A Comparative Study of Smartphone and Smartwatch Apps

Xiao Chen
Faculty of Information Technology
Monash University
Clayton, Australia
xiao.chen@monash.edu

Wanli Chen
Faculty of Information Technology
Monash University
Clayton, Australia
wanli.chen@monash.edu

Kui Liu
College of Computer Science and Technology
Nanjing University of Aeronautics and Astronautics
Nanjing, China
kui.liu@nuaa.edu.cn

Chunyang Chen
Faculty of Information Technology
Monash University
Clayton, Australia
chunyang.chen@monash.edu

Li Li
Faculty of Information Technology
Monash University
Clayton, Australia
li.li@monash.edu

ABSTRACT

Despite that our community has spent numerous efforts on analyzing mobile apps, there is no study proposed for characterizing the relationship between smartphone and smartwatch apps. To fill this gap, we present to the community a comparative study of smartphone and smartwatch apps, aiming at understanding the status quo of cross-phone/watch apps. Specifically, in this work, we first collect a set of cross-phone/watch app pairs and then experimentally look into them to explore their similarities or dissimilarities from different perspectives. Experimental results show that (1) Approximately, up to 40% of resource files, 30% of code methods are reused between smartphone/watch app pairs, (2) Smartphone apps may require more than twice as many as permissions and adopt more than five times as many as user interactions than their watch counterparts, and (3) Smartwatch apps can be released as either standalone (can be run independently) or companion versions (i.e., have to co-work with their smartphone counterparts), for which the former type of apps tends to require more permissions and reuse more code, involve more user interactions than the latter type. Our findings can help developers and researchers understand the ecosystem of smartwatch apps and further gain insight into migrating smartphone apps for smartwatches.

KEYWORDS

Android, smartwatch, mobile software engineering, static code analysis

ACM Reference Format:
Xiao Chen, Wanli Chen, Kui Liu, Chunyang Chen, and Li Li. 2021. A Comparative Study of Smartphone and Smartwatch Apps. In *The 36th ACM/SIGAPP Symposium on Applied Computing (SAC '21), March 22–26, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3412841.3442023

SAC '21, March 22–26, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8104-8/21/03…$15.00
https://doi.org/10.1145/3412841.3442023

1 INTRODUCTION

The Smartwatch market is booming. Many app developers and service providers have provided smartwatch versions for their smartphone apps. Indeed, the world has witnessed more than 79 million smartwatches sold in 2018 [3], making it the most popular wearable devices in the market. Due to its accessibility, people can check and send messages, get notifications, answer phone calls without accessing their phones. Not only smartphone vendors (e.g., Apple and Samsung) and other technology companies (e.g., Fitbit, Garmin) entered into the smartwatch market, traditional watch manufacturers such as Fossil and Casio also joined the smartwatch race.

Despite the increasing momentum of the smartwatch market (especially in terms of the number of smartwatch devices available in the ecosystem), the number of existing smartwatch apps is far less than that of smartphone apps. Indeed, compared with over 2.5 million Android apps and 1.8 million iOS apps published on Google Play and Apple App Store for smartphones, respectively, the numbers of Wear OS (i.e., a version of Android OS designed for smartwatches) apps and WatchOS (i.e., the operating system for the Apple Watch) apps for smartwatches are only around 4,000 and 15,000, respectively [11].

Taking the considerable potential of smartwatch's market share into consideration, we felt that it is quite strange to observe the aforementioned massive difference between the number of smartphone and smartwatch apps. The rationale is not clear at the moment, although app developers would have liked to offer the same app for multiple mobile platforms in order to attract as many users as possible [16]. To the best of our knowledge, developing the smartwatch version of a smartphone app may not be straightforward. The user's requirements for a smartphone app and a smartwatch app might be different. Users may expect a complete set of features from a smartphone messaging app, such as text and video chat, file transfer, etc. However, due to the limited screen size and hardware capability (e.g., lack of a camera) on the smartwatch, users may only use it to access prompt messages and reply with voice input. It might not be a wise choice as well to copy the onscreen

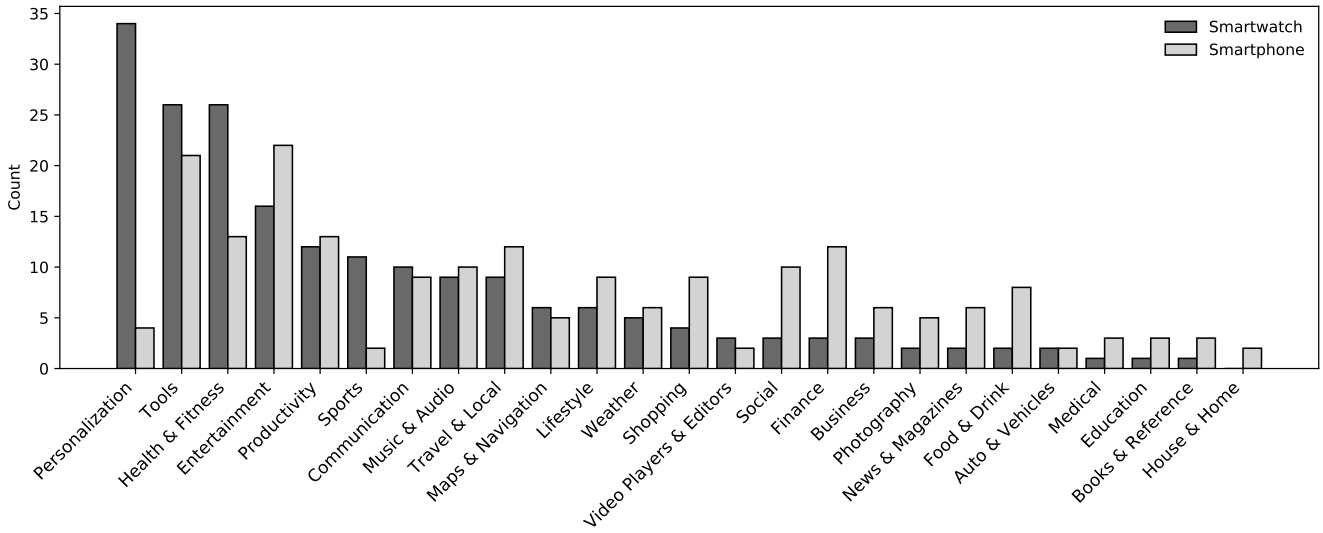


Figure 1: Category Distribution of 200 Most Popular Smartphone and Smartwatch Apps on Google Play.

keyboard and complicated User Interface (UI) design directly from the smartphone app to its smartwatch version.

Unfortunately, despite a significant amount of efforts the community has spent on analyzing mobile apps, including the comparative studies of cross-platform apps [4], the community has not yet explored the realm of comparing smartphone and smartwatch apps. It is still unknown why some apps have been published with smartwatch versions while others are not. For such smartphone apps with smartwatch counterparts, it is also not clear (1) whether their implemented functions are identical or not? and (2) should all the features of smartphone apps be migrated to their smartwatch counterparts? Towards answering the aforementioned questions, there is a strong need to understand the current status quo of the relationship between smartphone and smartwatch apps.

To this end, we present to the community a comparative study of smartphone and smartwatch apps, aiming at providing an overview of the status quo of cross-phone/watch apps. To fulfill this goal, we first resort to various resources (including the official Google Play store and an alternative Android app market APKMirror[1]) to harvest and build a dataset of phone/watch app pairs, i.e., *the same app implemented for Smartphone and Smartwatch devices*. By considering all the apps collected, we are able to construct a dataset with 223 app pairs. After that, we employ a mixed-methods approach using both quantitative and qualitative analyses to experimentally characterize these app pairs. For example, we empirically compare their non-code and code level similarities and investigate their differences in managing user interactions.

With this comparative study, we aim at providing the community an overview of the status quo of the cross-phone/watch apps, helping practitioners and researchers better understand the current development of cross-platform apps (i.e., not only between Android and iOS systems but also between smartphone and smartwatch devices). Notably, we expect our study to be useful for different stakeholders in the mobile ecosystem to understand the similarities and dissimilarities between smartphone and smartwatch app pairs, so as to support them in deciding if it is necessary to introduce smartwatch app versions for their smartphone apps (or vice versa). Furthermore, by comparing the implementation details of phone/watch app pairs, the findings of our study could also be helpful to gain insights into the challenges faced by app developers in migrating smartphone apps to smartwatch versions.

The contributions of this paper include:

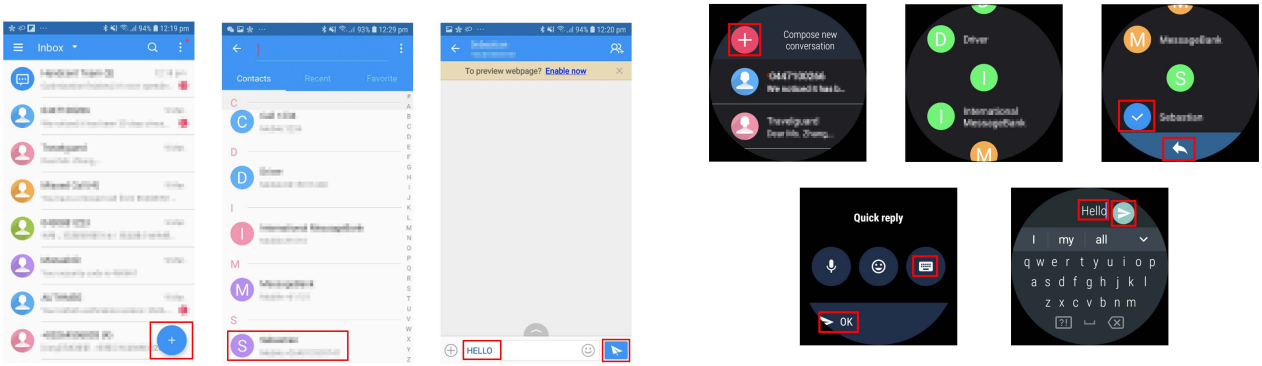
- We collected 223 pairs of Android Smartphone and Smartwatch app pairs from the Google Play and alternative app stores. Our dataset is publicly available [1].
- We conducted quantitative and qualitative analyses on the app pairs from various aspects, including non-code (e.g., metadata, resources, etc.) and code (e.g., components, methods, user interactions, etc.) levels.
- Our empirical findings could be beneficial towards developing an automatic smartphone-to-smartwatch app migration strategy.

The remainder of the paper is organized as follows. Section 2 presents the motivation and preliminary results of our study. Section 3 describes the design of the experiments, including data collection, data characteristics and research questions. Section 4 presents the results and findings of our empirical study, followed by the implications and the threats to validity of the study discussed in Section 5. Section 6 reviews related works, and section 7 conclude the paper.

2 MOTIVATION AND PRELIMINARY STUDY

Due to the differences in the accessibility, smartphones and smartwatches play different roles in people's daily lives. Smartphones have been intensively involved in our daily life and are used for completing routine tasks such as communicating with friends and paying bills. Smartwatches, on the other hand, are designed for on-the-go tasks, such as a quick response to a message. Therefore, it

[1]https://www.apkmirror.com



(a) Smartphone. (b) Smartwatch.

Figure 2: Three steps are needed on a smartphone to send an SMS while two more steps are needed to achieve the same purpose on a smartwatch, despite the fact that the same app (i.e., Messages App) is leveraged.

could be different on how people use these devices. We retrieve the metadata of the 200 most popular apps on smartphone and smartwatch, respectively, from the Google Play Store and present their category distributions in Figure 1. Interestingly, the most popular apps on two platforms fall into different categories. For example, the most popular category in the smartwatch market is *Personalization*, which counts 17% of the top apps. In contrast, only 2% of the top smartphone app falls in this category. For *Health & Fitness* and *Sport* categories, the related smartwatch apps overwhelm the smartphone apps as well. On the other side, the smartphone has much more apps of 14 categories than smartwatch. These empirical results show that there is indeed a difference between the preferred usage scenarios of smartphone and smartwatch apps. This evidence further suggests that certain apps should receive higher priorities to be migrated from smartphone to smartwatch.

Except the quantitative difference of smartphone and smartwatch apps, smartphone apps present qualitative differences against smartwatch apps in various aspects as we observed in the preliminary case study, despite smartphone and smartwatch apps are developed using the same programming language (i.e., Java or Kotlin) and run on the same operating system (i.e., Android). For example, due to the differences in hardware capability and screen size, the users' interaction logic in smartphone and smartwatch is quite different. As a preliminary case study shown in Figure 2, it illustrates that the steps of sending an SMS using the smartphone version and smartwatch version of a Text Messaging app are totally different. With limited screen size, the smartwatch version takes more steps to locate the contact and select the input source in Figure 2(b)). As a result, smartphone apps and smartwatch apps may require different testing strategies to test the code's correctness. In other words, the existing dynamic app analyzers developed for testing smartphone apps may not be readily reusable for testing smartwatch apps. It can also foresee that the functionalities in two scenarios would be different due to the difference in computational power. These differences observed in this preliminary study motivate us to deeply exploit how the apps on the smartphone and the smartwatch differ from each other, so as to obtain valuable findings and insights for helping practitioners to easily implement smartwatch apps as well as adequately proceed smartwatch app analysis.

3 EXPERIMENTAL DESIGN

In this section, we present the research questions of our empirical study (cf. Section 3.1), the dataset we used for answering these questions (cf. Section 3.2), and the characteristics of the harvested dataset (cf. Section 3.3).

3.1 Research Questions

The empirical study aims to find out the similarities and differences between the smartphone version and smartwatch version of the same mobile apps. Specifically, we would like to investigate their relationship by answering the following research questions:

- **RQ1 [Non-Code]**: How similar are the disassembled resource files between smartphone/watch app pairs?
- **RQ2 [Code (Syntax)]**: To what extent is the code of smartphone apps similar to their smartwatch counterparts?
- **RQ3 [Code (User Interaction)]**: How are user interactions varied between smartphone/watch app pairs?

3.2 Data Collection

To the best of our knowledge, our community has not released public datasets containing smartwatch apps that we can directly reuse to fulfill our experiments. We have to collect smartwatch apps from scratch. In this work, we decide to collect smartwatch apps from both the official Google Play store and an alternative app store named APKMirror.

Google Play dataset: It is challenging to collect smartwatch apps from Google Play, as there is no information on their app details pages that indicates whether the app has a smartwatch version. Fortunately, Google Play maintains a list of the top 200 most popular smartwatch apps. Therefore, we collected the listed 200 smartwatch apps and also downloaded their smartphone counterparts.

APKMirror dataset: APKMirror archives Android apps for various devices (e.g., phone, watch, TV, etc.) as well as their historical versions. Through a keyword search, more than 1,700 results were

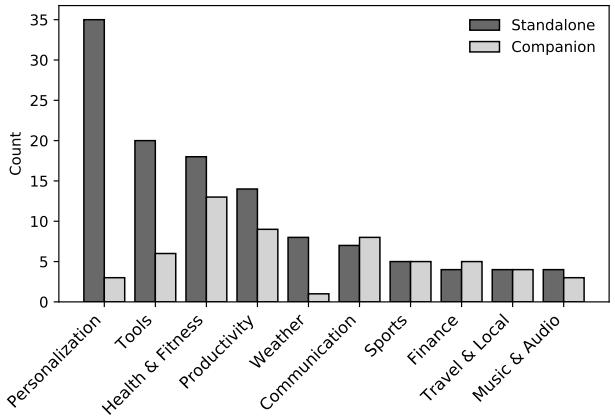


Figure 3: The counts of collected standalone and companion apps by category

returned with "Wear OS" in their names. We then filtered out the historical versions, and finally, 95 phone/watch app pairs were collected.

The two datasets were merged with duplicates removed. Eventually, the dataset for this study consists of 223 phone/watch app pairs.

3.3 Data Characteristics

Table 1 presents the count and average file size of the collected app pairs by category. Categories that have less than three apps were included in the *Others* category. It is observed that the size of smartwatch apps are much smaller than their smartphone counterparts, indicating a possible reduction of resources and functionalities in the smartwatch version. This hypothesis is further confirmed and detailed in Section 4.

Our manual investigation further discovers that smartwatch apps can be released as either *standalone* or *companion* apps. The former type of apps does not require a smartphone counterpart to function while the latter type of apps do require a smartphone-side app to operate. This feature is declared by a boolean value of the compulsory metadata *com.google.android.wearable.standalone* in the Android Manifest file. Out of the total 223 app pairs collected, 145 are standalone apps and 78 are companion apps. While the average size of smartwatch apps (i.e., 5.71 MB) is far smaller than the smartphone versions (i.e., 21.80 MB), the size differences between standalone and companion smartwatch apps are also tremendous. The average file size of the standalone watch apps (i.e., 6.78 MB) is approximately twice as large as the companion watch apps (i.e., 3.72 MB). A further breakdown of the apps into categories is illustrated in Figure 3. Apps in *Personalization*, *Tools*, and *Weather* categories are tend to be used independently, while more *Communication*, *Finance*, *Sports* and *Travel & Local* apps require to pair with the smartphone to function.

4 RESULTS

We now present our experimental results for the aforementioned research questions, respectively.

Table 1: Dataset characteristics.

Category	Count	Average Size (MB)	
		Smartphone	Smartwatch
Personalization	38	17.12	5.82
Health & Fitness	31	30.44	5.97
Tools	26	14.36	5.42
Productivity	23	12.74	3.28
Communication	15	27.92	10.45
Sports	10	26.65	10.10
Finance	9	27.94	3.69
Weather	9	21.05	3.37
Travel & Local	8	31.14	2.60
Music & Audio	7	26.62	6.87
Lifestyle	6	23.53	9.38
Maps & Navigation	6	30.14	5.50
Business	3	33.12	3.57
Education	3	11.33	5.81
Medical	3	12.58	2.32
News & Magazines	3	32.87	3.03
Shopping	3	24.04	7.71
Others	20	18.78	5.18
ALL	223	23.47	5.56

4.1 RQ1: Non-Code Similarity

We firstly investigate the similarities of the resources and the declared permissions between phone/watch app pairs. **Resources** are the non-code assets that can be accessed by the application code, such as images, string values, etc. We try to find out to what extent smartwatch counterparts reuse the resources in their smartphone versions or vice versa. To this end, we compare the file names and the hash values of the file contents in both versions. If an identical file name and its hash value are found in both versions, we identify it as a reused resource file. Figure 4 illustrates the percentage of resource files that have been reused across both smartphone and smartwatch versions. There are, on average, 40% of resource files being reused when developing cross-platform apps. We further inspect the types of these files and discover that the most commonly reused file types are PNG (e.g., UI widgets) and XML (e.g., UI layouts and string values), constituting 49.5% and 12.1% of the total reused files, respectively.

Among various resource files included in an Android app, there is a special file named *AndroidManifest.xml*, the configuration file responsible for defining app features such as *permissions*. App permission system is a front-line mechanism to protect the privacy of Android users. Android apps must declare the permissions in the manifest file before accessing sensitive user data (such as location and contact list), as well as certain system features (such as Internet and camera). In Android security framework, the permissions are grouped into various levels based on their riskiness, which include *normal*, *signature*, and *dangerous*. We leverage Android Asset Packaging Tool [2] (AAPT) to extract the declared permissions.

[2]https://developer.android.com/studio/command-line/aapt2

Table 2: Top 15 permissions in observed app pairs

Permissions in **Smartphone** Apps	Percentage	Permissions in **Smartwatch** Apps	Percentage
INTERNET	96.0%	WAKE_LOCK	83.0%
WAKE_LOCK	94.2%	INTERNET	65.0%
ACCESS_NETWORK_STATE	93.3%	ACCESS_NETWORK_STATE	59.6%
c2dm.permission.RECEIVE	78.0%	VIBRATE	49.8%
VIBRATE	70.9%	vending.BILLING	27.4%
vending.BILLING	65.5%	ACCESS_FINE_LOCATION	27.4%
BIND_GET_INSTALL_REFERRER_SERVICE	64.1%	RECEIVE_BOOT_COMPLETED	24.2%
RECEIVE_BOOT_COMPLETED	61.0%	c2dm.permission.RECEIVE	21.5%
WRITE_EXTERNAL_STORAGE	60.5%	ACCESS_COARSE_LOCATION	19.3%
ACCESS_FINE_LOCATION	57.4%	WRITE_EXTERNAL_STORAGE	19.3%
FOREGROUND_SERVICE	51.6%	wearable.permission.RECEIVE_COMPLICATION_DATA	18.4%
READ_EXTERNAL_STORAGE	47.5%	BODY_SENSORS	17.0%
ACCESS_WIFI_STATE	46.6%	BIND_GET_INSTALL_REFERRER_SERVICE	16.6%
ACCESS_COARSE_LOCATION	42.6%	ACCESS_WIFI_STATE	15.2%
BLUETOOTH	33.2%	PROVIDE_BACKGROUND	14.3%

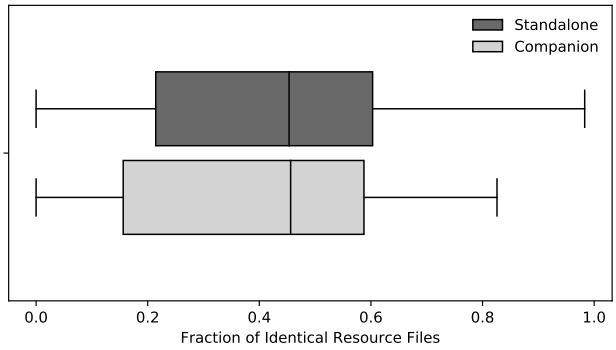


Figure 4: Distribution of the number of permissions in smartphone and smartwatch apps by category.

Table 2 illustrates the top 15 most frequently used permissions in smartphone and smartwatch apps. While most of the permissions in the table are classified as normal permissions, only five permissions `ACCESS_FINE_LOCATION`, `ACCESS_COARSE_LOCATION`, `WRITE_EXTERNAL_STORAGE`, `READ_EXTERNAL_STORAGE`, and `BODY_SENSORS` are listed as dangerous permissions that require the users to manually grant the permissions at run-time. These permissions are required to access the precise and approximate location of the device, write to and read from external storage, and access data from device's in-built sensors. As reported in the table, although there is little difference in the top permission list on both platforms, the number of apps that requested these permissions varies significantly. For instance, `Internet` is one of the most used permissions both in smartphone and smartwatch apps that allows the apps to open network sockets. There are 96.0% of the smartphone apps that have requested the `Internet` permission, while this number is decreased to 65.0% in smartwatch apps. We reviewed the smartphone apps that do not require `Internet` permission and found that these apps are compass apps, password manager apps, and watch face apps that do not retrieve information from the Internet. It is worth mention that two apps do not require Internet access in the smartphone version but added it in the smartwatch version. Through manual analysis, we found that both apps are watch face apps, whose smartwatch versions have added new features of retrieving weather data from the Internet and displaying it on the watch face.

The dangerous permissions `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION` allow an app to access the location of the device, which are very sensitive and always attract significant attention when prompt to the users. It is interesting to find that there are 136 apps requested at least one of the location permissions in their smartphone versions and 93 of them removed the access to location from their smartwatch versions. The reasons for removing the access to location is that the corresponding functions are deleted from the smartwatch version. For example, an SMS app can send out the user's current location in the text message, while this function is removed from the smartwatch version. The reason may be to save battery life of the smartwatch.

Another dangerous permission that has been requested by many smartwatch apps is `BODY_SENSORS`. This permission allows an app to access data from sensors that the users uses to measure what is happening inside their body, such as heart rate. The apps that require this permission mainly fall in *Health & Fitness* and *Personalization* categories. While it seems more legitimate for the apps in the former category to access the sensor data, *Personalization* apps do not have a convincing reason to access these data. After reviewing the *Personalization* apps that requested the `BODY_SENSORS` permission, we found that all of them are watch face apps that display the health data, such as the steps walked on the day, as an additional feature.

Permissions in Smartphone and Smartwatch apps. The average number of permissions found in a smartphone app is 17, which is more than twice as the permissions found in a smartwatch app with an average of 7 permissions declared. Figure 5 shows a break down of the permission distribution by category. It can be observed that, in all categories, the permissions in phone apps are approximately twice as many as the ones in watch apps. Apps in most categories exhibit a similar number of permissions declared in

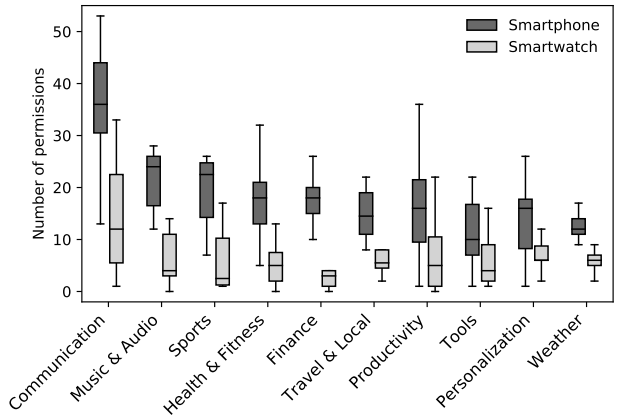


Figure 5: Distribution of the number of permissions in smartphone and smartwatch apps by category.

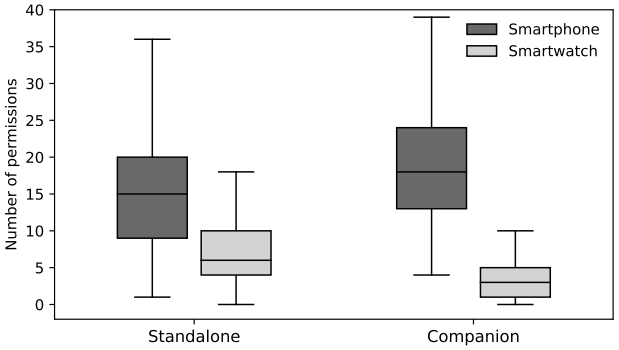


Figure 6: Distribution of the number of permissions in the standalone and companion apps.

each of them (i.e., 9~25 and 2~12 in their phone versions and watch versions, respectively). However, apps from the *Communication* category requires more than twice as many as permissions (i.e., 35 in the smartphone version, and 15 in the smartwatch version) on average in each of them. We took an in-depth look at the apps from the *Communication* category, and found that some permissions, such as writing and reading contacts, sending and receiving SMS/MMS, making phone calls, reading and writing the badges (i.e., showing the count of messages in the app icon), accessing the camera, and recording the voice are much intensively used in communication apps than other categories, and these permissions are highly related to the core functionalities of the communication apps such as sending/receiving messages, making phone calls, etc. The permissions that have been most frequently removed from the smartwatch versions in all categories are in-app billing, receiving notification when the system finishes booting, accessing location, writing to external storage, and accessing WiFi states. This result is also aligned with the observations that on smartwatch apps, power-intensive functions such as getting location information are usually removed.

Permissions in Standalone and Companion Apps. Figure 6 shows the average number of permissions in standalone and companion smartwatch apps. The average numbers of permissions in standalone and companion apps are 9 and 5 per app, respectively, demonstrating that standalone apps tend to use more permissions than the companion apps. Actually, companion apps can hand over functions such as getting/authenticating accounts, reading sync settings/stats, and accessing location data to their smartphone counterpart, hence, they do not require such permissions on their own. On the other hand, standalone apps need to request these permissions to function independently.

Answer to RQ1

- 40% resource files are reused when developing cross-platform apps.
- More than twice as many as permissions are required in the smartphone apps than their smartwatch counterparts.
- Standalone watch apps tend to require more permissions than companion watch apps.

4.2 RQ2: Code Similarity (Syntax)

Developing a smartwatch app can be as complex as developing a brand new app from scratch, or it can be as simple as copy and paste the code from their smartphone counterparts. The smartphone version and smartwatch version of an app usually have similar functionalities, however, due to the differences in hardware capacity, tasks that can be carried out on the smartwatch and the smartphone are different. Therefore, investigating how similar/different the code is in both platforms helps researchers and developers better understand the relationships between smartphone apps and smartwatch apps, and also is a preliminary but essential step towards developing any smartphone-to-smartwatch app migration strategy.

We investigate the similarity of code in smartphone and smartwatch app pairs from both the component and the method levels. Android **components** are the core building blocks of an Android app, which have well defined life cycles. The components include Activity (i.e., a User Interface with back-end class to handle the action performed on User Interface), Service (i.e., a back-end class with no User Interface), Broadcast Receiver (i.e., a component receives and handles broadcast intents from the Android system or other apps), and Content Provider (i.e., a component stores shared data of an app that can be accessed by other apps). The **methods** are the code blocks to perform certain actions, and will be executed when it is invoked. Comparing the component and the method level similarities can well indicate the code similarity between app pairs.

We leverage SimiDroid [18], a tool that compares the pairwise similarities and differences between Android apps. SimiDroid firstly processes the app pairs and represents them in a set of key/value mappings. Depending on the comparison level (i.e., component or method), different features are extracted to form their own keys and values. For component-based comparison, SimiDroid extracts the component name as the key, and other information (e.g., action, data, and category) that represents the characteristics of a component as the value. In method-based comparison, SimiDroid transforms code statements into an intermediate representation named *Jimple* [28]. Method names are then used as the keys, and types of the statements

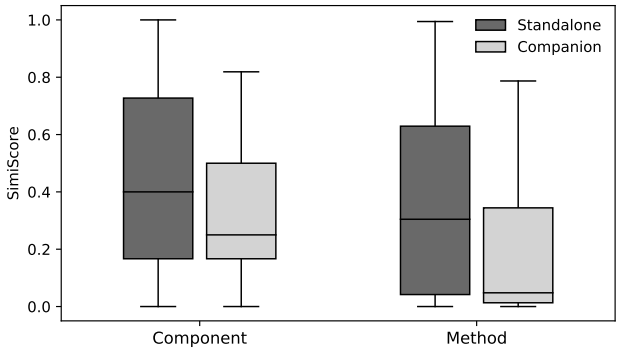


Figure 7: SimiScore of the standalone and companion app pairs

(i.e., if-statement, assign-statement) are used as the values. The extracted key/value mapping pairs (map_1, map_2) are then compared based on the following metrics: (1) **identical**, where there is an exact match between compared key/value pairs; (2) **similar**, where the compared pairs have the same key but different value; (3) **new**, where the key exists only in map_1 (i.e., the component/method only exists in the smartwatch version); and (4) **deleted**, where the key only exists in map_2 (i.e., the component/method only exists in the smartphone version). Finally, the similarity score is calculated as:

$$similarity = max\{\frac{identical}{total - new}, \frac{identical}{total - deleted}\}$$

where

$$total = identical + similar + new + deleted$$

Table 3 shows the average number of each component category in a smartphone app and a smartwatch app. The result suggests that the number of components in a smartphone app is much larger than that in a smartwatch app. This is in line with the intuition that the smartphone apps are usually more complex than a smartwatch app, given their differences in hardware capacities. Specifically, the number of UI screens (i.e., activities) in a smartwatch app is approximately a quarter of that in a smartphone app, while the number of functions running in the background (i.e. services) in a smartwatch app is approximately a half of that in a smartphone app. This result indicates that though an app's smartwatch version reduced both the number of activities and services, it tends to remove more foreground activities than background services. One possible reason may be that from the developers' perspective, compared with the limited computational power, the screen size of the smartwatches may have more impact on how users use the apps.

A further look into the smartwatch apps shows that the number of components in the standalone apps is much greater than the companion apps. The reason could be that the standalone apps would need to run all the functions on its own, while the companion apps can hand over some tasks to their smartphone counterparts and reduce its complexity.

Figure 7 presents the similarity score between the smartphone and smartwatch app pairs. The similarity score indicates the fraction of identical components and methods between the smartphone and the smartwatch versions. As reported, the standalone app pairs has higher similarity score than the companion app pairs. There are, on average, 43% components in the standalone smartwatch

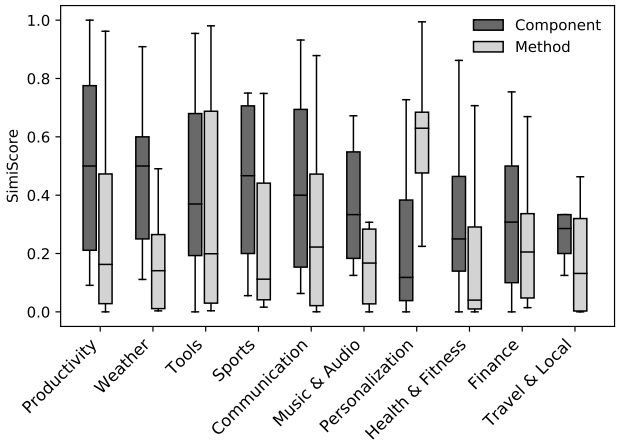


Figure 8: SimiScore of app pairs by category.

Table 3: Average number of components in smartphone apps and smartwatch apps.

		Activity	Service	Broadcast Receiver	Content Provider
Phone	—	42	17	12	5
Watch	Standalone	15	8	3	1
	Companion	7	4	2	1

apps and their smartphone counterparts are identical. This number is decreased to 32% for the companion app pairs. Similar trends can be discovered in the method-level comparison, where 36% and 19% of the methods are directly copied from the smartphone apps to their smartwatch version of standalone and companion app pairs, respectively. This observation suggests that despite the differences in the hardware and functionality, certain amount of code can be migrate from the smartphone version to the smartwatch without any alteration. A standalone smartwatch app is usually a mini version of a smartphone app to some extent, which tends to have identical functions of its smartphone version, therefore has more code reused from the smartphone version. On the other hand, instead of performing the same task, a companion smartwatch app may work as an assistant to the smartphone counterpart, therefore is likely to have less identical methods than the standalone ones.

Table 4 shows a detailed view of average percentage of identical, similar, and new components/methods of a smartwatch app compared with its smartphone counterpart. Despite the components and methods that are identical in both apps, most other components and methods are newly added (i.e., with a different component/method name) rather than modified from existing ones. This finding indicates that developers either exact the same component/method or write completely new component/method, rather than modify part of the code inside a component/method.

Figure 8 shows a breakdown of component/method similarities into different app categories. The component and method level similarities in all categories do not exhibit huge differences, except for the apps in *Personalization* category, whose method-level SimiScore is significantly higher than the other categories. After

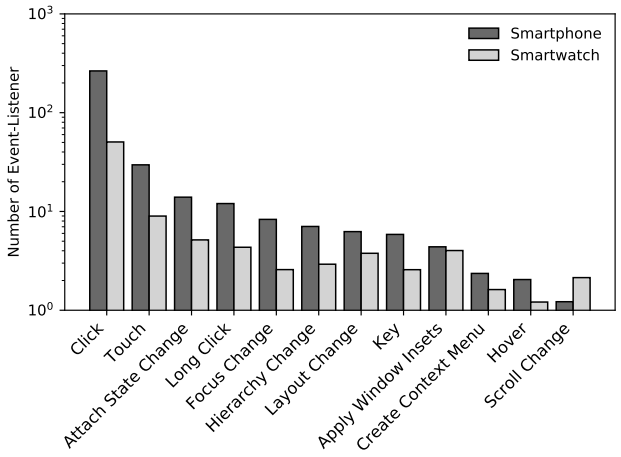


Figure 9: Average number of top 12 user-input event-listeners in each smartphone and smartwatch app (y-axis in log scale)

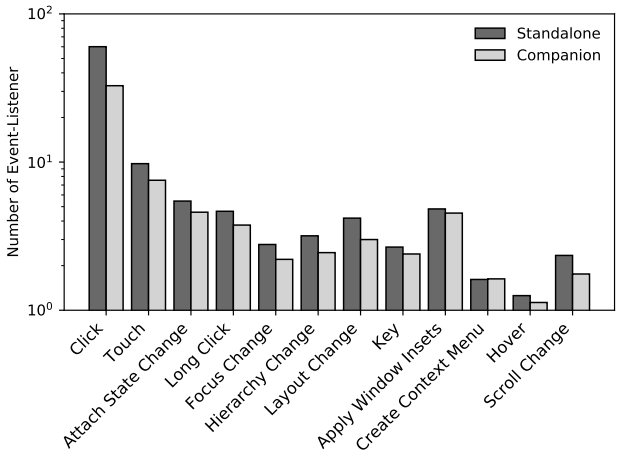


Figure 10: Average number of top 12 user-input event-listeners in each standalone and companion smartwatch app (y-axis in log scale)

reviewing all the *Personalization* app pairs, we find that most functions provided by both versions are identical. This phenomenon only appears in *Personalization* apps because their functions are relatively simple (i.e., showing different watch face), therefore, there is little difference in their smartphone and smartwatch versions.

Answer to RQ2

- Smartwatch apps are much less complicated than their smartphone counterparts in terms of User Interfaces and functionalities.
- Approximate 40% components and 30% methods are reused when developing cross-platform apps.
- Standalone watch apps reuse slightly more code (~6%) than companion apps.

4.3 RQ3: Code Similarity (User Interactions)

Android is an event-driven system, and most of the back-end methods are triggered by user interaction. To discover the differences in user interaction in smartphone and smartwatch apps, we analyze the callback event management approaches in the apps.

An event listener contains a callback method that will be invoked when users triggered specific widgets in the User Interface (UI). For example, the `OnClickListener` is triggered when the user clicked a widget in the UI (e.g., a button). Most event listeners are associated with and triggered by user interactions. By analyzing the event listeners in the apps, we are able to discover how the logic of users' interaction differs in the apps. To retrieve the input event listeners in an app, we decompile the bytecode to Java source code using JADX [3], and search for the presence of event listeners of user input provided in official Android documentation [2].

Figure 9 compares the top 12 user-input event-listeners in smartphone and smartwatch apps. As illustrated, click event is the mostly involved user-input method on both smartphones and smartwatches.

[3]https://github.com/skylot/jadx

Table 4: Detailed Simidroid results.

	Component			Method		
	Identical	Similar	New	Identical	Similar	New
Standalone	43.08%	1.03%	55.88%	33.44%	5.13%	61.43%
Companion	38.25%	2.44%	59.30%	26.23%	6.47%	67.31%

On average, each smartphone app has 265 `OnClickListener` registered, which is more than five times as many as the one registered in a smartwatch app (i.e., 51). Overall, phone apps have more user-input event-listeners registered than watch apps, suggesting that the number of user interactions on a smartwatch app are much less than that on a smartphone app. An exception is the `OnScrollChangeListener`, with on average one declared in smartphone apps and two in smartwatch apps. This is in line with the observation that smartwatch has much smaller screen size, therefore requires users to scroll more commonly to see all of the content on a page, for example, a long contact list.

Figure 10 presents the usage of the same event listeners in the standalone and companion smartwatch apps. It is interesting to find that the standalone apps always have more event listeners registered than companion apps. For example, the average number of `OnClickListener` on each standalone watch app is 60, while this number decreases to 33 on companion apps. The reason may be that some functions on companion apps are processed on the smartphone counterpart, therefore requires less interactions with users.

Answer to RQ3

- User interactions have been significantly reduced (i.e., five times less) in the smartwatch apps compared with their smartphone counterparts.
- Standalone watch apps involve more (i.e., two times more) user interaction than companion watch apps.

5 DISCUSSION

This section discusses implications of this study and promising research directions. We also enumerate some potential threats to validity in our findings.

Implication The findings of this study raise a number of issues and opportunities for the research and practice communities.

Possibility of automatic migration. As unveiled in our empirical results, approximately 25% to 35% of the code in the smartwatch apps are directly copied from their smartphone version, and 40% of the resources are reused in both versions. Given the gap in numbers between the smartphone apps and smartwatch apps in the market and the popularity of the smartwatch, there is a huge demand in creating smartwatch versions of the smartphone apps. Creating an automatic tool to migrate smartphone apps to smartwatch can benefit the app developers and contribute to the ecosystem of Android. Our empirical study can serve as a preliminary step in such an automatic migration tool.

Smartwatch apps are more than only the "simplified" version of smartphone apps. It has been commonly assumed that the smartwatch version of an app is usually a simplification of the smartphone version. Through analyzing the collected phone/watch app pairs, we found that smartwatch apps may request additional permissions (cf. RQ1) and add additional functions and new user interactions (cf. RQ2) to their smartphone counterparts. The characteristics of standalone and companion watch apps are also different, which can suit different app developers' requirements who want to create a smartwatch version of their app.

Threat To Validity Our study is conducted based on limited number of phone/watch app pairs, which may not be representative for the whole smartwatch ecosystem. Nevertheless, the majority of our dataset was collected from top 200 watch apps from the official Google Play Store, which demonstrate the landscape in popular apps. We plan to extend this work to analyze a more comprehensive dataset of watch apps in our future work. Furthermore, static analysis are known for its intrinsic vulnerability against code obfuscation, reflection, and dynamic loading [22]. Our analysis of code similarity (cf. RQ2) is based on static analysis, therefore, may be inaccurate it contains obfuscated code, java reflection, or dynamic loading. However, code obfuscation are more likely to be found in malware rather than benign apps [9]. Moreover, reflection and dynamic loading will not affect the similarity comparison of application code (e.g., if both apps calls `DexClassLoader()` in their code for dynamic loading, it is also an indication of similarity).

6 RELATED WORK

Previous research have performed extensively studies on large-scale analysis on the metadata and code of mobile apps. However, the characteristics of smartwatch apps have not yet been well investigated. To the best of our knowledge, our work is the first comparative analysis between an app's smartphone version and smartwatch version.

Metadata analysis. Large scale studies have been performed on analyzing the metadata of mobile apps [13, 15, 19]. Ali et al. [4] conducted a cross-platform study on apps' iOS version and Android version. They collected 80,169 pairs of iOS and Android apps and analyze the differences in user's ratings and reviews of the same app in Apple app store and Google Play store. Wang et al. [30] characterized more than 1 million mobile app developers and the apps they developed, across official Google Play store and 16 alternative app stores, and compared the developer's behaviors of developing, releasing, and maintaining the apps, as well as their misbehavior. Tian et al. [27] investigated the factors of making a high-rated apps compared with low-rated apps from Google Play store. They identified 17 key factors that have most impact on the app ratings, including the size of the app, the number of promotional images, and the target SDK version, etc. Carbunar et al. [8] observed 160,000 apps on Google Play store over a period of six months. They conducted temporal analysis on how the characteristics of these apps changed with the increased number of downloads.

Code analysis. Many research efforts have been focused on code level analysis of mobile apps [17, 22]. Syer et al. [26] presented a comparative study of 15 open-source Android apps with 5 open-source desktop applications on the code level. The issues have been investigated include the size of the code bases and the time taken to fix bugs in the projects. Similar works also studied the differences of bug fix on iOS and Android [5, 6]. Chia et al. [10] conducted a cross-platform study on usage of user-consent permissions in mobile apps and web apps. The scope of this study includes facebook apps, chrome extensions, and Android apps. Other works focused on analyzing the evolution of Android permission model and usage. Calciati et al. [7] studied over 14,000 releases of 227 Android apps, focusing on the change in permission usage in each update of the apps. A similar work [31] also include the study of Android permission system itself (from API level 3 to 15), and pre-installed apps from various vendors (HTC, Motorola, Samsung, and LG). The analysis of how third-party libraries are used in Android apps also attracts researcher's attention. Some studies have shown that third-party library are widely used in Android apps, and large portion of these libraries have many issues such as outdated and over-privileged [12, 23, 29]. Other code level analysis also include the code reuse in mobile apps and third-party libraries [14, 20, 21, 24, 25].

7 CONCLUSION

In this work, we have conducted an exploratory study of the relationship between the smartwatch and smartphone versions of Android apps. In particular, we have analyzed the collected app pairs from both non-code and code perspectives. Our experimental investigation finds that (1) Up to 40% of resource files (e.g., images, UI layouts, etc.) and 30% of code are reused between smartphone/watch app pairs, (2) Smartphone apps may require more than twice as many as permissions and adopt more than five times as many as user interactions than their watch counterparts, and (3) Smartwatch apps can be released as either standalone or companion versions depending on whether they can function independently. Standalone apps tend to request more permissions and reuse more code than companion apps.

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council (ARC) under a Discovery Early Career Researcher Award (DECRA) project

DE200100016, and a Discovery project DP200100020. This work was also partially supported by the National Natural Science Foundation of China (Grant No.61802180), the Natural Science Foundation of Jiangsu Province (Grant No.BK20180421), the National Cryptography Development Fund (Grant No.MMJJ20180105).

REFERENCES

[1] Comparative study of smartphone and smartwatch app pairs: Dataset. https://github.com/shell-coding/Phone-Watch-App-Pairs.

[2] Input events overview. https://developer.android.com/guide/topics/ui/ui-events, 2019. [Accessed May 26, 2020].

[3] Smartwatch shipments worldwide from 2018 to 2023. https://www.statista.com/statistics/878144/worldwide-smart-wristwear-shipments-forecast/, 2019. [Accessed May 26, 2020].

[4] Mohamed Ali, Mona Erfani Joorabchi, and Ali Mesbah. Same app, different app stores: A comparative study. In *MOBILESoft*, pages 79–90. IEEE, 2017.

[5] Wajdi Aljedaani, Meiyappan Nagappan, Bram Adams, and Michael Godfrey. A comparison of bugs across the ios and android platforms of two open source cross platform browser apps. In *MOBILESoft*, pages 76–86. IEEE, 2019.

[6] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. An empirical analysis of bug reports and bug fixing in open source android apps. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 133–143. IEEE, 2013.

[7] Paolo Calciati and Alessandra Gorla. How do apps evolve in their permission requests? a preliminary study. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 37–41. IEEE, 2017.

[8] Bogdan Carbunar and Rahul Potharaju. A longitudinal study of the google app market. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 242–249, 2015.

[9] Xiao Chen, Chaoran Li, Derui Wang, Sheng Wen, Jun Zhang, Surya Nepal, Yang Xiang, and Kui Ren. Android hiv: A study of repackaging malware for evading machine-learning detection. *IEEE Transactions on Information Forensics and Security*, 15:987–1001, 2019.

[10] Pern Hui Chia, Yusuke Yamamoto, and N Asokan. Is this app safe? a large scale study on application permissions and risk signals. In *Proceedings of the 21st international conference on World Wide Web*, pages 311–320, 2012.

[11] J. Clement. App stores: number of apps in leading app stores 2020, May 2020.

[12] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. Keep me updated: An empirical study of third-party library updatability on android. In *CCS*, pages 2187–2200, 2017.

[13] Jun Gao, Li Li, Pingfan Kong, Tegawendé F Bissyandé, and Jacques Klein. Should you consider adware as malware in your study? In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 2019)*, 2019.

[14] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. Juxtapp: A scalable system for detecting code reuse among android applications. In *DIMVA*, pages 62–81. Springer, 2012.

[15] Yangyu Hu, Haoyu Wang, Ren He, Li Li, Gareth Tyson, Ignacio Castro, Yao Guo, Lei Wu, and Guoai Xu. Mobile app squatting. In *Proceedings of The Web Conference 2020*, pages 1727–1738, 2020.

[16] Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24. IEEE, 2013.

[17] Pingfan Kong, Li Li, Jun Gao, Kui Liu, Tegawendé F Bissyandé, and Jacques Klein. Automated testing of android apps: A systematic literature review. *IEEE Transactions on Reliability*, 2018.

[18] Li Li, Tegawendé F Bissyandé, and Jacques Klein. Simidroid: Identifying and explaining similarities in android apps. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 136–143. IEEE, 2017.

[19] Li Li, Tegawendé F Bissyandé, and Jacques Klein. Moonlightbox: Mining android api histories for uncovering release-time inconsistencies. In *The 29th IEEE International Symposium on Software Reliability Engineering (ISSRE 2018)*, 2018.

[20] Li Li, Tegawendé F Bissyandé, and Jacques Klein. Rebooting research on detecting repackaged android apps: Literature review and benchmark. *IEEE Transactions on Software Engineering (TSE)*, 2019.

[21] Li Li, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. An investigation into the use of common libraries in android apps. In *The 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*, 2016.

[22] Li Li, Tegawendé F Bissyandé, Mike Papadakis, Siegfried Rasthofer, Alexandre Bartel, Damien Octeau, Jacques Klein, and Le Traon. Static analysis of android apps: A systematic literature review. *Information and Software Technology*, 88:67–95, 2017.

[23] Li Li, Daoyuan Li, Tegawendé F Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics & Security (TIFS)*, 2017.

[24] Israel J Mojica, Bram Adams, Meiyappan Nagappan, Steffen Dienst, Thorsten Berger, and Ahmed E Hassan. A large-scale empirical study on software reuse in mobile apps. *IEEE software*, 31(2):78–86, 2013.

[25] Israel J Mojica Ruiz, Meiyappan Nagappan, Bram Adams, and Ahmed E Hassan. Understanding reuse in the android market. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 113–122. IEEE, 2012.

[26] Mark D Syer, Meiyappan Nagappan, Ahmed E Hassan, and Bram Adams. Revisiting prior empirical findings for mobile apps: An empirical case study on the 15 most popular open-source android apps. In *CASCON*, pages 283–297, 2013.

[27] Yuan Tian, Meiyappan Nagappan, David Lo, and Ahmed E Hassan. What are the characteristics of high-rated apps? a case study on free android applications. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 301–310. IEEE, 2015.

[28] Raja Vallée-Rai, Phong Co, Etienne Gagnon, Laurie Hendren, Patrick Lam, and Vijay Sundaresan. Soot: A java bytecode optimization framework. In *CASCON First Decade High Impact Papers*, pages 214–224. 2010.

[29] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. Beyond google play: A large-scale comparative study of chinese android app markets. In *IMC*, pages 293–307, 2018.

[30] Haoyu Wang, Xupu Wang, and Yao Guo. Characterizing the global mobile app developers: a large-scale empirical study. In *MOBILESoft*, pages 150–161. IEEE, 2019.

[31] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. Permission evolution in the android ecosystem. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 31–40, 2012.