occurrences, and three-smell co-occurrences show strong correlations with Android project size (i.e., larger projects have more classes with one, two, or three smells), while four-smell co-occurrences show moderate correlation.

4 Conclusion

This paper presents an in-depth empirical analysis of code smell co-occurrences in Android applications, differing from prior work by focusing specifically on this platform. We found that code smell co-occurrences exist in Android applications, with 31.04% of smelly classes affected by multiple smells. Comparing Android and desktop applications revealed that Brain Class–Brain Method and God Class–Brain Method co-occur frequently on both platforms. Additionally, co-occurrences correlate with Android application size.

These findings help Android developers and maintainers better understand code smells to more effectively mitigate negative impacts, reduce costs, and improve quality. For researchers, smell dependencies inform detection and refactoring studies, necessitating consideration of both individual and interdependent smells. For tool developers, these dependencies can guide refactoring prioritization of frequently co-occurring smells.

Future work should expand the empirical scope to include more representative open-source and commercial Android applications, analyze additional smell types, and investigate co-occurrences in projects developed in other languages (e.g., Kotlin).

References

[1] Martin Fowler. Refactoring: improving the design of existing code [M]. translated by Xiong jie, Lin congyu. 2nd ed. Beijing: Posts&Telecom Press, 2019.

[2] Palomba F, Bavota G, Penta D M, et al. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation [J]. Empirical Software Engineering, 2018, 23 (3): 1188-1221.

[3] Palomba F, Bavota G, Penta D M, et al. A large-scale empirical study on software maintainability: An empirical study [C]// Proc of the 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 440-451.

[4] Macia I, Garcia J, Popescu D, et al. Are automatically-detected code anomalies relevant to architectural modularity? An exploratory analysis of evolving systems [C]// Proc of the 11th annual international conference on Aspect-oriented Software Development. 2012: 167-178.

[5] Oizumi W, Garcia A, Sousa L S, et al. Code anomalies flock together: Exploring code anomaly agglomerations for locating design problems [C]// Proc of IEEE/ACM 38th International Conference on Software Engineering (ICSE). IEEE, 2016: 440-451.

[6] Mannan U A, Ahmed I, Almurshed R A M, et al. Understanding code smells in android applications [C]// Proc of IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft). IEEE, 2016: 225-236.

[7] Rahkema K, Pfahl D. Comparison of Code Smells in iOS and Android Applications [C]// Proc of QuASoQ@ APSEC. 2020: 79-86.

[8] Jan Reimann, Martin Brylski, Uwe Assmann. A tool-supported quality smell catalogue for android developers [C]// Proc of the conference Modellierung in the Workshop Modellbasierte und modellgetriebene Softwaremodernisierung-MMSM. 2014, 2014.

[9] Habchi S, Moha N, Rouvoy R. Android code smells: From introduction to refactoring [J]. Journal of Systems and Software, 2021, 177: 110964.

[10] Rasool G, Ali A. Recovering Android Bad Smells from Android Applications [J]. Arabian Journal for Science and Engineering, 2020, 45 (4): 3289-3315.

[11] Martins J S, Bezerra C I M, Uchôa A, et al. Are code smell co-occurrences harmful to internal quality attributes? a mixed-method study [C]// Proc of the 34th Brazilian Symposium on Software Engineering. 2020: 52-61.

[12] Pietrzak B, Walter B. Leveraging code smell detection with inter-smell relations [C]// Proc of International Conference on Extreme Programming and Agile Processes in Software Engineering. Springer, Berlin, Heidelberg, 2006: 75-84.

[13] Yamashita A, Moonen L. Exploring the impact of inter-smell relations on software maintainability: An empirical study [C]// Proc of the 35th International Conference on Software Engineering (ICSE). IEEE, 2013: 440-451.

[14] Sjøberg D I K, Yamashita A, Anda B C D, et al. Quantifying the effect of code smells on maintenance effort [J]. IEEE Trans on Software Engineering,

2012, 39 (8): 1144-1156.

[15] Hamid O, Ouni A, AlOmar E A, et al. An empirical study on code smells co-occurrences in android applications [C]// Proc of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2021: 26-33.

[16] Liu Pei, Li Li, Zhao Yanjie, et al. Androzooopen: Collecting large-scale open source android apps for the research community [C]// Proc of the 17th International Conference on Mining Software Repositories. 2020: 1-5.

[17] Munaiah N, Kroh S, Cabrey C, et al. Curating github for engineered software projects [J]. Empirical Software Engineering, 2017, 22 (6): 3219-3253.

[18] Ahmed I, Mannan U A, Gopinath R, et al. An empirical study of design degradation: How software projects get worse over time [C]// Proc of ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2015: 1-10.

[19] Souza L B L, Maia M A. Do software categories impact coupling metrics? [C]// Proc of the 10th Working Conference on Mining Software Repositories (MSR). IEEE, 2013: 217-220.

[20] Fontana F A, Braione P, Zanoni M. Automatic detection of bad smells in code: An experimental assessment [J]. Journal of Object Technology, 2012, 11 (2): 5: 1-38.

[21] Reis J P, Abreu F B, Carneiro G F, et al. Code Smells Detection and Visualization: A Systematic Literature Review [J]. Archives of Computational Methods in Engineering, 2021: 1-48.

[22] Cohen, J. A Coefficient of Agreement for Nominal Scales [J]. Educational & Psychological Measurement, 1960, 20 (1): 37-46.

Note: Figure translations are in progress. See original paper for figures.