ISSN 2278-3091
Volume 9 No.2, March - April 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse194922020.pdf
https://doi.org/10.30534/ijatcse/2020/194922020

A Static Analysis of Android Source Code for Design Patterns Usage

Diaeddin Rimawi[1], Samer Zein[2]
[1]Master of Software Engineering, Birzeit University, Palestine, dmrimawi@gmail.com
[2]Master of Software Engineering, Birzeit University, Palestine, szain@birzeit.edu

ABSTRACT

Design patterns help software developers in building better software designs as well as fostering software maintainability and re-usability. Recently, mobile applications, apps hereinafter, have gained much ground in critical domains, such as banking, health, payments, and military, just to mention a few. Accordingly, it has become imperative to consider increasing the apps' code quality, which urges to better usage of design patterns. Although there is plethora of studies that discuss design patterns usage in object-oriented languages such as Java, C++, and C#, to our best of knowledge, no studies have discussed design patterns usage for Android apps. This study performs an exploratory research using static code analysis methods and a sample of more than 1400 Android apps toward finding design patterns implemented inside their source code. We extend our PatRoid framework, which detects all design patterns in Android app source code. Our initial results show that there is a variation in the extent in which design patterns are applied among different Android apps' categories. Overall, we argue that there is still a lack of proper usage of design patterns in Android apps development.

Key words: Android Apps, Design patterns, PatRoid, Static Analysis, Code Analysis.

1. INTRODUCTION

In a recent statistical comparison between different mobile platforms, it is apparent that Android platform leads the market with a significant difference, and is expected to maintain this position for the upcoming years [2]. Statistics in the beginning of 2020 show that Android OS holds 86.6% of total world OS shipment market, while iOS from Apple comes in second place holding the remaining 13.4%.

Android platform is not limit to mobile phone users only, in fact, manufacturing industry and software development community adopted Android in other contexts such as smart TVs, tablets, wearables, automobiles, etc.[1], [3], [4]. Nowadays, Google Play[5]the official Android apps store, holds over than 2.1 million apps divided over 60 different categories.

Our review to the literature, shows that there is a high diversity of applications for Android apps. They have become part of complex and critical categories, such as Medical, Health Monitoring, Banking, Education, Traveling, etc. Furthermore, each of these categories are discussed by several studies[6]–[11].

Android app development has special characteristics than traditional desktop, and web development. Since desktop and web apps development are considered mature, Android development is still considered a new field, with a large portion of developers are known to be novice [1], [12], [13]. That been said, firm guidance is required, and software design patterns can achieve better software quality, re-usability, maintainability, and evolution[14]–[19].

Recent and old studies have investigated design patterns importance, as well as the different approaches to detect them and how to improve these approaches with different Object Oriented (O-O) desktop languages, such as Java, C++, and C# [20]–[27].

On the other hand, the current state-of-the-art shows an apparent gap in the area of mobile apps. In fact, little studies were published to address design patterns with Android apps [19], [28], [29], however, they only investigate UI (User Interface) design patterns, and not O-O design patterns, which are the focus of this study.

To the best of our knowledge the only study that addresses design patterns detection for Android apps is our previous study [1]. In our previous study, we implemented a new open source automated framework for design patterns detection in Android apps (PatRoid[1]).

PatRoid is a model based on graph isomorphism approach. Where it divides each design patterns into smaller easy to catch sub-patterns. PatRoid is capable to detect all 23 Gang of Four (GoF) design patterns.

This study aims to explore what design patterns do Android apps developers apply. It extends our previous study to study a sample of more than 1400 Android apps' source code collected from F-Droid[2].

Our preliminary results show that Android app developers are applying O-O design patterns in varying extents depending on apps categories. Additionally, it shows that the usage of

[1]PatRoid: is an open source framework for Android O-O design patterns detection, it is implemented using Python language and can be found at GitHub on the following link "https://github.com/dmrimawi/PatRoid".

[2]F-Droid, a free and open source Android apps repository. It can be found at F-Droid website "https://f-droid.org/en/".

design patterns in general in Android apps is insufficient.
On the other hand, our study shows that there is high diversity over different Android apps categories which means that there are some categories that are more mature than others, such as the Tools and Lifestyle categories.
The rest of this paper is structured to first discusses the literature review in section 2, then illustrates the methodology in section 3. Section 4 shows the results of this study, and section 5 discusses these results. Threats to validity and future work are shown in section 6 and section 7, and finally section 8concludes this study.

2. LITERATURE REVIEW

Static code analysis has been a hot topic for researchers to achieve several objectives, such as exposing the source code flaws, testing, privacy, and security investigation.
Li et al. [4] showed that there is a research trend of static analysis in Android apps specially in security, privacy, and testing fields. More specifically, in their study they identified eight topics the literature has been focusing on a) Private Data Leaks, b) Vulnerabilities, c) Permission Misuse, d) Energy Consumption, e) Clone Detection, f) Test Cases Generation, g) Code Verification, and h) Cryptography Implementation Issues, like [30]–[44]. Moreover, recent studies analyzed Android source code from different aspects, like apps lifecycle[12], [13], and redundant apps detection [45].
Researchers started studying O-O design patterns for a relatively long time [46], and their interest continues until this very day. Some studies have addressed O-O design patterns to highlight their importantance in software maintainability, modularity, stability, re-usability, and quality [15], [16], [19]. These studies show that O-O design patterns have positive affect over the software quality in general.
One of these studies conducted by Panca, Mardiyanto, and Hendradjaya [19], implemented a case study of three different apps categories*learning, health, and survey*. Once without using O-O design patterns and another with the use of O-O design patterns. Their results show that the use of design patterns improves the apps maintainability and modularity.
Another study by Prabhakar et al. [18] studied the effect of design patterns in data mining systems. A three-layered architecture component was analyzed to expose this relationship, and finally prove that using design patterns in the right circumstances will relatively improve the system quality.
Another research trend has addressed O-O design patterns from detection perspective[1], [47]–[54]. Some of these studies used manual tagging to detect design patterns, others used machine learning techniques and ontology, while some other tools used similarity scoring.
Al-Obeidallah, Petridis, and Kapetanakis [20] compared different O-O design patterns detection approaches, then show that not all approaches manage to capture all GoF design patterns. Few studies until this day managed to capture all 23 of GoF design patterns and they are [1], [3], [55], [56].
Oruc, Akal, and Sever [48] create new tool (DesPaD), which extract design patterns by converting the source code into a graph model to visually extract them, they compared their results with related by applying the tool on four different source codes.
A study by Derezinska and Byczkowski [21] performed some enhancements on design patterns detection for C# applications. Their enhancements were applied on the approach produced by [57], which is also developed for C# apps design patterns detection. Then they compared the enhanced version with the original one.
Yu, Zhang, and Chen [3] developed a new approach to detect all 23 of GoF design patterns. This approach divides the problem of detecting design patterns into smaller problems, by defining 15 sub-pattern that combining one, two, or three of them will formulate one design pattern. These sub-patterns have been built using four kind of relations between O-O *classes inheritance, association, aggregation, and dependence*, which can be easily captured among the classes. Furthermore, Yu et al. [58] enhanced their approach by extending the original one with an improved search order, which makes it start from the most representative class and avoid all irrelevant classes, to reduce the search space.
However, as all other studies discussed design patterns detection in a desktop or web languages like Java, C++, C#, etc. and none of them handles this dilemma for mobile languages. The previous study of our research [1] is the only study that discusses O-O design patterns detection for Android source code, and has been tested to detect all 23 of GoF design patterns.
This research will extend PatRoid to be able to work with several Android app, and then use it to analyze over 1400 Android apps for the usage of O-O design patterns.

3. METHODOLOGY

3.1 Research Questions

This study aims to reveal the extent of usage for O-O design patterns for Android apps in general, and to study the relation of using design patterns for specific Android apps categories. That been said, we have formulated the follow research question:

RQ1) To what extent are design patterns applied in Android apps?

The sample apps that have been analyzed were categorized for 31 different categories. In respect to these categories we got motivated to know if the design patterns found in each Android app are related to its category or not. Where we can know if there are categories that are more mature than others, which leads us to the second research question:

RQ2) Are design patterns applied in certain Android app categories?

3.2 Data Collection

F-Droid is considered as a free Android apps store. In addition, it contains links for all the Android apps source code that it hosts, which makes it a huge repository for a big representative sample of Android apps, specially to this study [59].

This repository has been used in recent researches such as [12], [60], [61]. In this research, we collected a sample of1427 Android apps hosted by GitHub and Bitbucket. Later on, allthese apps were manuallymapped to Google Play to identify each app category, which resulted in a total of 31 Categories. Table 1 shows the numbers of Android apps collected in respect to their categories.

Table 1: Android Apps Categories

Category	Number of Apps
Tools	337
Games	223
Productivity	147
Personalization	80
Communication	74
Family	70
Music and Audio	54
Entertainment	39
Video Players and Editors	39
Books and Reference	39
Lifestyle	38
Health and Fitness	34
Finance	33
Travel and Local	29
Business	28
Photography	28
Social	27
Maps and Navigation	21
Weather	17
News and Magazines	16
Libraries and Demo	12
Food and Drink	7
Medical	6
Shopping	6
Art and Design	5
Auto and Vehicles	5
Beauty	5
Comics	3
Parenting	2
House and Home	2
Dating	1
Total	1427

3.3 PatRoid

PatRoid is a python open source framework to detect design patterns in Android app [1]. As an input PatRoid takes the Android app project directory to be analyzed, or it can work with the relational model for that specific app.

It starts by gathering AndroidManifest XML file that contains the app activities information, and all the Java classes based on each activity.

After categorizing the java classes based on the activities, PatRoid prepares a relational model from four relations among the classes, and creates an XML model describes these relations, Inheritance, Association, Aggregation, and Dependency.

The relational model yielded is taken as an input to extract sub-patterns instances. These sub-patterns consist form one or more relations that can be aggregated later to formulation design patterns. PatRoid extracts fifteen sub-patterns from the relational model, Aggregation Parent Inherited, Common Inheritance, Dependency Child Inheritance, Dependency Parent Inherited, Indirect Inheritance Aggregation, Inheritance Aggregation, Inheritance Association, Inheritance Child Association, Inheritance Child Dependency, Inheritance Parent Aggregation, Inheritance Parent Association, Inheritance Parent Dependency, Multi-Level Inheritance, Self-Aggregation and finally Self-Association.

After all sub-patterns are extracted, PatRoid aggregates each group of sub-patterns based on a predefined combination to formulate each design pattern. For example, Singleton design pattern is detected by finding Self-Association sub-pattern, while Visitor design pattern is detected by aggregating Aggregation Parent Inherited, Parent Inherited and Inheritance Child Dependency sub-patterns, and so on.

Further description over PatRoid structure is illustrated in Figure 1which shows the different components of PatRoid. The numbers appear on the figure indicate PatRoid components states. The first state shows the part where the tested Android app source code is being parsed. The second state is a map of relations parsed is returned to create a relational model in state three and store it in a new XML model in state four. The relation XML model is given to the component where extracting sub-patterns is done as shown in states five and six. Then the component in which these sub-patterns being aggregated to detect design patterns appear in states seven and eight. To finally dump the design patterns detection report.

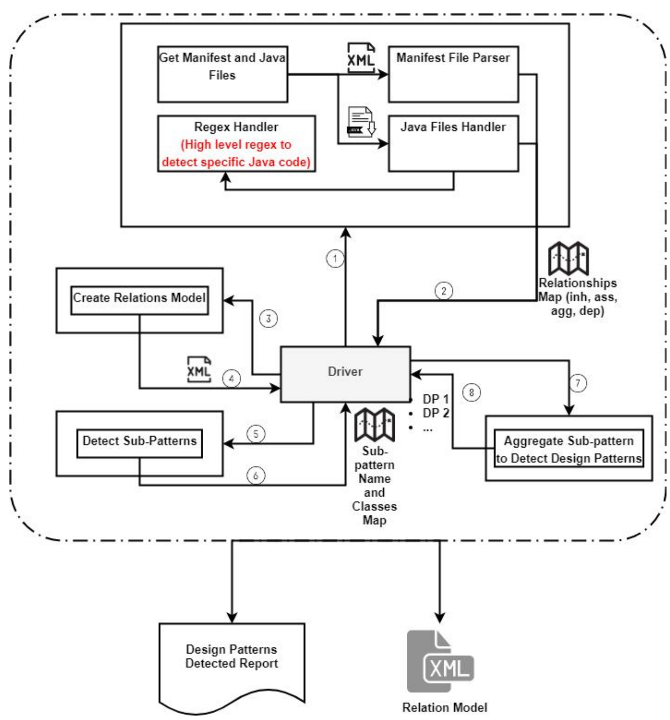

Figure 1: PatRoid Structure[1]

3.4 Extended PatRoid

In this study, PatRoid has been extended by adding the ability to work with any directory that contains one or more Android apps source code. As a result, PatRoid will support both working with one single Android app or with directory full with Android apps. This change requires adding support for all kind of Android app projects and exceptions, like single activity apps, Android apps with missing AndroidMainfest, Android apps with missing Java files, etc.

Additionally, in order to enhance PatRoid performance to avoid memory crash specially when analyzing big set of Android apps as in this study. This study extends PatRoid output methodology to dump every analyzed app dictionary to JSON file, which adds the ability to analyze PatRoid results offline.

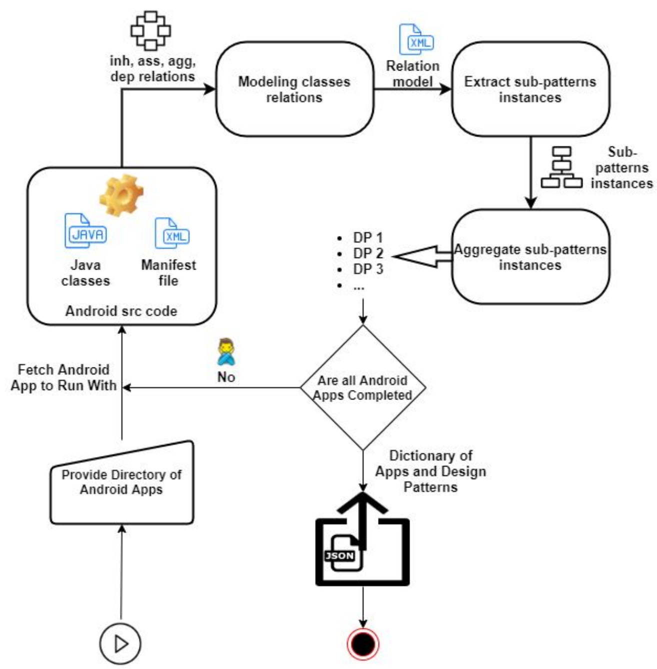

Figure 2: Extended PatRoid Workflow

Figure 2 shows the new workflow of PatRoid. The first step is to take the directory that contains Android apps source code repositories, to start fetching the apps one by one, and then creates the relations among classes to build the relational model. After that it will extract the sub-patterns instances, so they'll be ready for the aggregation process to detect design patterns. In the end it will save the design patterns detected details (like design pattern classes, and combination of sub-patterns) into a dictionary of all Android apps design patterns. Then finally check if all Android apps under the root directory are finished, and save the dictionary into a JSON file.

4. RESULTS

Results are divided in two parts. The first part shows the general usage of each design pattern over the whole sample, and the second part shows design patterns usage distribution over each apps' category.

4.1 Design Patterns Usage

Results in this part shows the numbers of Android apps that uses each design pattern. In addition, the results for not using any design patterns is shown as well. All these results appear in Table 2, where all design patterns are sorted based on the number of Android apps that use it, also it shows the usage percentage for each design patterns that is the number of Android apps that use each design pattern from all the Android apps sample.

Table 2: Design Patterns Usage

Design Pattern	Number of Android Apps Using it	Usage Percentage
Singleton	791	55%
Template	593	42%
Adapter	433	30%
Proxy	376	26%
Composite	248	17%
Abstract Factory	187	13%
Chain of Responsibility	171	12%
Factory	165	12%
Façade	149	10%
Mediator	146	10%
Strategy	144	10%
State	144	10%
Flyweight	143	10%
Prototype	130	9%
Builder	119	8%
Command	119	8%
Memento	107	7%
Observer	79	6%
Bridge	68	5%
Iterator	28	2%
Visitor	21	1%
Interpreter	13	1%
Decorator	4	0%
No Design Patterns	546	38%

Additionally, a bar chart showing the design patterns usage is presented in Figure 3, the chart shows the design patterns on the horizontal axis, and the usage percentage on the vertical axis.

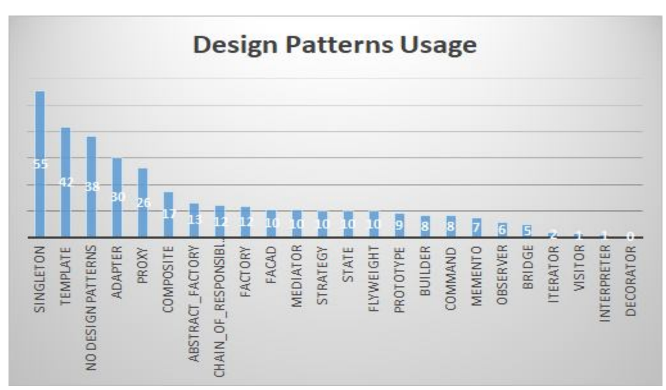

Figure 3: Design Patterns Usage Chart

4.2 Design Patterns Usage Per Category

As discussed before the sample of Android apps is categorized into 31 categories (as shown in Table 1: Android Apps Categories). The results of this part show the design patterns distribution per each category, or in other words, what kind of design patterns are used in each Android apps category.

First of all, in Table 3 the number of design patterns applied by each category is shown, where the number of 23 design patters is an indication that all of the 23 GoF design patterns were detected in the Android apps that fall under this category. Secondly, a more detailed table that shows each Android apps category with specific list of design patterns detected in it, is provided in Table 4, where mark indicates that the Android apps category in that row has at least one instance of the design pattern in that column.

Table 3: Design Patterns Usage Per Category

Category	Number of Design Patterns Used
Lifestyle	23
Tools	23
Productivity	22
Communication	22
Books and Reference	22
News and Magazines	21
Entertainment	21
Music and Audio	21
Games	21
Video Players and Editors	21
Libraries and Demo	20
Shopping	20
Finance	20
Health and Fitness	20
Business	20
Personalization	20
Photography	19
Social	19
Family	19
Travel and Local	17
Weather	14
Maps and Navigation	13
Art and Design	13
Beauty	8
Comics	8
Medical	8
Auto and Vehicles	7
Parenting	5
Food and Drink	1
Dating	1
House and Home	1

Moreover, a bar chart that describes each category and the number of design patterns used in that category is shown in Figure 4, where the vertical axis shows the design patterns numbers and the horizontal axis for the Android apps categories.

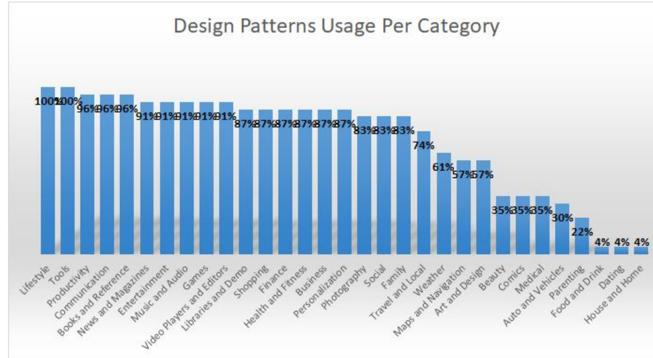

Figure 4: Design Patterns Distribution Per Category

5. DISCUSSION

5.1 RQ1 -To what extent are design patterns applied in Android apps?

Answering the first research question, gives us a lot of information about the what level of re-usability, maintainability, and evolution Android apps have, also it gives us information about the Android apps developers maturity.

The results shown in Table 2 and illustrated in Figure 3 describe the number of Android apps that use each design pattern, it is apparent that Singleton design patterns is the top design pattern used. Over than 55% of Android apps are using this design pattern, which make sense, because this design pattern is widely used due to the need of creating a single activity, other than the usual use of this design pattern. Secondly, the Template design pattern appears as the second design pattern with 42% of Android apps are using it, this because template describes the known IS-A relation, where two or more classes are inherited from the same super class, or in other words Common Inheritance sub-pattern.

As for the rest of GoF design patterns, the usage percentage keep decreasing to reach almost 0% for the Decorator design pattern. Decorator design pattern provides the code with a fixable inheritance relationship for objects by changing the functionalities of objects in run-time. This design pattern implementation requires a combination of Common Inheritance and Inheritance Aggregation sub-patterns or

Table 4: Design Patterns Distribution Per Category

	abstract factory	adapter	bridge	builder	chain of responsibility	command	composite	decorator	façade	factory	flyweight	interpreter	iterator	mediator	memento	observer	prototype	proxy	singleton	state	strategy	template	visitor
Productivity	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
News and Magazines	X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	
Entertainment	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Libraries and Demo	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	
Food and Drink																			X				
Dating																			X				
Travel and Local	X	X		X	X	X	X		X	X	X	X		X			X	X	X	X	X	X	
Shopping	X	X	X	X	X	X	X		X	X	X	X		X	X	X	X	X	X	X	X	X	
Finance	X	X	X	X	X	X	X		X	X	X			X	X	X	X	X	X	X	X	X	X
Beauty	X	X					X		X	X								X	X			X	
Photography	X	X	X	X	X	X	X		X	X	X			X	X	X	X	X	X	X	X	X	
Parenting		X			X													X	X			X	
Auto and Vehicles	X	X								X				X				X	X			X	
Maps and Navigation	X	X		X	X	X	X		X	X					X	X		X	X			X	
Music and Audio	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	X
Lifestyle	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Art and Design	X	X		X	X	X	X		X	X				X		X		X	X			X	
Comics	X	X					X		X	X								X	X			X	
Medical	X	X			X		X							X				X	X			X	
Games	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	
Social	X	X	X	X	X	X	X		X	X	X			X	X	X	X	X	X	X	X	X	
Health and Fitness	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	
Business	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	
Family	X	X	X	X	X	X	X		X	X	X			X	X	X	X	X	X	X	X	X	
Communication	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Video Players and Editors	X	X	X	X	X	X	X		X	X	X		X	X	X	X	X	X	X	X	X	X	X
Weather	X	X		X	X	X	X		X	X				X	X	X		X	X			X	
Personalization	X	X	X	X	X	X	X		X	X	X	X		X	X	X	X	X	X	X	X	X	
House and Home																			X				
Tools	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Books and Reference	X	X	X	X	X	X	X		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Common Inheritance, Inheritance Aggregation and Multi-Level Inheritance sub-patterns. Both combinations require strong coding skills to accomplish. The lack of this kind of design patterns, which is part of other 19 design patterns that their usage percentage didn't reach 20% of the total usage among this study sample, in addition to 38% of the Android apps that don't use any of the 23 GoF design patterns. These results can lead us to conclude that the Android development community still not mature enough, also Android apps in general cannot be consider as highly mature, maintainable, nor reusable. This conclusion answers our first research question.

5.2 RQ2 -Are design patterns applied in certain Android app categories?

Answering this research question gives us a clear comparison between Android apps categories in term of using design patterns, which eventually leads us to conclude what category is more mature than the other.

Each of Table 3 and Table 4 shows a detailed information about the usage of design patterns per each Android category. These details are summarized in Figure 4 and from there it can be observed that there are some categories that use 100% of the total 23 GoF design patterns (Lifestyle and Tools Categories), followed by eight categories that use more than 90% of design patterns, then nine categories use more than

80% if design patterns. After that the usage number start to rapidly decrease with only one category uses more than 70% if design patterns, one that uses more than 60%, two categories use more than 50%, four use more than 30%, one uses more than 20%, and finish with three categories that use less than 10% of the total 23 GoF design patterns.

Based on these results it appears that there is high diversity over Android categories maturity, for instance, the Tools category has the highest amount of Android apps, which means that it contains vary maturity between the Android apps belong to this category that leads to fully cumulative amount of design patterns used in this category apps, and this conclusion answers the second research question.

6. FUTURE WORK

In extend this study, PatRoid can be enhanced more in the accuracy part, and in the same context different approaches can be used with PatRoid, like using machine learning methods instead of isomorphism approach, and compare the results between this study and the new PatRoid.

7. THREATS TO VALIDITY

Our sample included more than 1400 Android apps available over F-Droid. Extending the sample number will indeed provide better and more accurate results. Further, F-Droid is the official hub for Android source code, but the number of apps available is very small compared to the number of apps available on Google Play.

8. CONCLUSION

As a conclusion, this study shows that the usage of design patterns in general in Android apps is still needs improvements specially with only four design patterns has a usage percentage between 20% to 55%, and the rest nineteen design patterns are used in a percentage less than 20%, in addition to 38% of the Android apps that runs in this study are not using any design patterns.

Additionally, the study shows that there is high diversity over the different Android apps categories which means that there are some categories that more mature than others.

This study has proven that the area of Android apps development is still lack to proper usage of design patterns.

This conclusion means that there is still a lot to be done in this area, especially with the growth toward using Android in more critical fields.

Design patterns have proven it importance specially for better software re-usability, maintainability, and evolution. Which means that using it is very important to spread the awareness of its important in Android different fields.

REFERENCES

1. D. Rimawi and S. Zein, "A Model Based Approach for Android Design Patterns Detection," in *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2019, pp. 1–10. https://doi.org/10.1109/ISMSIT.2019.8932921
2. "IDC - Smartphone Market Share - OS," *IDC: The premier global market intelligence company*, Jan. 20, 2020. https://www.idc.com/promo/smartphone-market-share (accessed Apr. 19, 2020).
3. D. Yu, Y. Zhang, and Z. Chen, "A comprehensive approach to the recovery of design pattern instances based on sub-patterns and method signatures," *J. Syst. Softw.*, vol. 103, pp. 1–16, 2015. https://doi.org/10.1016/j.jss.2015.01.019
4. L. Li *et al.*, "Static analysis of android apps: A systematic literature review," *Inf. Softw. Technol.*, vol. 88, pp. 67–95, 2017. https://doi.org/10.1016/j.infsof.2017.04.001
5. "Android Apps on Google Play." https://play.google.com/store/apps?hl=en (accessed May 13, 2019).
6. M. Prakash, U. Gowshika, and T. Ravichandran, "A smart device integrated with an android for alerting a person's health condition: Internet of Things," *Indian J. Sci. Technol.*, vol. 9, no. 6, pp. 1–6, 2016.
7. F. M. Kundi, A. Habib, A. Habib, and M. Z. Asghar, "Android-based health care management system," *Int. J. Comput. Sci. Inf. Secur.*, vol. 14, no. 7, p. 77, 2016.
8. S. Papadakis, M. Kalogiannakis, and N. Zaranis, "Educational apps from the Android Google Play for Greek preschoolers: A systematic review," *Comput. Educ.*, vol. 116, pp. 139–160, 2018. https://doi.org/10.1016/j.compedu.2017.09.007
9. P. Kirci and M. O. Kahraman, "Game based education with android mobile devices," in *Modeling, Simulation, and Applied Optimization (ICMSAO), 2015 6th International Conference on*, 2015, pp. 1–4.
10. S. Bojjagani and V. N. Sastry, "Stamba: Security testing for Android mobile banking apps," in *Advances in Signal Processing and Intelligent Recognition Systems*, Springer, 2016, pp. 671–683.
11. J. B. ek Jørgensen, B. Knudsen, L. Sloth, J. R. Vase, and H. B. erbak Christensen, "Variability Handling for Mobile Banking Apps on iOS and Android," in *Software Architecture (WICSA), 2016 13th Working IEEE/IFIP Conference on*, 2016, pp. 283–286. https://doi.org/10.1109/WICSA.2016.29
12. N. Hoshieah, S. Zein, N. Salleh, and J. Grundy, "A static analysis of android source code for lifecycle development usage patterns," *J. Comput. Sci.*, vol. 15, no. 1, pp. 92–107, 2019.
13. S. Zein, N. Salleh, and J. Grundy, "Static analysis of android apps for lifecycle conformance," in *Information Technology (ICIT), 2017 8th International Conference on*, 2017, pp. 102–109.
14. B. B. Mayvan, A. Rasoolzadegan, and Z. G. Yazdi, "The state of the art on design patterns: A systematic mapping of the literature," *J. Syst. Softw.*, vol. 125, pp. 93–118, 2017. https://doi.org/10.1016/j.jss.2016.11.030
15. M. Vokáč, W. Tichy, D. I. Sjøberg, E. Arisholm, and M. Aldrin, "A controlled experiment comparing the maintainability of programs designed with and without design patterns—a replication in a real programming environment," *Empir. Softw. Eng.*, vol. 9, no. 3, pp. 149–195, 2004.

16. A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou, and P. Avgeriou, "The effect of GoF design patterns on stability: a case study," *IEEE Trans. Softw. Eng.*, vol. 41, no. 8, pp. 781–802, 2015.
17. D. Feitosa, R. Alders, A. Ampatzoglou, P. Avgeriou, and E. Y. Nakagawa, "Investigating the effect of design patterns on energy consumption," *J. Softw. Evol. Process*, vol. 29, no. 2, 2017.
18. N. P. Prabhakar, D. Rani, A. H. Narayanan, and M. V. Judy, "Analyzing the Impact of Software Design Patterns in Data Mining Application," in *Artificial Intelligence and Evolutionary Computations in Engineering Systems*, Springer, 2017, pp. 73–80.
19. B. S. Panca, S. Mardiyanto, and B. Hendradjaya, "Evaluation of software design pattern on mobile application based service development related to the value of maintainability and modularity," in *Data and Software Engineering (ICoDSE), 2016 International Conference on*, 2016, pp. 1–5. https://doi.org/10.1109/ICODSE.2016.7936132
20. M. G. Al-Obeidallah, M. Petridis, and S. Kapetanakis, "A survey on design pattern detection approaches," *Int. J. Softw. Eng. IJSE*, vol. 7, no. 3, 2016.
21. A. Derezińska and M. Byczkowski, "Enhancements of detecting gang-of-four design patterns in C# programs," in *International Conference on Information Systems Architecture and Technology*, 2018, pp. 277–286.
22. A. De Lucia, V. Deufemia, C. Gravino, and M. Risi, "An Eclipse plug-in for the detection of design pattern instances through static and dynamic analysis," in *2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–6.
23. C. Liu, B. van Dongen, N. Assy, and W. M. van der Aalst, "Poster: A General Framework to Detect Behavioral Design Patterns," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 234–235. https://doi.org/10.1145/3183440.3194947
24. B. B. Mayvan and A. Rasoolzadegan, "Design pattern detection based on the graph theory," *Knowl.-Based Syst.*, vol. 120, pp. 211–225, 2017.
25. N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. T. Halkidis, "Design pattern detection using similarity scoring," *IEEE Trans. Softw. Eng.*, vol. 32, no. 11, pp. 896–909, 2006.
26. D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection," in *11th IEEE International Workshop on Program Comprehension, 2003.*, 2003, pp. 94–103.
27. J. Dong, Y. Sun, and Y. Zhao, "Design pattern detection by template matching," in *Proceedings of the 2008 ACM symposium on Applied computing*, 2008, pp. 765–769. https://doi.org/10.1145/1363686.1363864
28. E. G. Nilsson, "Design patterns for user interface for mobile applications," *Adv. Eng. Softw.*, vol. 40, no. 12, pp. 1318–1328, 2009.
29. J. L. Wesson, N. L. O. Cowley, and C. E. Brooks, "Extending a mobile prototyping tool to support user interface design patterns and reusability," in *Proceedings of the South African Institute of Computer Scientists and Information Technologists*, 2017, p. 39. https://doi.org/10.1145/3129416.3129444
30. Y. Chang, B. Liu, L. Cong, H. Deng, J. Li, and Y. Chen, "Vulnerability Parser: A Static Vulnerability Analysis System for Android Applications," in *Journal of Physics: Conference Series*, 2019, vol. 1288, no. 1, p. 012053.
31. H. Song, D. Lin, S. Zhu, W. Wang, and S. Zhang, "ADS-SA: System for Automatically Detecting Sensitive Path of Android Applications Based on Static Analysis," in *International Conference on Smart City and Informatization*, 2019, pp. 309–322.
32. R. Bonett, K. Kafle, K. Moran, A. Nadkarni, and D. Poshyvanyk, "Discovering flaws in security-focused static analysis tools for android using systematic mutation," in *27th ${$USENIX$}$ Security Symposium (${$USENIX$}$ Security 18)*, 2018, pp. 1263–1280.
33. S. Roy, D. Chaulagain, and S. Bhusal, "Static Analysis for Security Vetting of Android Apps," in *From Database to Cyber Security*, Springer, 2018, pp. 375–404.
34. H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, 2018. https://doi.org/10.1016/j.neucom.2017.07.030
35. J. Zhao, A. Albarghouthi, V. Rastogi, S. Jha, and D. Octeau, "Neural-augmented static analysis of Android communication," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 342–353.
36. S. Arzt *et al.*, "Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps," *Acm Sigplan Not.*, vol. 49, no. 6, pp. 259–269, 2014.
37. W. Klieber, L. Flynn, A. Bhosale, L. Jia, and L. Bauer, "Android taint flow analysis for app sets," in *Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis*, 2014, pp. 1–6.
38. L. Li, A. Bartel, J. Klein, and Y. Le Traon, "Automatically exploiting potential component leaks in android applications," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2014 IEEE 13th International Conference on*, 2014, pp. 388–397.
39. L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 229–240.
40. D. Octeau, D. Luchaup, M. Dering, S. Jha, and P. McDaniel, "Composite constant propagation: Application to android inter-component communication analysis," in *Proceedings of the 37th*

International Conference on Software Engineering-Volume 1, 2015, pp. 77–88. https://doi.org/10.1109/ICSE.2015.30

41. A. Bartel, J. Klein, M. Monperrus, and Y. Le Traon, "Static analysis for extracting permission checks of a large scale framework: The challenges and solutions for analyzing android," *IEEE Trans. Softw. Eng.*, vol. 40, no. 6, pp. 617–632, 2014.
42. D. Li, S. Hao, W. G. Halfond, and R. Govindan, "Calculating source line level energy information for android applications," in *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 2013, pp. 78–89.
43. J. Crussell, C. Gibler, and H. Chen, "Attack of the clones: Detecting cloned applications on android markets," in *European Symposium on Research in Computer Security*, 2012, pp. 37–54.
44. M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, "An empirical study of cryptographic misuse in android applications," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 73–84. https://doi.org/10.1145/2508859.2516693
45. Y. Jiang, Q. Bao, S. Wang, X. Liu, and D. Wu, "RedDroid: Android application redundancy customization based on static analysis," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, 2018, pp. 189–199.
46. M. Grand, *Patterns in Java: a catalog of reusable design patterns illustrated with UML*. John Wiley & Sons, 2003.
47. A. K. Dwivedi, A. Tirkey, and S. K. Rath, "Applying learning-based methods for recognizing design patterns," *Innov. Syst. Softw. Eng.*, vol. 15, no. 2, pp. 87–100, 2019.
48. M. Oruc, F. Akal, and H. Sever, "Detecting design patterns in object-oriented design models by using a graph mining approach," in *Software Engineering Research and Innovation (CONISOFT), 2016 4th International Conference in*, 2016, pp. 115–121.
49. S. Uchiyama, A. Kubo, H. Washizaki, and Y. Fukazawa, "Detecting design patterns in object-oriented program source code by using metrics and machine learning," *J. Softw. Eng. Appl.*, vol. 7, no. 12, p. 983, 2014. https://doi.org/10.4236/jsea.2014.712086
50. A. Chihada, S. Jalili, S. M. H. Hasheminejad, and M. H. Zangooei, "Source code and design conformance, design pattern detection from source code by classification approach," *Appl. Soft Comput.*, vol. 26, pp. 357–367, 2015.
51. M. Elaasar, L. C. Briand, and Y. Labiche, "VPML: an approach to detect design patterns of MOF-based modeling languages," *Softw. Syst. Model.*, vol. 14, no. 2, pp. 735–764, 2015.
52. A. D. Lucia, V. Deufemia, C. Gravino, and M. Risi, "Detecting the Behavior of Design Patterns through Model Checking and Dynamic Analysis," *ACM Trans. Softw. Eng. Methodol. TOSEM*, vol. 26, no. 4, p. 13, 2018. https://doi.org/10.1145/3176643
53. M. Elaasar, "Detecting design patterns in models by utilizing transformation language," Apr. 2014.
54. K. Kumar and S. Jarzabek, "Detecting design similarity patterns using program execution traces," in *Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*, 2014, pp. 55–56.
55. I. Philippow, D. Streitferdt, M. Riebisch, and S. Naumann, "An approach for reverse engineering of design patterns," *Softw. Syst. Model.*, vol. 4, no. 1, pp. 55–70, 2005.
56. H. Kim and C. Boldyreff, "A method to recover design patterns using software product metrics," in *International Conference on Software Reuse*, 2000, pp. 318–335.
57. A. Nagy and B. Kovari, "Programming language neutral design pattern detection," in *2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, 2015, pp. 215–219. https://doi.org/10.1109/CINTI.2015.7382925
58. D. Yu, P. Zhang, J. Yang, Z. Chen, C. Liu, and J. Chen, "Efficiently detecting structural design pattern instances based on ordered sequences," *J. Syst. Softw.*, vol. 142, pp. 35–56, 2018.
59. "F-Droid - Free and Open Source Android App Repository." https://f-droid.org/en/ (accessed Apr. 19, 2020).
60. K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 2016, pp. 468–471.
61. 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 *Proceedings of the 3rd ACM SIGSOFT International Workshop on App Market Analytics*, 2019, pp. 8–14. https://doi.org/10.1145/3340496.3342759