Same App, Different App Stores: A Comparative Study

Mohamed Ali   Mona Erfani Joorabchi   Ali Mesbah
University of British Columbia
Vancouver, BC, Canada
{mohamedha, merfani, amesbah}@ece.ubc.ca

ABSTRACT

To attract more users, implementing the same mobile app for different platforms has become a common industry practice. App stores provide a unique channel for users to share feedback on the acquired apps through ratings and textual reviews. However, each mobile platform has its own online store for distributing apps to users. To understand the characteristics of and differences in how users perceive the *same* app implemented for and distributed through *different* platforms, we present a large-scale comparative study of cross-platform apps. We mine the characteristics of 80,000 app-pairs (160K apps in total) from a corpus of 2.4 million apps collected from the Apple and Google Play app stores. We quantitatively compare their app-store attributes, such as stars, versions, and prices. We measure the aggregated user-perceived ratings and find many differences across the platforms. Further, we employ machine learning to classify 1.7 million textual user reviews obtained from 2,000 of the mined app-pairs. We analyze discrepancies and root causes of user complaints to understand cross-platform development challenges that impact cross-platform user-perceived ratings. We also follow up with the developers to understand the reasons behind identified differences.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.9 [**Software Engineering**]: Management – Software quality assurance (SQA)

Keywords

Mobile Apps, Mining App Stores, Android, iOS

1. INTRODUCTION

Online app stores are the primary media for the distribution of mobile apps. App stores also provide an important channel for app developers to collect user feedback, such as the overall user ratings or issues reported through user reviews.

To attract as many users as possible, developers often implement the same app for multiple mobile platforms [12]. While ideally, a given app should provide the same functionality and high-level behavior across different platforms, this is not always the case in practice [11]. For instance, a user of the Android Starbucks app complains: "*I downloaded the app so I could place a mobile order only to find out it's only available through the iPhone app.*" Or an iOS NFL app review reads: "*on the Galaxy you can watch the game live..., on this (iPad) the app crashes sometimes, you can't watch live games, and it is slow.*"

Recently, researchers have mined app stores by analyzing user-reviews [8, 25, 41], app descriptions [18, 33, 46], and app bytecode [6, 42, 43]. However, existing studies focus on one store at a time only. To the best of our knowledge, there is no cross-platform study that analyzes the *same* apps, published on *different* app stores.

Currently, iOS [4] and Android [3] dominate the app market [47], each with over 2 million apps in their respective app stores; hence, in this paper, we focus on these two platforms. To understand how the same app is experienced by the users on different platforms, we present a large-scale study on mobile **app-pairs**, i.e., *the same app implemented for iOS and Android platforms.* We employ a mixed-methods approach using both quantitative and qualitative analysis. We mine app-pairs and compare their various app-store attributes. We classify textual user reviews to identify discrepancies of user complaints across the platforms.

Our study helps to gain insight into the challenges faced by developers in cross-platform app development. It can help app developers to understand why the users of their apps might perceive and experience the same app differently across platforms, and to mitigate the differences. Android has gained the majority of the attention from the software engineering research community so far. One of the major obstacles with cross-platform analysis is the lack of a dataset for such apps [37]. Our mined artifact of more than 80,000 app-pairs is publicly available and can be used by researchers to go beyond Android and study different aspects of cross-platform apps.

Overall, our work makes the following main contributions:

- A dataset of 80,169 cross-platform app-pairs (iOS/Android), extracted by analyzing the properties of 2.4M apps from the Google Play and Apple app stores. Our app-pair dataset is publicly available [36].

© 2017 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

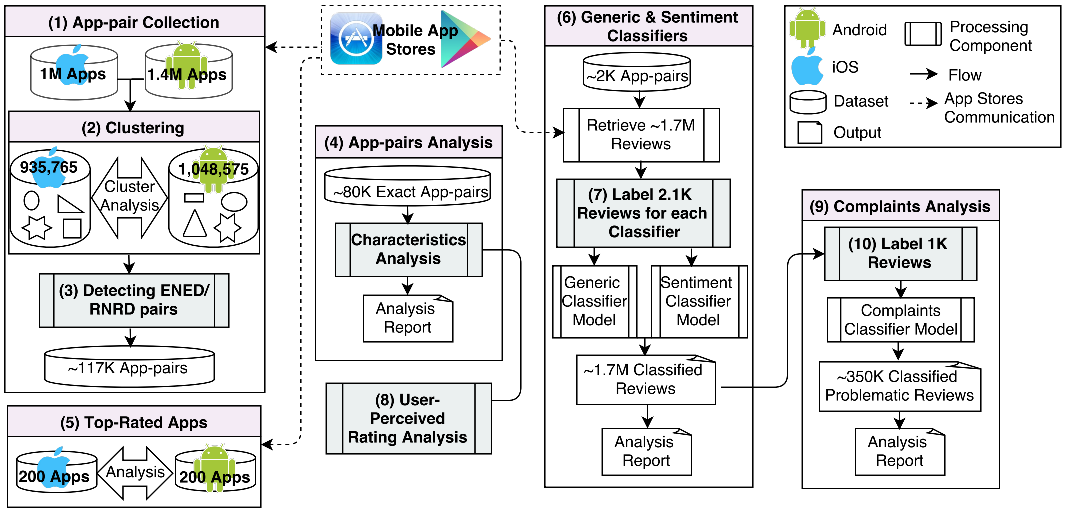


Figure 1: Overview of our methodology.

- A metric for measuring an app's aggregated user-perceived ratings, which combines ratings and stars.
- A characterization and comparison of app-pair attributes such as stars, ratings, prices, versions, and updates across platforms.
- Qualitative developer feedback, providing insights into the cause of variations in development, prices, and user-perceived ratings across platforms.
- Sentiment and complaints analysis of user reviews across app-pairs.

2. METHODOLOGY

Our analysis is based on a mixed-methods research approach [10], where we collect and analyze both quantitative and qualitative data. We address the following research questions in our study:

RQ1. How prevalent are app-pairs? Do app-pairs exhibit the same characteristics across app-stores?
RQ2. Why do some developers make their apps only available on one platform?
RQ3. Do users perceive app-pairs equally across platforms?
RQ4. Are the major user concerns or complaints the same across platforms?

Figure 1 depicts our overall approach. We use this figure to illustrate our methodology throughout this section.

2.1 Data Collection

To collect Android and iOS apps along with their attributes (Box 1 in Figure 1), we use two open-source crawlers, namely Google Play Store Crawler [16] and Apple Store Crawler [5] and mine apps from the two app stores, respectively. We only collect app attributes that are available on both stores. For instance, information about the number of downloads is only available for Android but not iOS, and thus, we ignore this attribute. Table 1 outlines the list of attributes we collect. This mining step results in 1.4 million Android apps and 1 million iOS apps. Data collection was conducted between Sep–Nov 2015 and the data was stored in a MongoDB database, which takes up approximately 2.1GB of storage [36].

Table 1: Collected app-pair attributes

#	Apple; Google	Description
1	name; title	Name of the app.
2	developerName; developer_name	Name of the developer/company of the app.
3	description; description	Indicates the description of the app.
4	category; category	Indicates the category of the app; 23 Apple & 27* Google categories.
5	isFree; free	True if the app is free.
6	price; price	Price ($) of the app excluding in-app purchases.
7	ratingsAllVersions; ratingsAllVersions	Number of users rating the app.
8	starsVersionAllVersions; star_rating	Average of all stars (1 to 5) given to the app.
9	version; version_string	User-visible version string/number.
10	updated; updated	Date the app was last updated.

*Google has its apps split into Games and Applications. We count Games as one category.

2.2 Matching Apps to Find App-Pairs

After creating the Android and iOS datasets separately, we set out to find app-pairs by matching similar apps in the two datasets. The unique IDs for iOS and Android apps are different and thus cannot be used to match apps, i.e., Android apps have an application ID composed of characters while iOS apps have a unique 8 digit number. However, app names are generally consistent across the platforms since they are often built by the same company/developer. Thus, we use app name and developer name to automatically search for app-pairs. This approach could result in multiple possible matches because (1) on one platform, developers may develop close variants of their apps with extra features that have *similar* names (See Figure 2); (2) the same app could have slightly different names across the platforms (See Figure 3–a); (3) the same app could have slightly different developer names across the platforms (See Figure 3–b).

Clustering per platform. To find app-pairs more accurately, we first cluster the apps on each platform. This step (outlined in Box 2 of Figure 1) groups together apps on each platform that belong to the same category, have *similar* app names (i.e., having the exact root word, but allowing permutations) and the same developer name. Figure 2 is an example of a detected Android cluster. The apps in this cluster are all developed by *iGold Technologies*, belong to the Game category and have *similar* (but not exact) names.

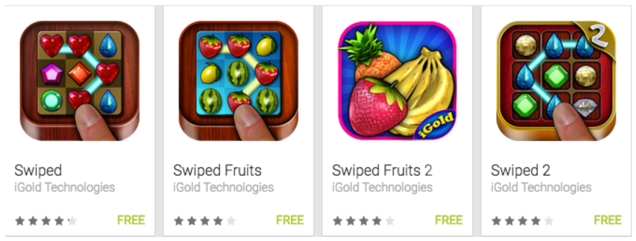
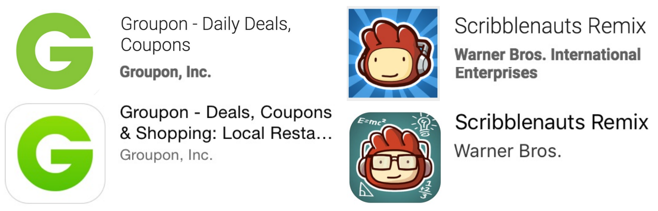


Figure 2: Android Cluster for Swiped app.

We execute a clustering algorithm on the Android and iOS datasets, separately. The algorithm takes as input a collection of apps and annotates the collection to group the apps together.For each app, we extract the app name, developer name, and category. Next, if an app has not been annotated previously, we annotate it with a unique *clusterID*. Then we search for apps in the collection that have a similar name, exact developer name, and belong to the same category. If a match is found, we annotate the found app with the same *clusterID*.

Detecting App-Pairs. We consider an app-pair to consist of the iOS version and the Android version of the same app. In our attempt to find app-pairs (Box 3 in Figure 1), we noticed that Android and iOS apps have different naming conventions for app names and developer names. For instance, Figure 3–a depicts an app developed by *'Groupon, Inc.'*, with different naming conventions for app names; *'Groupon - Daily Deals, Coupons'* on the Android platform whereas *'Groupon - Deals, Coupons & Shopping: Local Restaurants, Hotels, Beauty & Spa'* on the iOS platform. Similarly, Figure 3–b shows the *'Scribblenauts Remix'* app, which has the exact name on both platforms, but has differences in the developer's name.

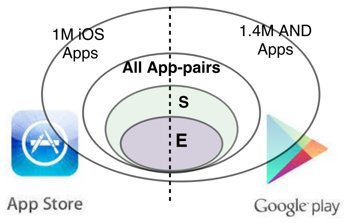

Groupon - Daily Deals, Coupons
Groupon, Inc.
Groupon - Deals, Coupons & Shopping: Local Resta...
Groupon, Inc.
Scribblenauts Remix
Warner Bros. International Enterprises
Scribblenauts Remix
Warner Bros.

(a) (b)

Figure 3: a) Groupon and b) Scribblenauts apps. Android apps are shown on the top and iOS apps at the bottom.

Figure 4 shows the app-pairs we find using matching criteria with different constraints. Criteria E looks for app-pairs having *exact app and developer name* whereas Criteria S relaxes both the app and developer name, thus matching the apps in Figure 3 as app-pairs.

To find app-pairs, we match the Android clusters with their iOS counterparts. First, we narrow down the search for a matching cluster by only retrieving those with a similar developer name. This results in one or more possible matching clusters and we identify the best match by comparing the items in each cluster. Thus, for each app in the Android cluster, we look for an exact match (criteria E) in the iOS cluster. If no match is found, we relax the criteria



ID	App-pair Criteria
E	EXACT(AppName) & EXACT(DevName)
S	SIMILAR(AppName) & SIMILAR(DevName)

Figure 4: Matching App-pair Criteria.

and look for matches having a similar app and developer name (criteria S). The set of all possible app-pairs is a superset of S, and S is a superset of E, as depicted in the Venn diagram of Figure 4.

Exact App-Pairs. We perform the rest of our study using criteria E, which provides a large-enough set of exactly matched app-pairs needed for our analysis. To validate whether criteria E correctly matches app-pairs, the first two authors manually compared app names, descriptions, developers' names, app icons and screenshots of 100 randomly selected app-pairs and the results indicated that there are no false positives. This is, however, no surprise given the strict criteria defined in E.

2.3 App-store Attribute Analysis

To address RQ1, (Box 4 in Figure 1) we compare the captured attributes between the iOS and Android app-pairs and present the results in section 3.

To address RQ2, we use the iTunes Store RSS Feed Generator [29] to retrieve the top rated apps, which enables us to create custom RSS feeds by specifying feed length, genres, country, and types of the apps to be retrieved. These feeds reflect the latest data in the Apple app store. The Google Play store provides the list of top rated Android apps [27] as well. We collected the top 100 free and 100 paid iOS apps belonging to all genres, as well as top 100 free and 100 paid Android apps belonging to all categories (Box 5 in Figure 1). To check whether a top app exists on both platforms, we apply our exact app-pair technique as described in the previous section. Since the lists were not long, we also manually validated the pairs using the app name, developer name, description and screenshots.

2.4 User Reviews

In addition to collecting app-store attributes for our app-pairs in RQ1, we analyze user reviews of app-pairs to see if there are any discrepancies in the way users experience the same app on two different platforms (RQ4).

To that end, we first select 2,000 app-pairs that have more than 500 ratings, from our app-pair dataset. This allows us to target the most popular apps with enough user reviews to conduct a thorough analysis. To retrieve the user reviews, we use two open-source scrapers, namely the iTunes App Store Review Scraper [28] and the Goole Play Store Review Scraper [17]. In total, we retrieve 1.7 million user reviews from the 2K app-pairs.

The goal is to semi-automatically classify the user reviews of the app-pairs and compare them at the app and platform level. To achieve this, we use natural language processing

Table 2: Real-world reviews and their classifications.

	C1 – Generic Feedback Classifier
1	**Problem Discovery**: "*Videos don't work. The sound is working but the video is just a black screen.*"
2	**Feature Request**: "*I would give it a 5 if there were a way to exclude chain restaurants from dining options.*"
3	**Non-informative**: "*A far cry from Photoshop on the desktop, but still a handy photo editor for mobile devices with...*"
	C2 – Sentiment Classifier
1	**Positive**: "*Amazing and works exactly how I want it to work. Nothing bad about this awesome and amazing app!*"
2	**Negative**: "*The worst, I downloaded it with quite a lot of excitement but ended up very disappointed*"
3	**Neutral**: "*No complaints because I'm not a complainer save your option for something that matters*"

and machine learning to train two classifiers (Box 6 in Figure 1). Each classifier can automatically put a review into one of its three classes.

Generic Feedback Analysis. As shown in Table 2, our generic feedback classifier (C1) has three unique class labels {*Problem Discovery, Feature Request, Non-informative*}; where *Problem Discovery* implies that the user review pertains to a functional (bug), or non-functional (e.g., performance), or an unexpected issue with the app. *Feature Request* indicates that the review contains suggestions, improvements, requests to add/modify/bring back/remove features. Finally, *Non-informative* means that the review is not a constructive or useful feedback; such reviews typically contain user emotional expressions (e.g., 'I love this app', descriptions (e.g., features, actions) or general comments. We have adopted these classes from recent studies [8, 41] and slightly adapted them to fit our analysis of user complaints and feedback across the two platforms.

Sentiment Analysis. Additionally, we are interested in comparing the sentiment (C2 in Table 2) classes of {*Positive, Negative, Neutral*} between the reviews of app-pairs. We use these rules to assign class labels to review instances. Table 2 provides real review examples of the classes in our classifiers.

Labelling Reviews. Since labelling is a tedious and time-consuming task, we constrain the number of app-pairs and reviews to manually label. We randomly selected 1,050 Android user reviews and 1,050 iOS user reviews from 14 app-pairs. These app-pairs were in the list of the most popular apps and categories in their app stores. The manual labeling of reviews was first conducted by one author following the classification rules inferred in Table 2. Subsequently, any uncertainties were cross-validated and resolved through discussions and refinements between the authors. Overall, we label 2,100 reviews for training each of the two classifiers (Box 7 in Figure 1).

Building Classifiers. To build our classifiers, we first preprocess the text, tokenize it and filter stop words. We use the *bags of words* representation, which counts the number of occurrences of each word to turn the textual content into numerical feature vectors. We use the feature vectors to train our classifier and apply a machine learning algorithm on the historical training data. In this work, we experimented with two well-known and representative semi-supervised algorithms, Naive Bayes (NB) and Support Vector Machines (SVM). We use the Scikit Learn Tool [45] to build our classifiers. The training and testing data for our classifiers were randomly composed of 1,575 and 525 of the manually labelled reviews, respectively. We repeated this trial 25 times to train both our generic and sentiment classifiers and compared the NB and SVM algorithms. We choose the generic (C1) and sentiment (C2) classifiers with the best F-measures.

We use the trained classifiers to classify ~1.7 million reviews of the 2K app-pairs.

2.5 User-Perceived Rating

There are multiple ways to measure how end-users perceive an app. For example the number of downloads can be an indication of the popularity of an app. However, as discussed by Tian et al. [49], many users download an app without ever using it. More importantly, as explained in subsection 2.1, Apple does not publish the download count for iOS apps, which means we cannot use this metric in our cross-platform study.

Another method is to measure the sentiment of user reviews through NLP techniques. Such techniques, however, lack the required accuracy for measuring succes [21, 48].

The star rating of an app, which is the average rating of an app (between 1–5), has been used in many studies to measure an app's success rate [7, 19, 23, 49]. However, relying only on the average star rating of an app might be misleading since it does not take into account the number of ratings the app receives. For instance the *Facebook* app on the Google Play store currently has an average star rating of 4 with over 40 million ratings. On the other hand, *OneRepMaxCalculator* currently has an average star rating of 5, but only seven ratings. Despite having a lower star rating, logically the *Facebook* app is better perceived because it has much more ratings. To mitigate this issue, we combine the average star rating with the number of ratings to measure the Aggregated User-perceived Rating (AUR) (Box 8 in Figure 1) of an app as follows:

$$\text{AUR}(app_i) = \frac{v_i \times r_i}{v_i + m} + \frac{m \times c}{v_i + m} \tag{1}$$

where

1. v_i is the number of ratings for app_i,
2. r_i is the average stars for the app_i,
3. m is the average number of ratings (for all apps in the dataset),
4. c is the average number of stars (for all apps in the dataset).

If an app does not have enough ratings (i.e., less than m) we cannot place much trust on the few ratings to accurately measure aggregate rating, and thus the formula penalizes it by bringing in the average values of m ratings and c stars.

We were inspired by the movies ranking algorithm [26] of the Internet Movie Database (IMDB), which uses user votes to generate the top 250 movies. The formula is believed to provide a true Bayesian estimate [13, 26].

AUR provides a number between [1–5], which we convert into a percentage to better represent the results. In practice, some apps have no ratings and no stars. In our work, we require that an app must have at least one rating to be included in the analysis.

To illustrate the need for combining ratings and stars, and evaluate our proposed AUR metric, we randomly selected 100 apps from our dataset and ranked them based on different metrics. The average ratings (m) and stars (c)

Table 3: Ranking apps using different metrics.

App	Ratings (R)	Stars (S)	Rank S	Rank R	Rank AUR
A	1	5.0	1	6	4
B	4	4.8	2	5	3
C	1825	4.7	3	2	1
D	11	2.1	5	4	5
E	67	4.6	4	3	2
F	2796	1.8	6	1	6

across the 100 apps were 142 and 4.1, respectively. Table 3 presents the rankings for six of the apps based on the stars, the ratings, and AUR. Using only the stars ranks app A first although it only has one single rating. Using only the ratings would rank app F first although it has only 1.8 stars. Our proposed metric, AUR, ranks C first, because it has many ratings (1825) and relatively high stars (4.7). It ranks F last, which has many ratings but the lowest stars.

2.6 Cross-platform Complaint Analysis

The goal in RQ4 is to understand the nature of user complaints and how they differ on the two platforms (Box 9 in Figure 1). To address this, we first collect the *Problem Discovery* reviews for 20 app-pairs having (1) the biggest differences in AUR rates between the platforms, and (2) over 100 problematic reviews. These 20 app-pairs are split into 10 in which Android has a higher AUR than iOS and 10 in which iOS has a higher AUR than Android. Then, we manually inspect and label 1K problematic reviews (Box 10 in Figure 1), by randomly selecting 25 Android user reviews and 25 iOS user reviews from each of the 20 app-pairs. We noticed that user complaints usually fall into the following five subcategories: (1) *Critical*: issues related to crashes and freezes; (2) *Post Update*: problems occurring after an update/upgrade; (3) *Price Complaints*: issues related to app prices; (4) *App Features*: issues related to functionality of a feature, or its compatibility, usability, security, or performance; (5) *Other*: irrelevant comments.

We use the labelled dataset to build a complaints classifier to automatically classify ~350K problematic reviews of our 2K app-pairs.

2.7 Datasets and Classifiers

All our extracted data, datasets for the identified app-pairs and the 2K app-pairs along with their extracted user reviews, as well as all our scripts and classifiers are publicly available [36].

3. FINDINGS

In this section, we present the results of our study for each research question.

3.1 Prevalence and Attributes (RQ1)

We found 1,048,575 (~1M) Android clusters for 1,402,894 (~1.4M) Android apps and 935,765 (~0.9M) iOS clusters for 980,588 (~1M) iOS apps in our dataset. The largest Android cluster contains 219 apps[1] and the largest iOS cluster contains 65 apps.[2] Additionally, 7,845 Android and 9,016 iOS clusters have more than one item. The first row of Table 4 shows descriptive statistics along with p-value (Mann-Whitney) for cluster sizes, ignoring clusters of size 1. Figure 5 depicts the cluster sizes for the two platforms.

[1] https://play.google.com/store/search?q=Kira-Kira&c=apps&hl=en

[2] https://itunes.apple.com/us/developer/urban-fox-production-llc/id395696788

Table 4: iOS & Andriod (AND) descriptive statistics: Cluster Size (C), Ratings (R), Stars (S), and Price (P).

ID	Type	Min	Mean	Median	SD	Max	p
C	iOS	2	3.30	3.00	2.11	65	0
	AND	2	3.00	2.00	3.69	219	
R	iOS	5	1,935.00	**21.00**	26,827.24	1,710,251	0
	AND	1	4,892.00	**11.00**	171,362.40	28,056,146	
R*	iOS	0	353.10	0.00	11,483.19	1,710,251	0
	AND	0	3,302.00	3.00	140,807.60	28,056,146	
S	iOS	1	3.80	**4.00**	0.90	5	0
	AND	1	4.04	**4.10**	0.82	5	
S*	iOS	0	0.70	0.00	1.52	5	0
	AND	0	2.73	3.70	2.01	5	
P	iOS	0.99	3.58	1.99	9.73	500	0
	AND	0.95	4.00	2.11	9.81	210	

*Including apps that have no ratings/stars/prices (i.e., all apps).

Prevalence of app-pairs. We found 80,169 (~80K) exact app-pairs (Criteria E in Figure 4), which is *8% of the total iOS apps, and 5.7% of the total Android apps in our datasets*. When we relax both app and developer names, the number of app-pairs increases to 116,326 (~117K) app-pairs, which is *13% of our iOS collection and 9.2% of our Android collection.*

Finding 1*: Our results indicate that a large portion of apps (87–95%) are developed for one particular platform only.*

Ratings & Stars. Interestingly, 68% of Android and only 18% of iOS apps have ratings. The Median is 0 for all iOS and 3 for all Android, as depicted in Table 4. However, when we only consider apps with at least one rating, the median increases to 21 for iOS and 11 for Android (See Figure 6). We ignore outliers for legibility. Furthermore, we compare the differences between ratings for each pair. In 63% of the pairs, Android apps have more users rating them (on average 4,821 more users) whereas in only 5% of the pairs, iOS apps have more users rating them (on average 1,966 more users).

Similarly, 68% of Android and 18% of iOS apps have stars. When we consider the apps with stars, the median increases to 4 for iOS and 4.1 for Android (See Figure 7). Comparing the differences between the stars for each pair, in 58% of the pairs, Android apps have more stars while in only 8% of the pairs, iOS apps have more stars.

Finding 2*: Android users tend to rate apps more than iOS users.*

Prices of app-pairs. Ideally, the same app should have the same price on different platforms. The general belief is that developers price their iOS apps higher than Android apps. We explored to what extend this is true.

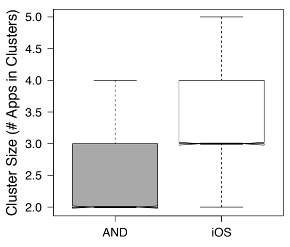


Figure 5: Clusters

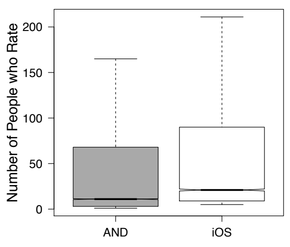


Figure 6: Ratings

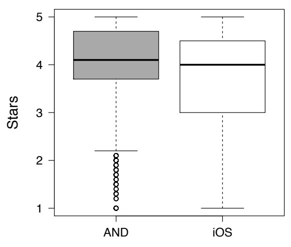


Figure 7: Stars

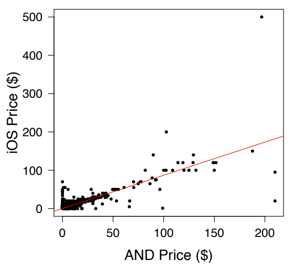


Figure 8: Prices

Table 5: Statistics of 14 Apps used to build the classifiers (C1 = Generic Classifier, C2 = Sentiment Classifier, NB = Naive Bayes Algorithm, SVM = Support Vector Machines Algorithm)

#	App	GoogleCategory	AppleCategory	F(C1-NB)	F(C2- NB)	F(C1-SVM)	F(C2-SVM)
1	FruitNinja	Game(Arcade)	Game	0.77	0.68	0.83	0.75
2	UPSMobile	Business	Business	0.80	0.69	0.82	0.76
3	Starbucks	Lifestyle	Food & Drink	0.75	0.63	0.84	0.77
4	YellowPages	Travel & Local	Travel	0.78	0.62	0.85	0.75
5	Vine	Social	Photo & Video	0.81	0.70	0.84	0.76
6	Twitter	Social	Social Networking	0.79	0.67	0.84	0.75
7	AdobePhotoShop	Photography	Photo & Video	0.82	0.72	0.85	0.75
...	...			...	...	...	...
	Total / Average of 14 Apps			0.77	0.65	**0.84**	**0.74**

Our results show that *88% of app-pairs have different prices for their Android versus iOS versions.* Comparing the rate of free and paid apps, 10% of the Android and 12% of iOS apps are paid. In 34% of the pairs, iOS apps have a higher price whereas in 54% of the pairs, Android apps have a higher price. As Table 4 shows, the mean and median for paid apps are slightly higher for Android compared to iOS.

Finding 3: *Our results indicate that while more Android apps are free, the paid Android apps have slightly higher prices than their iOS counterparts.*

For some of the app-pairs, the price differences is huge, as depicted in Figure 8. To understand the reasons behind these differences, we sent emails to all the developers of app-pairs with price differences of more than $10 (US) and asked why their app-pairs were priced differently on the two platforms. Out of 52 emails sent, we received 25 responses and categorized the main reasons:

Different monetization strategies per app store. For instance, "*the difference is that the Android version includes consolidation ($9.99), charting ($14.99), reports ($9.99) and rosters ($14.99), whereas these are 'in app purchase' options on Apple devices.*"

Different set of features on the two platforms: "*the iOS version offers more features than the Android version.*"

Development/maintenance costs of the app: one respondent said "*the effort to maintain an App on iOS is much higher than on Android*", while another stated "*Android is relatively expensive and painful to create for and much harder to maintain and support.*" It is interesting to see that developers have different, even conflicting, perspectives of the difficulties involved in the development and maintenance of apps for each platform.

Exchange rate differences e.g., "*price in both are set to 99 EUR as we are mainly selling this in Europe. Play Store apparently still used USD converted by the exchange rate of the day the app was published.*"

We have to note that some of the developers we contacted were unaware of the price differences.

Versions and last updated. While the app stores' guidelines suggest that developers follow typical software versioning conventions such as *semantic versioning*[3] — in the form of (major.minor.patch) — they do not enforce any scheme. Therefore, mobile apps exhibit a wide variety of versioning formats containing letters and numbers, e.g., date-based schemes (year.month.patch). Our data indicate that only 25% of the app-pairs have identical versions. When we inspect the major digit only, 78% of the pairs have the same version. 13% of the Android apps have a higher version compared to 9% of the iOS apps that have a higher version.

Comparing the date the apps were last updated, 58% of the app-pairs have an iOS update date which is more recent than Android; while 42% have a more recent Android update date.

Finding 4: *Our results indicate that the majority of cross-platform apps are not consistently released. Only one in every four app-pairs has identical versions across platforms and 30% of the app-pairs have update dates which are more than 6 months apart.*

3.2 Top Rated Apps (RQ2)

Interestingly, our analysis on the top 100 *free* iOS and Android apps shows that 88% of the top iOS and 86% of the top Android apps have pairs. 37 app-pairs are in the

[3] http://semver.org

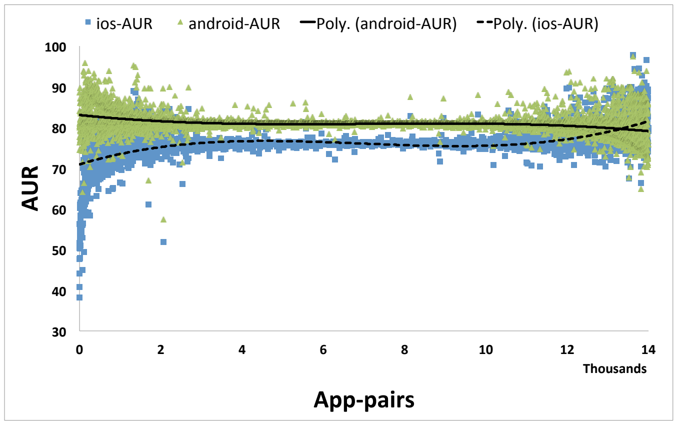


Figure 9: AUR scores calculated per platform.

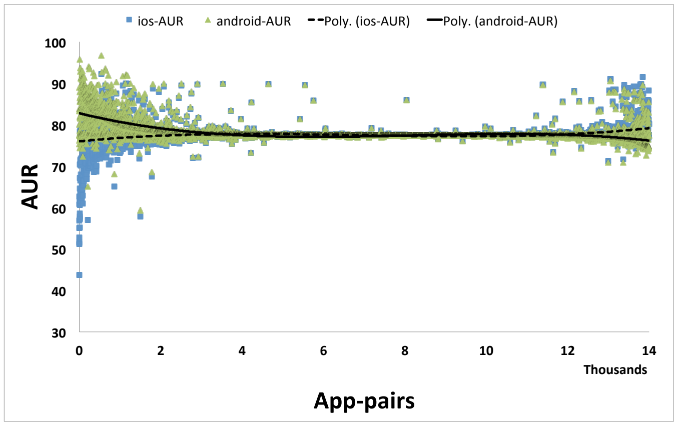


Figure 10: AUR scores calculated across the platforms.

top 100 list for both platforms. On the other hand, for the top 100 *paid* iOS and Android apps, 66% of the top iOS and 79% of the top Android apps have pairs. 30 of the paid pairs are in the top 100 for both platforms.

Finding 5*: Over 80% of the top-rated apps are available on both platforms.*

To understand why some developers of successful apps only develop for one platform, we sent emails to all the developers of apps with no pairs. Out of 81 emails sent, we received 29 responses and categorized the main reasons below:

Lack of resources: "*building the same app across two platforms is actually twice the work given we can't share code ... so we'd rather make a really good app for one platform than make a mediocre one for two.*"

Platform restrictions: "*I've only focused on the Android platform simply because Apple doesn't allow for much customization to their UI.*"

Revenue per platform: "*In my experience, iOS users spend more money, which means a premium [paid app with a price higher than 2.99] is more likely to succeed. ... while the Android platform has the market size, it proves to be harder for small [companies] to make good money.*"

Fragmentation within a platform: "*my app is very CPU intensive and thus, I must test it on every model. With a much-limited number of models for iOS, it's feasible. On Android, it's impossible to test on every model and quality would thus suffer.*"

Similar apps already exist on the other platform: "*Apple devices already have a default podcast app.*"

A common response from developers was that the app for the other platform is under development.

3.3 Aggregated User Perceived Ratings (RQ3)

Figure 9 shows the AUR rates for our app-pairs, computed using formula 1. Pairs of triangular and square points represent an app-pair. We only keep app-pairs that contained at least 1 Android and 1 iOS rating; this reduced the number of app-pairs to 14,000. The average number of ratings (m) across the Android apps was 18,199 and the average number of stars (c) was 3.9. For iOS, m was 1,979 ratings and c was 3.8 stars. The app-pairs are sorted based on the difference in their AUR rates on the two platforms. The far ends of the figure indicate apps that are rated higher on one of the two platforms.

The results indicate that in 95% of the app-pairs, the Android version is perceived better by users. Figure 10 shows the AUR rates for the app-pairs; but now with m and c set as the averages across all the Android and iOS apps combined. The averages for the ratings and stars were 10,089 and 3.8 respectively. Using these values for m and c results in 59% of the Android apps being perceived better compared with their iOS counterparts.

Finding 6*: The Android version of app-pairs receives higher user-perceived ratings compared to the iOS version.*

The method used to implement an app-pair might affect how its perceived by end-users. To explore this, we randomly selected and downloaded 30 app-pairs with similar AUR scores (within 5% range). We found that eight of them were implemented using a *hybrid* approach. The hybrid approach uses web technologies such as `HTML`, `CSS`, and `Javascript` to build mobile apps that can run across platforms. We also analyzed 30 app-pairs that had a higher AUR on iOS than Android and 30 app-pairs with higher AUR on Android (i.e., with differences greater than 20%). We found only four in each set used the hybrid approach. In total, we found 16 hybrid apps, which represents 17.7% of 90 app-pairs we inspected. This result is in line with previous studies [50], which found that 15% of Android apps are developed using a hybrid approach. Our analysis indicates that hybrid apps are usually equally rated by users on the platforms, which is not surprising as they have the same implementation on the two platforms.

Furthermore, to understand why an app-pair is perceived differently on each platform, we sent emails to all the developers of app-pairs which had a difference of more than 30% in their AUR scores. We asked if they have noticed the difference and possible reasons that their two apps are not equally rated across platforms. Out of 200 sent emails, we received 20 responses. All the respondents agreed with our findings and were aware of the differences; for example, one developer said: "*our app was by far more successful on iOS than on Android (about a million downloads on iOS and 5k on Android).*" The reasons given were as follows. Timing (release/update) and first impressions were thought to

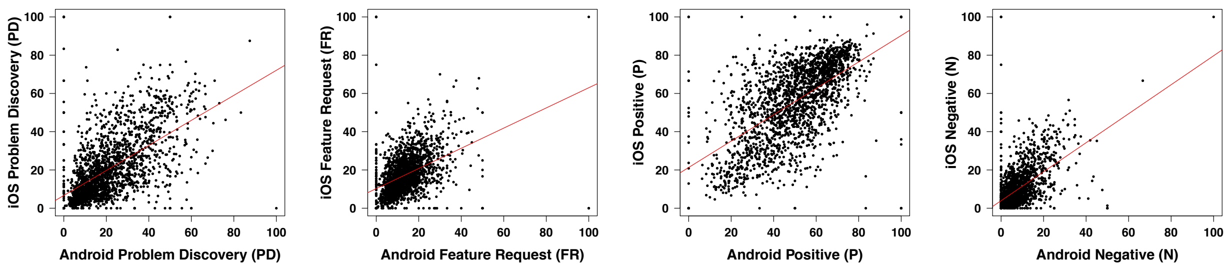


Figure 11: The rates of classifiers' categories for the 2K app-pairs, where each dot represents an app-pair.

Table 6: Descriptive statistics for iOS & AND reviews: Problem Discovery (PD), Feature Request (FR), Non-informative (NI), Positive (P), Negative (N), Neutral (NL), and AUR.

ID	Type	Min	Mean	Median	SD	Max	p
PD	iOS	0.00	20.47	**15.62**	16.65	100.0	0.00
	AND	0.00	21.06	**17.54**	14.61	100.0	
FR	iOS	0.00	17.50	**16.03**	10.81	100.0	0.00
	AND	0.00	13.71	**12.50**	8.88	100.0	
NI	iOS	0.00	62.04	64.95	20.77	100.0	0.00
	AND	0.00	65.23	67.10	17.45	100.0	
P	iOS	0.00	55.62	59.26	20.41	100.0	0.00
	AND	0.00	49.74	51.36	17.64	100.0	
N	iOS	0.00	9.80	6.66	10.07	100.0	0.00
	AND	0.00	7.72	5.74	7.39	100.0	
NL	iOS	0.00	34.57	32.45	14.87	100.0	0.00
	AND	0.00	42.54	41.73	13.97	100.0	
AUR	iOS	38.22	76.21	76.03	3.06	99.75	0.27
	AND	52.53	80.88	80.90	2.07	97.48	

make a big difference in how users perceive an app. The variation in ratings across platforms can also be attributed to the degree at which developers provide support on either side. Additionally, app store support and promotional opportunities were mentioned to help developers, e.g., "*Apple … promote your work if they find it of good quality, this happened to us 4–5 times and this makes a big difference indeed*". Furthermore, some respondents find the Google Play's quick review process helpful to release bug fixes and updates quickly.

3.4 Complaints Across Platforms (RQ4)

Classification. To evaluate the accuracy of the classifiers, we measured the F-measure for the Naive Bayes and SVM algorithms, listed in Table 5. We found that SVM achieves a higher F-measure. On average, F(SVM) = 0.84 for the generic classifier and F(SVM) = 0.74 for the sentiment classifier. The F-measures obtained by our classifiers are similar to previous studies such as Panichella et al. [41] (0.72) and Chen et al. [8] (0.79). We selected the classifiers with the best F-measures and used them to classify 1,702,100 (~1.7 million) reviews for 2,003 (~2K) app-pairs.

Sentiment and Generic Reviews. Figure 11 plots the distribution of the rates for the main categories in the sentiment and generic classifiers for our app-pairs. Note that each dot represents an *app-pair*. The descriptive statistics are shown in Table 6. On average, *Feature Request*, *Positive*, and *Negative* reviews are more among the iOS versions whereas *Problem Discovery*, *Non-informative* and *Neutral* are more among Android versions of app-pairs. Further, we found that the average length of reviews on the iOS platform is larger, namely 103 characters versus 76 characters on the Android platform.

The goal in RQ4 was to understand the nature of user complaints and whether they differ on the two platforms.

Table 7: Descriptive statistics for problematic reviews: App Feature (AF), Critical (CR), Post Update (PU), and Price Complaints (PC).

ID	Type	Min	Mean	Median	SD	Max	P-Val
AF	iOS	0.00	**53.71**	54.29	18.15	100.0	0.00
	AND	0.00	**60.55**	60.92	16.25	100.0	
CR	iOS	0.00	**23.72**	21.05	16.40	100.0	0.00
	AND	0.00	**19.98**	17.65	13.66	100.0	
PU	iOS	0.00	**6.08**	4.24	7.44	100.0	0.00
	AND	0.00	**3.91**	2.33	5.17	50.0	
PC	iOS	0.00	7.76	5.00	9.41	100.0	0.00
	AND	0.00	6.70	4.54	8.20	100.0	

Complaints. Our complaint classifier has, on average, an F-measure of F(SVM) = 0.7. We used the classifier to classify 350,324 (~350K) problematic reviews for our 2K app-pairs.

The results, depicted in Figure 12 and Table 7, show that the complaints about the apps vary between the two platforms. On average, iOS apps have more critical and post update problems than their Android counterparts, which could be due to Apple regularly forcing developers to migrate their apps to their latest OS and SDK. Examples of iOS post update complaints include users unable to login, features no longer working, loss of information or data, and unresponsive or slow UI.

On the other hand, Android apps have more complaints related to features, which could be due to device fragmentation on Android. The wide array of Android devices running with different versions of Android, different screen sizes, and different CPUs can cause non-functional complaints related to security, performance or usability problems. This negative side-effects of fragmentation is discussed in other studies [12, 15, 22, 52]. Examples of Android complaints include dissatisfaction with a certain functionality, incompatibility with a certain device/OS, and network and connectivity problems.

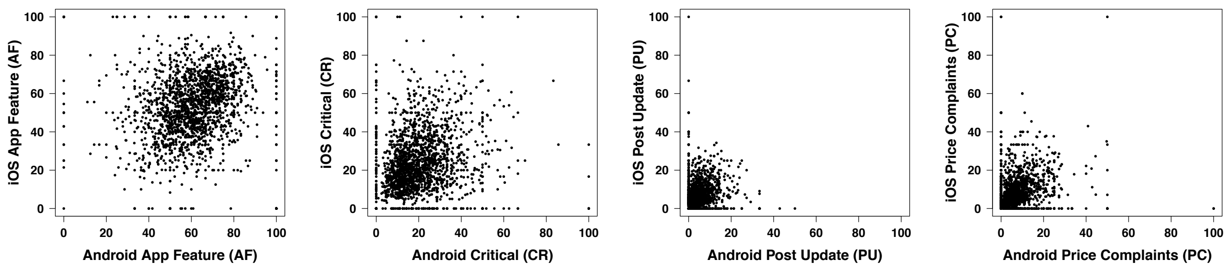


Figure 12: The rates of complaint categories for the 2K app-pairs; each dot represents an app-pair.

Finding 7*: The results indicate that on average, iOS apps receive more critical and post update complaints while Android apps receive more complaints related to app features and non-functional properties.*

4. DISCUSSION

In this section, we discuss several implications of our study and the threats to validity of our results.

Implications. Our study helps to gain insights into the challenges faced by developers such as inconsistencies that might arise due to different strategies for maintaining, releasing, and pricing apps across platforms. It can help app developers to understand why users of their apps might experience, complain, or rate the same app differently across platforms, and to mitigate the differences.

Our results indicate that a large portion of apps (87–95%) are developed for one platform only. While both platforms are popular and equally important, Android has gained the majority of the attention from the software engineering research community by far. Our results suggest that apps from both Apple Store and Google Play need to be included in future studies to have a more representative coverage.

More than 80% of the top-rated apps exist on both the Apple and Google Play app stores. As recently identified by Nagappan and Shihad [37], one of the obstacles with cross-platform analysis is the lack of a dataset for such apps. Our work provides a large dataset with more than 80,000 exact app-pairs of iOS and Android apps [36]. This large dataset, which is now publicly available, can be leveraged by other researchers for further cross-platform analysis of mobile apps.

Our results show that end-users can perceive and rate cross-platforms differently on each platform. This is especially true for native apps that are built with different languages and technologies. Hybrid apps are less susceptible to such user-perceived variations across platforms. We recently conducted an empirical study on hybrid apps [2] and discovered that hybrid apps are well perceived by users across platforms and can outperform native apps in terms of aggregated ratings; out of the 25 possible app store categories, the hybrid apps had better ratings in 18. Developers willing to achieve more consistency for their apps across platforms can benefit from creating hybrid apps.

Review Classification. There are many techniques available to classify textual user reviews. The goal of this work was not to develop a new classification technique to outperform other techniques, but to simply compare the nature of reviews for the same apps, on the two platforms. To this end, we surveyed the literature and chose the technique best suited to our need while achieving a decent F-score. The Support Vector Machine (SVM) algorithm along with the NLP features of the Scikit Learn Tool were the best choice for our study and resulted in F-scores that were comparable to other similar studies [8, 41].

Threats to Validity. Our manual labelling of the reviews to train the classifiers could be a source of internal threat to validity. In order to mitigate this threat, uncertainties were cross-validated and resolved through discussions and refinements between the authors. As shown in Figure 4, the app-pairs detected in our study are a subset of all possible app-pairs. Our study only considers exact matches for app-pairs, which means there exist app-pairs that are not included in our analysis. For instance, an app named *The Wonder Weeks*[4] on iOS has a pair on the Android platform with the name *Baby Wonder Weeks Milestones*,[5] but not included in our study. While our study has false negatives, our manual validation of 100 randomly selected app-pairs shows that there are no false positives. In terms of representativeness, we chose app-pairs from a large representative sample of popular mobile apps and categories. With respect to generalizability, iTunes and Google Play are the most popular systems currently, although apps in other app stores could have other characteristics. Regarding replication, all our data is publicly available [36], making the findings of our study reproducible.

5. RELATED WORK

Many studies have been conducted recently through mining and analysis of app store content. Most studies, however, have focused on one platform only.

App Ratings. Nayebi *et al.* [38] conducted surveys with users and developers to understand the release strategies used for mobile apps and found that an app's strategy affects its success and how it is perceived by the users. This is inline with our cross-platform findings. Mojica *et al.* [44] conducted a study to examine if the store rating of an app is able to capture how the users perceive an app. Their results indicate that the app store's current metric does not accurately reflect the users changing levels of satisfaction as the app evolves. Khalid *et al.* [30, 31] manually analyzed and tagged reviews of iOS apps to identify different issues

[4] https://itunes.apple.com/app/the-wonder-weeks/id529815782?mt=8

[5] https://play.google.com/store/apps/details?id=org.twisevictory.apps&hl=en

that users complain about. They studied 6,390 low star-rating reviews for 20 free iOS apps and uncovered 12 types of complaints. They found that functional errors, feature requests, and app crashes are the most frequent complaints while privacy and ethical issues, and hidden app costs are the complaints with the most negative impact on app ratings. Wiscom [14] is a tool that analyzes ratings and user reviews at three different levels of detail, and analyzes why users hate or like a certain app. CRISTAL [40] was proposed to track informative user reviews to changes in the app's source code. Using this tool, a positive relationship between implementing the user requests in reviews and the app's overall success measured in terms of ratings was found. Tian *et al.* [49] conducted a study to understand how high-rated apps differ from low-rated ones. Their results indicate that the size of an app, number of promotional images, and the target SDK version contribute the most to an app's rating. Vasquez *et al.* [34] investigated how the fault and change-proneness of APIs used by free Android apps relate to their success estimated as the average rating provided by the users. They [7] also surveyed 45 Android developers and found that apps having high user ratings use APIs that are less fault- and change-prone than the APIs used by low rated apps.

User Reviews. Guzman *et al.* [20] compared the performance of various machine learning algorithms for classifying reviews and found that Logistic Regression and Neural Network classifiers outperform Naive Bayes and SVM models. Lacob *et al.* [24] found that 23% of reviews represent feature requests. They proposed a prototype for automatic retrieval of mobile app feature requests from online reviews. Chen *et al.* [8] found that 35% of app reviews contain information that can directly help developers improve their apps. They proposed AR-Miner, a technique to extract the most informative user reviews. Panichella *et al.* [41] built on top of AR-Miner to automatically classify app reviews into different categories. Pagano *et al.* [39] carried out an exploratory study on over one million reviews from iOS apps to determine their potential for requirements engineering processes. They found that most of the feedback is provided shortly after a new release. Guzman *et al.* [21] proposed an approach using NLP and topic modelling to analyze user sentiment of certain app features. MARK [51] is a keyword-based framework for semi-automatically classifying user reviews. Mercado *et al.* [35] used NLP models to classify 787,228 user reviews for 50 hybrid apps. They found that on average hybrid apps tend to have more user complaints concerned with reliability and performance.

App Descriptions. Gorla *et al.* [18] clustered Android apps by their descriptions to identify potentially malicious outliers in terms of API usage. Their CHABADA tool identified several anomalies in a set of 22K Android apps. Al-Subaihin*et al.* [1] used agglomerative hierarchical clustering to group apps based on features extracted from app descriptions. Kuznetsov *et al.* [32] developed a technique to compare the natural language topics of an app's user interface against the topics from its description on the app store. They analyzed a dataset of 3735 apps and found multiple apps which exhibit discrepancies between their user interface and description. Seneviratne *et al.* [46] manually inspected and analyzed why some spam apps get removed from the app store. They used their manual observations to create an automated technique which looks at an app's metadata such as description, reviews, identifier and detects if the app is indeed spam. Chen *et al.* [9] developed mechanisms to compare the maturity ratings across the iOS and Android platforms. They used the iOS ratings as ground truth and found that over 30% of Android apps have unreliable maturity ratings. This work is closest to our work since it examines two different platforms however, there are a few differences. First, our work compares various app attributes such as ratings, stars, prices and AUR, with the goal of understanding how apps differ on the two platforms. Furthermore, the work presented in [9] uses a dataset of 1,464 app-pairs, while our work uses a dataset of 80,169 app-pairs, from all app categories. Chen's work is an excellent example for the need for an app-pair study along with a large diversified dataset.

Most previous work has focused on studying one platform only, our work on the other hand, aims at characterizing the differences in mobile app-pairs across two different platforms.

6. CONCLUSION & FUTURE WORK

In this paper, we present a large-scale study of cross-platform mobile apps. We mined 80K iOS and Android app-pairs and compared their app-store attributes. We built three automated classifiers and classified 1.7 million reviews to understand how user complaints and concerns vary across platforms. Additionally, we contacted app developers to understand some of the major differences in app-pair attributes such as prices, update frequencies, AUR rates and top rated apps existing only on one platform.

For future work, the testing and analysis of apps across multiple platforms could be explored. While our recent study [11] is a step toward better understanding of it, with the increased fragmentation in devices and platforms, it still remains a challenge to test mobile apps across varying hardware and platforms [37]. Additionally, app features, extracted from app descriptions, can be used to compare on different platforms. Finally, while we combined the stars and ratings to measure how an app is perceived by users, in the future we will explore ways of measuring user retention, number of downloads, user loyalty, or recency.

7. REFERENCES

[1] A. A. Al-Subaihin, F. Sarro, S. Black, L. Capra, M. Harman, Y. Jia, and Y. Zhang. Clustering mobile apps based on mined textual features. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, pages 38:1–38:10, New York, NY, USA, 2016. ACM.

[2] M. Ali and A. Mesbah. Mining and characterizing hybrid apps. In *Proceedings of the 1st International Workshop on App Market Analytics (WAMA 2016)*, page 7 pages, 2016.

[3] Android Market Stats. http://www.appbrain.com/stats/.

[4] App Store Metrics. http://148apps.biz/app-store-metrics/.

[5] Apple Store Crawler. https://github.com/MarcelloLins/Apple-Store-Crawler.

[6] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden. Mining apps for

abnormal usage of sensitive data. In *Proceedings of the International Conference on Software Engineering*, ICSE 2015. ACM, 2015.

[7] G. Bavota, M. Linares-Vasquez, C. Bernal-Cardenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk. The impact of API change- and fault-proneness on the user ratings of Android apps. *IEEE Transactions on Software Engineering*, 99, 2015.

[8] N. Chen, J. Lin, S. C. H. Hoi, X. Xiao, and B. Zhang. AR-miner: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the International Conference on Software Engineering*, ICSE, pages 767–778. ACM, 2014.

[9] Y. Chen, H. Xu, Y. Zhou, and S. Zhu. Is this app safe for children?: A comparison study of maturity ratings on Android and iOS applications. In *Proceedings of the International Conference on World Wide Web*, WWW '13, pages 201–212. ACM, 2013.

[10] J. W. Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches.* Sage Publications, Incorporated, 2013.

[11] M. Erfani Joorabchi, M. Ali, and A. Mesbah. Detecting inconsistencies in multi-platform mobile apps. In *Proceedings of the International Symposium on Software Reliability Engineering*, ISSRE '15. IEEE Computer Society, 2015.

[12] M. Erfani Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, ESEM'13, pages 15–24. ACM, 2013.

[13] M. A. Figueiredo. Lecture notes on bayesian estimation and classification. *Instituto de Telecomunicacoes-Instituto Superior Tecnico*, page 60, 2004.

[14] B. Fu, J. Lin, L. Li, C. Faloutsos, J. Hong, and N. Sadeh. Why people hate your app: Making sense of user feedback in a mobile app store. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 1276–1284. ACM, 2013.

[15] R. Gallo, P. Hongo, R. Dahab, L. C. Navarro, H. Kawakami, K. Galvão, G. Junqueira, and L. Ribeiro. Security and system architecture: Comparison of Android customizations. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '15, pages 12:1–12:6, New York, NY, USA, 2015. ACM.

[16] Google Play Store Crawler. https://github.com/MarcelloLins/GooglePlayAppsCrawler.

[17] Goole Play Store Review scraper. https://github.com/jkao/GooglePlayScraper.

[18] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller. Checking app behavior against app descriptions. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 1025–1035. ACM, 2014.

[19] L. Guerrouj, S. Azad, and P. C. Rigby. The influence of app churn on app success and stackoverflow discussions. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 321–330. IEEE, 2015.

[20] E. Guzman, M. El-Haliby, and B. Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 771–776, Nov 2015.

[21] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 153–162, 2014.

[22] D. Han, C. Zhang, X. Fan, A. Hindle, K. Wong, and E. Stroulia. Understanding Android fragmentation with topic analysis of vendor-specific bugs. In *Proceedings of the Working Conference on Reverse Engineering (WCRE)*, pages 83–92, Oct 2012.

[23] M. Harman, Y. Jia, and Y. Zhang. App store mining and analysis: MSR for app stores. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*, pages 108–111. IEEE, June 2012.

[24] C. Iacob and R. Harrison. Retrieving and analyzing mobile apps feature requests from online reviews. In *Proceedings of the Working Conference on Mining Software Repositories*, MSR '13, pages 41–44. IEEE Press, 2013.

[25] C. Iacob, V. Veerappa, and R. Harrison. What are you complaining about?: A study of online reviews of mobile applications. In *Proceedings of the 27th International BCS Human Computer Interaction Conference*, BCS-HCI '13, pages 29:1–29:6. British Computer Society, 2013.

[26] IMDB. http://www.imdb.com/help/show_leaf?votestopfaq.

[27] T. F. in Android Apps. https://play.google.com/store/apps/collection/topselling_free?hl=en.

[28] iTunes App Store Review scraper. https://github.com/grych/AppStoreReviews.

[29] iTunes RSS Feed Generator. https://rss.itunes.apple.com/ca/.

[30] H. Khalid. On identifying user complaints of iOS apps. In *Proceedings of the International Conference on Software Engineering*, ICSE '13, pages 1474–1476. IEEE Press, 2013.

[31] H. Khalid, E. Shihab, M. Nagappan, and A. Hassan. What do mobile app users complain about? a study on free iOS apps. *IEEE Software*, 99, 2014.

[32] K. Kuznetsov, V. Avdiienko, A. Gorla, and A. Zeller. Checking app user interfaces against app descriptions. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, pages 1–7, New York, NY, USA, 2016. ACM.

[33] D. Lavid Ben Lulu and T. Kuflik. Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device. In *Proceedings of the International Conference on Intelligent User Interfaces*, IUI '13, pages 297–306. ACM, 2013.

[34] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. API change and fault proneness: A threat to the success of Android apps. In *Proceedings of the International Symposium on the Foundations of Software*

Engineering, ESEC/FSE 2013, pages 477–487. ACM, 2013.

[35] I. T. Mercado, N. Munaiah, and A. Meneely. The impact of cross-platform development approaches for mobile applications from the user's perspective. In *Proceedings of the International Workshop on App Market Analytics*, WAMA 2016, pages 43–49, New York, NY, USA, 2016. ACM.

[36] Mining iOS and Android mobile app-pairs: Toolset and dataset. https://github.com/saltlab/Minning-App-Stores.

[37] M. Nagappan and E. Shihab. Future trends in software engineering research for mobile apps. In *Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering, FoSE*, 2016.

[38] M. Nayebi, B. Adams, and G. Ruhe. Release practices for mobile apps – what do users and developers think? In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 552–562, March 2016.

[39] D. Pagano and W. Maalej. User feedback in the appstore: An empirical study. In *Proceedings of the International Requirements Engineering Conference (RE)*, pages 125–134. IEEE, 2013.

[40] F. Palomba, M. Linares-Vasquez, G. Bavota, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. User reviews matter! tracking crowdsourced reviews to support evolution of successful apps. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015.

[41] S. Panichella, A. D. Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. How can i improve my app? classifying user reviews for software maintenance and evolution. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015.

[42] I. Ruiz, M. Nagappan, B. Adams, and A. Hassan. Understanding reuse in the Android market. In *Program Comprehension (ICPC), International Conference on*, pages 113–122. IEEE, June 2012.

[43] I. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. Hassan. On the relationship between the number of ad libraries in an Android app and its rating. *IEEE Software*, 99, 2014.

[44] I. M. Ruiz, M. Nagappan, B. Adams, T. Berger, S. Dienst, and A. Hassan. An examination of the current rating system used in mobile app stores. *IEEE Software*, PP(99):1–1, 2015.

[45] Scikit Learn: Machine Learning in Python. http://scikit-learn.org/stable/index.html.

[46] S. Seneviratne, A. Seneviratne, M. A. Kaafar, A. Mahanti, and P. Mohapatra. Early detection of spam mobile apps. In *Proceedings of the International Conference on World Wide Web*, WWW, pages 949–959. ACM, 2015.

[47] Statista. Number of apps available in leading app stores. http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/.

[48] M. Thelwall, K. Buckley, and G. Paltoglou. Sentiment strength detection for the social web. *J. Am. Soc. Inf. Sci. Technol.*, 63(1):163–173, Jan. 2012.

[49] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan. What are the characteristics of high-rated apps? a case study on free Android applications. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015.

[50] N. Viennot, E. Garcia, and J. Nieh. A measurement study of google play. In *The 2014 ACM International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '14, pages 221–233, New York, NY, USA, 2014. ACM.

[51] P. M. Vu, T. T. Nguyen, H. V. Pham, and T. T. Nguyen. Mining user opinions in mobile app reviews: A keyword-based approach. *CoRR*, abs/1505.04657, 2015.

[52] L. Wu, M. Grace, Y. Zhou, C. Wu, and X. Jiang. The impact of vendor customizations on Android security. In *Proceedings of the Conference on Computer and Communications Security*, CCS '13, pages 623–634, New York, NY, USA, 2013. ACM.