Mining and Characterizing Cross-Platform Apps

by

Mohamed Hassanein Tawfik Ibrahim Ali

B.Sc., The University of British Columbia, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL
STUDIES

(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

December 2016

© Mohamed Hassanein Tawfik Ibrahim Ali, 2016
Abstract

Smartphones and the applications (apps), which run on them, have grown tremendously over the past few years. To capitalize on this growth and attract more users, developing the same app for different platforms has become a common industry practice. However, each mobile platform has its own development language, Application program interfaces (APIs), software development kits (SDKs) and online stores 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.

Further, we take a closer look at a special category of cross-platform apps, which are built using Cross Platform Tools (CPTs). CPTs allow developers to use a common code-base to simultaneously create apps for multiple platforms. Apps created using these CPTs are called hybrid apps. We mine 15,512 hybrid apps; measure their aggregated user-perceived ratings and compare them to native apps of the same category.
This thesis presents two large-scale empirical studies on cross-platform apps. The work presented was conducted by myself in collaboration with my supervisor, Professor Ali Mesbah. Chapter 2 of this thesis was done with equal collaboration from Mona Erfani Joorabchi. I was responsible for devising the approach and collecting the data, running the experiments, analyzing the results and writing the manuscript. My collaborators guided me with the creation of the methodology, analysis of results, editing and writing portions of the manuscript. The results described in Chapter 2 are submitted as a full paper to an ACM SIGSOFT conference and are currently under review. The work presented in Chapter 3 has been published as a workshop paper in November 2016 in the Proceedings of the 1st International Workshop on App Market Analytics (WAMA) [4].
# Table of Contents

Abstract ................................................................. ii
Preface ................................................................. iii
Table of Contents .................................................... iv
List of Tables ......................................................... vi
List of Figures ......................................................... vii
Acknowledgments ..................................................... viii

1 Introduction ......................................................... 1
   1.1 Contributions .................................................. 3
   1.2 Thesis Organization ............................................ 4

2 Same App, Different App Stores:  
A Large-Scale Study of Cross-Platform Apps ..................... 5
   2.1 Methodology .................................................... 6
      2.1.1 Data Collection ......................................... 6
      2.1.2 Matching Apps to Find App-Pairs ....................... 7
      2.1.3 App-store Attribute Analysis ........................... 10
      2.1.4 User Reviews ............................................ 10
      2.1.5 User-Perceived Rating .................................. 12
      2.1.6 Cross-platform Complaint Analysis ..................... 14
      2.1.7 Datasets and Classifiers ................................ 15
   2.2 Findings ....................................................... 15
      2.2.1 Prevalence and Attributes (RQ1) ....................... 15
      2.2.2 Top Rated Apps (RQ2) .................................. 21
      2.2.3 Aggregated User-Perceived Ratings (RQ3) ............... 22
      2.2.4 Complaints Across Platforms (RQ4) ..................... 24
   2.3 Discussion ..................................................... 30
   2.4 Conclusions ................................................... 32
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Collected app-pair attributes</td>
<td>7</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>Real-world reviews and their classifications</td>
<td>12</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Ranking apps using different metrics.</td>
<td>14</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>iOS &amp; AND descriptive statistics: Cluster Size (C), Ratings (R), Stars (S), and Price (P).</td>
<td>17</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>Statistics of 14 Apps used to build the classifiers (C1 = Generic Classifier, C2 = Sentiment Classifier, NB = Naive Bayes Algorithm, SVM = Support Vector Machines Algorithm)</td>
<td>26</td>
</tr>
<tr>
<td>Table 2.6</td>
<td>Descriptive statistics for iOS &amp; AND reviews: Problem Discovery (PD), Feature Request (FR), Non-informative (NI), Positive (P), Negative (N), Neutral (NL), and AUR.</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.7</td>
<td>Descriptive statistics for problematic reviews: App Feature (AF), Critical (CR), Post Update (PU), and Price Complaints (PC).</td>
<td>28</td>
</tr>
<tr>
<td>Table 3.1</td>
<td>Number of Hybrid apps using different CPTs</td>
<td>37</td>
</tr>
<tr>
<td>Table 3.2</td>
<td>Descriptive statistics for the hybrid apps: Ratings (R), Stars (S), and Downloads (D).</td>
<td>39</td>
</tr>
<tr>
<td>Table 3.3</td>
<td>Descriptive statistics for the hybrid apps: AUR</td>
<td>40</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Overview of our methodology.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Android Cluster for Swiped app.</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>a) Groupon and b) Scribblenauts apps. Android apps are shown on the top and iOS apps at the bottom.</td>
<td>9</td>
</tr>
<tr>
<td>2.4</td>
<td>Matching App-pair Criteria.</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>Clusters.</td>
<td>18</td>
</tr>
<tr>
<td>2.6</td>
<td>Ratings.</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Stars.</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>Prices.</td>
<td>23</td>
</tr>
<tr>
<td>2.9</td>
<td>AUR scores calculated per platform.</td>
<td>27</td>
</tr>
<tr>
<td>2.10</td>
<td>AUR scores calculated across the platforms.</td>
<td>29</td>
</tr>
<tr>
<td>2.11</td>
<td>The rates of classifiers’ categories for our 2K app-pairs, where each dot represents an app-pair.</td>
<td>38</td>
</tr>
<tr>
<td>2.12</td>
<td>The rates of complaints categories for our 2K app-pairs, where each dot represents an app-pair.</td>
<td>39</td>
</tr>
<tr>
<td>3.1</td>
<td>Number of apps in each category created using the PhoneGap CPT.</td>
<td>40</td>
</tr>
<tr>
<td>3.2</td>
<td>Number of apps in each category created using the Titanium CPT.</td>
<td>41</td>
</tr>
<tr>
<td>3.3</td>
<td>Number of apps in each category created using the Adobe Air CPT.</td>
<td>42</td>
</tr>
<tr>
<td>3.4</td>
<td>AUR rates for the apps overall and across each CPT.</td>
<td>43</td>
</tr>
<tr>
<td>3.5</td>
<td>Average AUR for app categories for Native and Hyrbid Apps.</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>AUR scores for 1400 hybrid app-pairs. Each pair of diamond(iOS) and square(Android) dots represents an app. The solid and dashed lines show the trend of AUR across the apps.</td>
<td>45</td>
</tr>
</tbody>
</table>
Acknowledgments

I would like to thank my supervisor Dr. Ali Mesbah for giving me the chance to pursue my masters and be part of his research group. His experience, professionalism and valuable insights have been key to the success of this work. His mentorship and guidance has been exemplary and I am confident that the lessons I learned from him will be valuable for my engineering career.

I would like to thank my family especially my dad, Hassanein Ali, my mom Nabila Mohamed and my brothers, Karim, Diaa and Ahmed for their constant support and love. Thank you for your encouragement and for inspiring me to become a better person. Special thanks to my friends at the SALT lab for their valuable help, feedback and for creating a fun and productive environment. I would like to acknowledge and thank all my friends for their support and motivation.

Last but not least, I would like to thank God for his countless blessings and for giving me the chance to pursue my masters.
Chapter 1

Introduction

Smartphone use has grown tremendously in the last few years. This increasing popularity can be attributed to the introduction of Apple’s App Store and Google’s Play Store which were launched in 2008. These stores serve as the primary medium for the distribution of mobile applications (apps). Through app stores, users can download and install apps on their mobile devices. App stores also provide an important channel for app developers to collect user feedback, such as the overall rating of their app, issues or feature requests through user reviews.

To attract as many users as possible, developers often implement the same app for multiple mobile platforms [32]. 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 [33]. 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 [22, 59, 88], app descriptions [46, 66, 94], and app bytecode [12, 90, 91]. 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 [9] and Android [7] dominate the app market, each with over 1.5 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, Chapter 2 of this thesis presents a large-scale study on mo-
bile 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 [84]. Our mined artifact of more than 80,000 app-pairs is the first publicly available dataset that can be used by researchers to go beyond Android and study different aspects of cross-platform apps.

Currently there are three popular ways to build mobile apps. The first method, “Native”, allows developers to use the software development kits (SDKs) and frameworks for the targeted platform to build the app [55]. “Native”, allows developers to use all of the capabilities of a device, provides the best performance and is distributed through the platform’s dedicated app store. However, implementing a native app for multiple platforms requires familiarity with the languages, APIs and SDKs of each specific platform, which is resource intensive and time consuming.

The second method, “Mobile Web App”, uses web technologies such as HTML, CSS and Javascript to build the application as one website which is optimized for mobile devices. This approach allows the app to be used across multiple platforms which reduces development cost and time. However, “Mobile Web App” cannot use device specific hardware features such as the camera or accelerometer [55].

The third method to build an app is “Hybrid”, which bridges the gap between the first two methods. “Hybrid” uses a common code base to simultaneously deliver native-like apps to multiple platforms. These apps can access the hardware features of the device and are also distributed using the platform’s app store. Hybrid apps are created using Cross Platform Tools (CPTs). These CPTs provide two approaches to create apps; the first approach allows the developer to use web technologies such as HTML, CSS and Javascript to create a code base which runs in an internal browser (WebView) that is wrapped in a native app. Examples of CPTs us-
ing this approach include PhoneGap [3] and Trigger.io [102]. The other approach
CPTs offer allows the developer to write their code in a language such as C# or
Javascript which then gets compiled to native code for each platform. Examples
of CPTs that use this approach include Xamarin [110], Appcelerator Titanium [10]
and Adobe Air [1]. Mobile Apps created using a CPT are referred to as hybrid
apps and we use that term to refer to these apps throughout this paper.

A recent study conducted by Viennot et al. [103] found that out of 1.1 Million
Android apps, 129,800 (11.8 %) were hybrid apps.

In Chapter 3 we present a large-scale study on hybrid apps in order to un-
derstand their behavior and characteristics by analyzing their various app-store at-
tributes. Additionally, we compare hybrid apps across the Android and iOS plat-
forms. Finally we compare hybrid apps to native ones in terms of user perception.
By presenting real market data, our study can help developers decide if using a
CPT to create their apps is the best solution.

1.1 Contributions

This thesis makes the following main contributions:

• The first 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 [80].

• 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, rat-
ings, prices, versions, and updates across platforms.

• Qualitative developer feedback, providing insights into the cause of vari-
  ations in development, prices, and user-perceived ratings across platforms.

• Sentiment and complaints analysis of user reviews across app-pairs.

• A technique to identify hybrid apps along with a large dataset of 15,512
  hybrid apps. Our dataset is publicly available [58].
• A characterization of hybrid app attributes such as ratings, stars and down-
loads.

• A comparison between hybrid and native apps in terms of user perception.

• A comparison of hybrid apps across the Android and iOS platforms.

1.2 Thesis Organization

In Chapter 2 of this thesis we describe the empirical study we conducted to mine and characterize cross-platform app-pairs, the results of our analysis and their implications on developers and researchers. In Chapter 3 we present our empirical study on hybrid apps. This chapter outlines how we mine hybrid apps and presents the results of our analysis on the app attributes. Chapter 4 discusses the related work, and Chapter 5 concludes and presents future research directions.
Chapter 2

Same App, Different App Stores:
A Large-Scale Study of Cross-Platform Apps

Summary

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.

1This chapter is submitted to an ACM SIGSOFT conference.
2.1 Methodology

Our analysis is based on a mixed-methods research approach [27], 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 2.1 depicts our overall approach. We use this figure to illustrate our methodology throughout this section.

2.1.1 Data Collection

To collect Android and iOS apps along with their attributes (Box 1 in Figure 2.1), we use two open-source crawlers, namely Google Play Store Crawler [44] and Apple Store Crawler [11] 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 2.1 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 [80].

### 2.1.2 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); (2) the same app could have slightly different names across the platforms (See Figure 2.3–a); (3) the same app could have slightly different developer names across the platforms (See Figure 2.3–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 2.1) 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 2.2 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.

![Android Cluster for Swiped app.](figure)

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 2.1), we noticed that Android and iOS apps have different naming conventions for app names and developer names. For instance, Figure 2.3–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 2.3–b shows the ‘Scribblenauts Remix’ app, which has the exact name on both platforms, but has differences in the developer’s name.

![Figure 2.3: a) Groupon and b) Scribblenauts apps. Android apps are shown on the top and iOS apps at the bottom.](image)

Figure 2.4 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 2.3 as app-pairs.

<table>
<thead>
<tr>
<th>ID</th>
<th>App-pair Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>EXACT(AppName) &amp; EXACT(DevName)</td>
</tr>
<tr>
<td>S</td>
<td>SIMILAR(AppName) &amp; SIMILAR(DevName)</td>
</tr>
</tbody>
</table>

Figure 2.4: Matching App-pair Criteria.

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 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 2.4.

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.

2.1.3 App-store Attribute Analysis

To address RQ1, (Box 4 in Figure 2.1) we compare the captured attributes between the iOS and Android app-pairs and present the results in Section 4.3.

To address RQ2, we use the iTunes Store RSS Feed Generator [63] 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 [61] 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 2.1). 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.

2.1.4 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 [62] and the Google Play Store Review Scraper [45]. In total, we retrieve 1.7M user reviews for 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 and machine learning to train two classifiers (Box 6 in Figure 2.1). Each classifier can automatically put a review into one of its three classes.

**Generic Feedback Analysis.** As shown in Table 2.2, 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 [22, 88] 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 2.2) classes of \{**Positive**, **Negative**, **Neutral**\} between the reviews of app-pairs. We use these rules to assign class labels to review instances. Table 2.2 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 2.2. Subsequently, any uncertainties were cross-validated and resolved through discussions and refinements between the authors. Overall,
Table 2.2: Real-world reviews and their classifications.

<table>
<thead>
<tr>
<th>C1 – Generic Feedback Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Problem Discovery: “Videos don’t work. The sound is working but the video is just a black screen.”</td>
</tr>
<tr>
<td>2 Feature Request: “I would give it a 5 if there were a way to exclude chain restaurants from dining options.”</td>
</tr>
<tr>
<td>3 Non-informative: “A far cry from Photoshop on the desktop, but still a handy photo editor for mobile devices with...”</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C2 – Sentiment Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Positive: “Amazing and works exactly how I want it to work. Nothing bad about this awesome and amazing app!”</td>
</tr>
<tr>
<td>2 Negative: “The worst, I downloaded it with quite a lot of excitement but ended up very disappointed”</td>
</tr>
<tr>
<td>3 Neutral: “No complaints because I’m not a complainer save your option for something that matters”</td>
</tr>
</tbody>
</table>

we label 2.1K reviews for training each of the two classifiers (Box 7 in Figure 2.1).

**Building Classifiers**

To build our classifiers, 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. Next, we preprocess the text, tokenize it and filter stop words. 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 [93] 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 ~1.7M reviews of the 2K app-pairs.

**2.1.5 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. [100], many users download an app without ever
using it. More importantly, as explained in Section 2.1.1, 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 success [50, 99].

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 [14, 49, 53, 100]. 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 2.1) of an app as follows:

\[
\text{AUR}(\text{app}_i) = \frac{v_i \times r_i}{v_i + m} + \frac{m \times c}{v_i + m}
\]  

(2.1)

where

1. \(v_i\) is the number of ratings for \(\text{app}_i\),
2. \(r_i\) is the average stars for the \(\text{app}_i\),
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 [60] of the Internet Movie Database (IMDB), which uses user votes to generate the top 250 movies. The formulae is believed to provide a true Bayesian estimate [35, 60].
Table 2.3: Ranking apps using different metrics.

<table>
<thead>
<tr>
<th>App</th>
<th>Ratings (R)</th>
<th>Stars (S)</th>
<th>Rank S</th>
<th>Rank R</th>
<th>Rank AUR</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>5.0</td>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>4.8</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1825</td>
<td>4.7</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>11</td>
<td>2.1</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>E</td>
<td>67</td>
<td>4.6</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>F</td>
<td>2796</td>
<td>1.8</td>
<td>6</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>

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 ($c$) across the 100 apps were 142 and 4.1, respectively. Table 2.3 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.

2.1.6 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 2.1). 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 2.1), 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.

2.1.7 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 [80].

2.2 Findings

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

2.2.1 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\(^2\) and the largest iOS cluster contains 65 apps\(^3\). Additionally, 7,845 Android and 9,016 iOS clusters have more than one item. The first row of Table 2.4 shows descriptive statistics along with p-value (Mann-Whitney) for cluster sizes, ignoring clusters of size 1. Figure 2.5 depicts the cluster sizes for the two platforms. We ignore outliers for legibility. The results are statistically significant \((p < 0.05)\) and show that while Android clusters deviate more than iOS clusters, the median in iOS is higher than Android by one. This could be explained perhaps by the following two observations: (1) not all iOS apps are universal apps (i.e., run on all iOS devices) and some apps have both iPhone-only and iPad-only apps instead of one universal app; (2) iOS has more free and pro versions of the same app than Android.

\(^2\)https://play.google.com/store/search?q=Kira-Kira&c=apps&hl=en
\(^3\)https://itunes.apple.com/us/developer/urban-fox-production-llc/id395696788
Prevalence of app-pairs

We found 80,169 (∼80,000) exact app-pairs (Criteria E in Figure 2.4), 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. While our dataset contains apps from 22 Apple and 25 Google categories, most of the pairs belong to popular categories, which exist on both platforms: \{Games, Business, Lifestyle, Education, Travel, Entertainment, Music, Finance, Sports\}.

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 2.4. However, when we only consider apps with at least one rating, the median increases to 21 for iOS and 11 for Android (See Figure 2.6). 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
Table 2.4: iOS & AND descriptive statistics: Cluster Size (C), Ratings (R), Stars (S), and Price (P).

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Min</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
<th>Max</th>
<th>P-Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>iOS</td>
<td>2</td>
<td>3.30</td>
<td>3.00</td>
<td>2.11</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>2</td>
<td>3.00</td>
<td>2.00</td>
<td>3.69</td>
<td>219</td>
<td>0</td>
</tr>
<tr>
<td>R</td>
<td>iOS</td>
<td>5</td>
<td>1,935.00</td>
<td>21.00</td>
<td>26,827.23</td>
<td>1,710,251</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>1</td>
<td>4,892.00</td>
<td>11.00</td>
<td>171,362.40</td>
<td>28,056,146</td>
<td>0</td>
</tr>
<tr>
<td>R*</td>
<td>iOS</td>
<td>0</td>
<td>353.10</td>
<td>0.00</td>
<td>11,483.19</td>
<td>1,710,251</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>0</td>
<td>3,302.00</td>
<td>3.00</td>
<td>140,807.60</td>
<td>28,056,146</td>
<td>0</td>
</tr>
<tr>
<td>S</td>
<td>iOS</td>
<td>1</td>
<td>3.80</td>
<td><strong>4.00</strong></td>
<td>0.90</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>1</td>
<td>4.04</td>
<td><strong>4.10</strong></td>
<td>0.82</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>S*</td>
<td>iOS</td>
<td>0</td>
<td>0.70</td>
<td>0.00</td>
<td>1.52</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>0</td>
<td>2.73</td>
<td>3.70</td>
<td>2.01</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>P</td>
<td>iOS</td>
<td>0.99</td>
<td>3.58</td>
<td>1.99</td>
<td>9.73</td>
<td>500</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>0.95</td>
<td>4.00</td>
<td>2.11</td>
<td>9.81</td>
<td>210</td>
<td>0</td>
</tr>
</tbody>
</table>

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

5% of the pairs, iOS apps have more users rating them (on average 1,966 more users). Additionally, the results of ratings in Table 2.4 are statistically significant ($p < 0.05$), indicating that Android users tend to rate apps more than iOS users. The categories with the highest ratings were {Personalization, Communication, Games} on Android and {Games, Social Networking, Photo & Video} on iOS.

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 2.7). 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. Additionally, while the results of stars are statistically significant ($p < 0.05$), the observed differences, having almost the same medians (see Table 2.4), are not indicative, meaning that although Android users tend to star apps more than iOS users, the starred app-pairs have similar stars. The categories with the highest number of stars were {Weather, Music & Audio, Comics} on Android and {Games, Weather, Photo & Video} on iOS.

**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
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 2.4 shows, the mean and median for paid apps are slightly higher for Android compared to iOS. The categories with the most expensive apps were \{Medical, Books & Reference, Education\} on Android and \{Medical, Books, Navigation\} on 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 2.8.

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[^1] — in the form of (major.minor.patch)

[^1]: [http://semver.org](http://semver.org)
— 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. Interestingly, 30% of the app-pairs have update dates which are more than 6 months apart. To understand why developers update their apps inconsistently across the platforms, we emailed all the developers of app-pairs which were recently updated (after January 2016) on either platform; and in which the other platform has not been updated in 80 days or more. Out of 65 emails, we received 15 responses and categorized the reasons below:

**Ease of releasing updates** e.g., “we are experimenting with a new 3D printing feature, and wanted to try it on Android before we released it on iOS. As you know, developers can release updates quickly to fix any problems on Android, but on iOS, we have to wait a week or two while Apple reviews the game.”

**Preferring one platform over the other** for various reasons, e.g., “while there are many Android handsets and potentially many downloads, this doesn’t translate well to dollars spent, relative to iOS.”

**Developer skills and expertise** The developers might be more skilled at building apps for one of the platforms than the other; e.g., “I learned iOS first and am developing for iOS full time, so everything is easier for me with iOS.”

**Update due to a platform-specific feature** e.g., “only updated the iOS version to switch over to AdMob as the advertising network for iOS. Apple announced that iAd is being discontinued.”

We have to note that, similar to the reasons behind the price differences, some of the developers we contacted, mentioned that the development/maintenance cost of the app could affect updates on either platform.
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.

2.2.2 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 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.

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.

**Finding 5:** More than 80% of the top-rated apps are cross-platform.

### 2.2.3 Aggregated User-Perceived Ratings (RQ3)

Figure 2.9 shows the AUR rates for our app-pairs, computed using formulae [2.1]. 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 18199 and the average number of stars ($c$) was 3.9. For iOS, $m$ was 1979 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 2.10 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 versions of cross-platform apps receive higher user-perceived ratings compared to the iOS versions.

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 [103], 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 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.

2.2.4 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 2.5, where

\[ \text{Precision} = \frac{TP}{TP + FP} \quad \text{and} \quad \text{Recall} = \frac{TP}{TP + FN}. \]

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. [88] (0.72) and Chen et al. [22] (0.79). We selected the classifiers with the best F-measures and used them to classify 1,702,100 (∼1.7M) reviews for 2,003 (∼2K) app-pairs.

**Sentiment and Generic Reviews**

Figure 2.11 plots the distribution of the rates for the main categories in the sentiment and generic classifiers for our app-pairs. Each dot represents an app-pair. The descriptive statistics are shown in Table 2.6. 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 2.5: Statistics of 14 Apps used to build the classifiers (C1 = Generic Classifier, C2 = Sentiment Classifier, NB = Naive Bayes Algorithm, SVM = Support Vector Machines Algorithm)

<table>
<thead>
<tr>
<th>#</th>
<th>App</th>
<th>GoogleCategory</th>
<th>AppleCategory</th>
<th>F(C1-NB)</th>
<th>F(C2-NB)</th>
<th>F(C1-SVM)</th>
<th>F(C2-SVM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FruitNinja</td>
<td>Game(Arcade)</td>
<td>Game</td>
<td>0.77</td>
<td>0.68</td>
<td>0.83</td>
<td>0.75</td>
</tr>
<tr>
<td>2</td>
<td>UPSMobile</td>
<td>Business</td>
<td>Business</td>
<td>0.80</td>
<td>0.69</td>
<td>0.82</td>
<td>0.76</td>
</tr>
<tr>
<td>3</td>
<td>Starbucks</td>
<td>Lifestyle</td>
<td>Food &amp; Drink</td>
<td>0.75</td>
<td>0.63</td>
<td>0.84</td>
<td>0.77</td>
</tr>
<tr>
<td>4</td>
<td>YellowPages</td>
<td>Travel &amp; Local</td>
<td>Travel</td>
<td>0.78</td>
<td>0.62</td>
<td>0.85</td>
<td>0.75</td>
</tr>
<tr>
<td>5</td>
<td>Vine</td>
<td>Social</td>
<td>Photo &amp; Video</td>
<td>0.81</td>
<td>0.70</td>
<td>0.84</td>
<td>0.76</td>
</tr>
<tr>
<td>6</td>
<td>Twitter</td>
<td>Social</td>
<td>Social Networking</td>
<td>0.79</td>
<td>0.67</td>
<td>0.84</td>
<td>0.75</td>
</tr>
<tr>
<td>7</td>
<td>AdobePhotoShop</td>
<td>Photography</td>
<td>Photo &amp; Video</td>
<td>0.82</td>
<td>0.72</td>
<td>0.85</td>
<td>0.75</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Total</td>
<td>Average of 14 Apps</td>
<td></td>
<td></td>
<td>0.77</td>
<td>0.65</td>
<td>0.84</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Figure 2.11: The rates of classifiers’ categories for our 2K app-pairs, where each dot represents an app-pair.
Table 2.6: Descriptive statistics for iOS & AND reviews: Problem Discovery (PD), Feature Request (FR), Non-informative (NI), Positive (P), Negative (N), Neutral (NL), and AUR.

<table>
<thead>
<tr>
<th>ID Type</th>
<th>ID Type</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
<th>Max</th>
<th>P-Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>iOS</td>
<td>20.47</td>
<td>15.62</td>
<td>16.65</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>21.06</td>
<td>17.54</td>
<td>14.61</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>FR</td>
<td>iOS</td>
<td>17.50</td>
<td>16.03</td>
<td>10.81</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>13.71</td>
<td>12.50</td>
<td>8.88</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>NI</td>
<td>iOS</td>
<td>62.04</td>
<td>59.26</td>
<td>20.77</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>65.23</td>
<td>67.10</td>
<td>17.45</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>P</td>
<td>iOS</td>
<td>55.62</td>
<td>59.36</td>
<td>17.64</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>49.74</td>
<td>51.36</td>
<td>17.64</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>N</td>
<td>iOS</td>
<td>9.80</td>
<td>6.66</td>
<td>10.01</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>7.72</td>
<td>5.74</td>
<td>7.39</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>NL</td>
<td>iOS</td>
<td>34.57</td>
<td>32.45</td>
<td>14.87</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>42.54</td>
<td>41.73</td>
<td>13.97</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>AUR</td>
<td>iOS</td>
<td>38.22</td>
<td>76.21</td>
<td>76.03</td>
<td>3.06</td>
<td>99.75</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>52.53</td>
<td>80.88</td>
<td>80.90</td>
<td>2.07</td>
<td>97.48</td>
</tr>
</tbody>
</table>

Table 2.7: Descriptive statistics for problematic reviews: App Feature (AF), Critical (CR), Post Update (PU), and Price Complaints (PC).

<table>
<thead>
<tr>
<th>ID Type</th>
<th>ID Type</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
<th>Max</th>
<th>P-Val</th>
</tr>
</thead>
<tbody>
<tr>
<td>AF</td>
<td>iOS</td>
<td>53.71</td>
<td>54.29</td>
<td>18.15</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>60.55</td>
<td>60.92</td>
<td>16.25</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>CR</td>
<td>iOS</td>
<td>23.72</td>
<td>21.05</td>
<td>16.40</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>19.98</td>
<td>17.65</td>
<td>13.66</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td>PU</td>
<td>iOS</td>
<td>6.08</td>
<td>4.23</td>
<td>7.44</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>3.91</td>
<td>2.33</td>
<td>5.17</td>
<td>50.0</td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>iOS</td>
<td>7.76</td>
<td>5.00</td>
<td>9.41</td>
<td>100.0</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>AND</td>
<td>6.70</td>
<td>4.54</td>
<td>8.20</td>
<td>100.0</td>
<td>0.00</td>
</tr>
</tbody>
</table>

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 2.12 and Table 2.7, 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 [32, 37, 52, 109]. Examples of Android complaints include dissatisfaction with a certain functionality, incompatibility with a certain device/OS, and network and connectivity problems.
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.

2.3 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 [84], one of the obstacles with cross-platform analysis is the lack of a dataset for such apps. Our work provides the first large dataset with more than 80,000 exact app-pairs of iOS and Android apps [80]. 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. Our empirical study presented in Chapter 3 shows that hybrid apps have less variance in terms of user perception across the 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.
ers 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 four out study and resulted in F-scores that were comparable to other similar studies [22, 88].

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 2.4, 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\(^5\) on iOS has a pair on the Android platform with the name Baby Wonder Weeks Milestones\(^6\), 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 [80], making the findings of our study reproducible.

\(^5\)https://itunes.apple.com/app/the-wonder-weeks/id529815782?mt=8
2.4 Conclusions

In this chapter, we present the first large-scale study of cross-platform mobile app-pairs. We mined 80K iOS and Android pairs and compared their app-store attributes. We built three automated classifiers and classified 1.7M 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.
Chapter 3

Mining and Characterizing Hybrid Apps

Summary

Mobile apps have grown tremendously over the past few years. To capitalize on this growth and to attract more users, implementing the same mobile app for different platforms has become a common industry practice. Building the same app natively for each platform is resource intensive and time consuming since every platform has different environments, languages and APIs. Cross Platform Tools (CPTs) address this challenge by allowing developers to use a common code-base to simultaneously create apps for multiple platforms. Apps created using these CPTs are called hybrid apps. We mine 15,512 hybrid apps and present the first study of its kind on such apps. We identify which CPTs these apps use and how users perceive them. Further, we compare the user-perceived ratings of hybrid apps to native apps of the same category. Finally, we compare the user-perceived ratings of the same hybrid app on the Android and iOS platforms.

3.1 Methodology

We address the following research questions in our study:

RQ1. How prevalent are hybrid apps and which cross-platform tool is widely used?

This chapter appeared at the 1st International Workshop on App Market Analytics (WAMA 2016) [4]
RQ2. Does the choice of a cross-platform tool influence how it is perceived by users?

RQ3. How do hybrid apps compare to native apps of the same category in terms of user-perceived ratings?

RQ4. Does using a cross-platform tool ensure the app is perceived similarly on multiple platforms?

We first describe how we identify hybrid apps and then explain the analysis steps we performed on those apps.

3.1.1 Data Collection

The first step in our work is to mine apps from the app stores. To this end, we used the dataset of Android and iOS app-pairs introduced in Chapter 2. The dataset contains the attributes of 80,000 app-pairs. This dataset only contains the attributes of the apps and does not include the source code. We used a dataset of app-pairs since one of the main usages of CPTs is to generate the app for more than platform; by looking at app-pairs the chances of finding hybrid apps is much higher. Additionally, having the app-pair allows us to answer RQ4 which compares how a hybrid app is perceived on the iOS and Android platforms.

3.1.2 Finding Hybrid Apps

In order to determine if an app is hybrid, a manual approach can be used. Such an approach would involve installing the app on a device and exercising its functionality and try to infer from the user experience if the app is hybrid. The manual approach is time consuming and subjective to the user’s opinion, which can lead to many false positives. Furthermore, previous work [103] has quantified the number of hybrid apps and discovered that out of 1.1 Million Android apps 129,800 were hybrid. However, that dataset of hybrid apps is not publicly available and hence we had to build our own. For this work, we provide a fully automated technique to detect hybrid apps with 100% accuracy. Our technique supports the detection of apps made using 3 CPTs which are PhoneGap [3], Appcelerator Titanium [10] and
Adobe Air [1]. We target these CPTs since previous work [103] has shown that they are the most popular CPTs to develop hybrid apps.

To identify a hybrid app, we download its Android application package file (APK) which is the file format used by the Android operating system to distribute and install application software and middleware. Since the dataset used in 3.1.1 does not include the APKs, our technique first attempts to find it by using the app’s id to search through a dataset of 1.1 Million Android apps [85] and downloads the APK if it is available. Every Android APK includes a file called “classes.dex” which includes the classes of an Android app compiled in the dex file format. We use an open source tool called android-classyshark [17] to decompile the “classes.dex” into a readable format and then inspect it to check if an app uses a CPT. To determine if an app is hybrid we check its class contents for the following references; PhoneGap - “org.apache.cordova”, Appcelerator Titanium - “org.appcelerator.titanium”, Adobe Air -“com.adobe.air”. This technique of inspecting “classes.dex” and looking for references of usage of CPTs allows us to identity hybrid apps with 100% accuracy. We chose to download the Android APK instead of the iOS application package file because such information is not publicly available for iOS apps. To validate that our technique correctly identifies hybrid apps, we manually compared app icons and screen shots between the iOS and Android version of the apps and looked for clues such as having the exact UI layout across platforms or the use of un-native UI elements to conclude that an app is hybrid. We sampled 100 random apps and all of them were indeed hybrid, indicating that there are no false positives.

Algorithm 1 summarizes our approach and is used on the dataset in 3.1.1 to create a dataset of hybrid app-pairs. We use this dataset to answer RQ1 and the results are presented in Section 4.3.

3.1.3 User-Perceived-Rating

To measure how end-users perceive hybrid apps we use the same AUR metric introduced in Section 2.1.5. CPT vs App AUR. Since every CPT uses a different set of programming languages and techniques to generate hybrid apps, the goal of RQ2 is to analyze the relation-
Algorithm 1: Identifying hybrid apps

```
input : Collection of Apps
output: Collection of Hybrid Apps
begin
  phoneGapApps ← []
  titaniumApps ← []
  adobeAirApps ← []
  foreach i = 0, i < COUNT(APPS), i++ do
    app ← APPS[i]
    appId ← app.id
    apk ← lookForApk(appId)
    classes ← classyShark(apk)
    if classes.contains("org.apache.cordova") then
      phoneGapApps.append(app)
    end
    if classes.contains("org.appcelerator.titanium") then
      titaniumApps.append(app)
    end
    if classes.contains("com.adobe.air") then
      adobeAirApps.append(app)
    end
  end
end
```

ship between the CPT used to develop an app and how it is perceived by users. We use the metric discussed earlier to measure the AUR of apps generated by each CPT and compare the results in Section 4.3.

Hybrid vs Native Apps. While Hybrid apps have been increasing in popularity, native apps still dominate the market place due to their competitive advantage in terms of performance and supported features. The goal of RQ3, is to compare native apps and hybrid apps in terms of AUR. Since there is no way of directly comparing the apps, we compare the categories of apps with one another.

AUR Across Platforms. One of the main reasons for using a CPT is the convenience of generating an app for multiple platforms using a single code base. Furthermore, this ensures that the user experience is uniform across platforms. The final RQ examines whether these identically created apps are also identical in terms of their AUR. The dataset we use from Section 3.1.1 contains information about Google’s Android and Apple’s iOS app-pairs (the same app implemented for different platforms). We again use the formula discussed earlier to measure the AUR of hybrid apps across these platforms and present the comparisons in Section 4.3. Since we are performing cross-platform analysis, we require that the app
Table 3.1: Number of Hybrid apps using different CPTs

<table>
<thead>
<tr>
<th>CPT</th>
<th># of Apps</th>
<th>% of Hybrid Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneGap</td>
<td>10,562</td>
<td>68.0%</td>
</tr>
<tr>
<td>Titanium</td>
<td>2,881</td>
<td>18.5%</td>
</tr>
<tr>
<td>Adobe Air</td>
<td>2,069</td>
<td>13.5%</td>
</tr>
<tr>
<td>Total</td>
<td>15,512</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

has at least 1 rating on both platforms.

3.1.4 Dataset and Results

Our extracted dataset and results for the identified hybrid apps, as well as all our scripts are available for download at [58].

3.2 Findings

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

3.2.1 Prevalence and Popularity of CPTs (RQ1)

Out of the 80,000 apps that were inspected, our technique (described in 3.1.2) was able to find a total of 15,512 hybrid apps. As shown in Table 3.1, 10,562 hybrid apps used the PhoneGap CPT, 2,881 used Appcelerator Titanium, and 2,069 used Adobe Air. Furthermore, Figures 3.1–3.3 show the distribution of hybrid apps across the various categories in the Google Play store for each of the CPTs. The most popular categories for PhoneGap are business, lifestyle, travel & local, sports and education. For Titanium, the most popular categories are travel & local, lifestyle, finance, business and education. Finally for Adobe Air the most popular categories are games, education, business, lifestyle, and entertainment. Looking at the number of paid vs free apps, we found that all the hybrid apps in our dataset were free, regardless of which CPT they used.

Ratings, Stars & Downloads

We found that 79% of all the hybrid apps in our dataset, 76% of the PhoneGap apps, 81% of the Titanium apps and 90% of the AdobeAir apps have at least one
rating. As depicted in Table 3.2, the median for the number of ratings is 7 overall, 6 for PhoneGap, 7 for Titanium and 19 for Adobe Air.

As for the stars, the median is 4.20 overall, 4.3 for PhoneGap, 4.1 for Titanium and 3.9 for Adobe Air. 99% of the apps have been downloaded at least once. The median was 100 downloads overall, 100 for PhoneGap and Titanium and 500 for Adobe Air.

**Finding 8:** The PhoneGap CPT dominates the hybrid app market with a 68% share and is mainly being used to develop business, lifestyle, travel & local apps. Despite having the smallest market share at 13.5%, apps created using the AdobeAir CPT are downloaded and reviewed much more by users. The AdobeAir CPT is mainly used to develop games, and we attribute its popularity to this reason.

### 3.2.2 Effect of CPT on App’s AUR (RQ2)

Table 3.3 shows the AUR across all hybrid apps and for each of the CPTs. In our analysis, we only keep apps that contained at least 1 rating; this reduced the number of apps to 9948. Overall the median for AUR was 84%, 86% for PhoneGap, 82% for Titanium and 78% for AdobeAir. As can be seen in Figure 3.4, the PhoneGap
CPT generates apps with a better AUR score, followed by Titanium and AdobeAir. The results are statistically significant, with the p-value (Mann-Whitney) being 0.00 for all comparisons between CPTs.

**Finding 9**: The PhoneGap CPT results in apps which are better perceived by users. Furthermore, our results indicate that a high number of downloads and ratings does not necessarily mean that an app is well received by users. This is evident by the results of the AdobeAir CPT, which had the highest number of ratings but resulted in apps with the lowest AUR.
### Table 3.3: Descriptive statistics for the hybrid apps: AUR

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Min</th>
<th>Mean</th>
<th>Median</th>
<th>SD</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>20.00</td>
<td>81.48</td>
<td>84.00</td>
<td>17.11</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>PhoneGap</td>
<td>20.00</td>
<td>82.75</td>
<td>86.00</td>
<td>17.17</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>Titanium</td>
<td>20.00</td>
<td>80.14</td>
<td>82.00</td>
<td>17.86</td>
<td>100.00</td>
</tr>
<tr>
<td></td>
<td>AdobeAir</td>
<td>20.00</td>
<td>77.67</td>
<td>78.00</td>
<td>15.03</td>
<td>100.00</td>
</tr>
</tbody>
</table>

### 3.2.3 Hybrid Vs Native (RQ3)

Figure [3.5](#) plots the AUR scores of various app categories for Native and Hybrid Apps. To fairly compare the categories, we used an equal number of apps, distributed equally across the categories. The total number of Hybrid apps was 9948 and an equal number of Native apps was used. When exploring this RQ, we expected the native apps to be better perceived than the hybrid ones due to their advantage in terms of performance and supported native features. To our surprise, the hybrid apps were very close to the native ones in terms of AUR scores and even scored slightly higher in some of the categories. Out of the 25 possible categories for apps, the Hybrid scored higher in 17 of them.
Finding 10: The hybrid Apps analyzed had AUR scores which matched; and in some categories were higher than the native apps. This indicates that it is possible to create a successful hybrid app that is well perceived by users and that can compete with native variants.

3.2.4 AUR Across Platforms (RQ4)

Figure 3.6 plots the AUR scores of 1400 hybrid app-pairs. The results show that using a CPT to create hybrid identical apps for the iOS and Android platform does not necessarily mean it will be equally perceived by users on both platforms. The far ends of the plot show apps that are better perceived on one platform but not the other.

3.3 Threats to Validity

The Hybrid apps detected in our study are a subset of all the possible hybrid apps. Our study uses an existing dataset of 80,000 to identify 15,512 hybrid apps.

In terms of representativeness, our identified hybrid app dataset contains apps
from all the categories available on the Google Play store. 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 [58], making the findings of our study reproducible.

### 3.4 Conclusions

In this chapter, we present the first study of hybrid mobile apps. We mined 15,512 hybrid apps, identified the CPTs they used and analyzed their attributes like the number of ratings, stars and downloads. We indirectly compared the success of the CPTs by measuring how the app is perceived by users. We compared the AUR of hybrid apps to native apps of the same category. Additionally, we compared the AUR of the same hybrid app on the Android and iOS platforms.
Figure 3.6: AUR scores for 1400 hybrid app-pairs. Each pair of diamond(iOS) and square(Android) dots represents an app. The solid and dashed lines show the trend of AUR across the apps.
Chapter 4

Related Work

Many studies have been conducted recently to mine and analyze app store content. Most studies, however, have focused on one platform only. Our work, on the other hand, aims at characterizing the differences in mobile app-pairs across two different platforms. To the best of our knowledge, this is the first work to report a large-scale study targeting iOS and Android app-pairs. This chapter presents an informal survey of the recent works in the mobile apps domain.

4.1 Cross Platform

Malavolta et al. [75] conducted a study to investigate hybrid mobile apps in the Google Play Store. To find hybrid apps, the authors downloaded the top 500 apps from each app category and developed a tool which inspects app binaries to determine if an app is hybrid. Using this technique they were able to identify 445 hybrid apps and found Apache Cordova [2] and Appcelerator Titanium [10] to be the most common cross platform frameworks. They found jQuery, jQuery Mobile and Json2 to be the most common third-party web libraries and found that some hybrid apps use MVC frameworks such as AngularJS [43] and Backbone [64]. Finally they found that hybrid and native apps are rated similarly by end users. This work is the closest to our empirical study on hybrid apps. We expand on their work in a few ways. First, our dataset is 33 times larger and includes 15,512 hybrid apps. Second, we propose a novel metric which combines the ratings and stars to measure the aggregated user-perceived ratings. Third, since our dataset contains the Android and iOS versions of the apps, we are able to conduct cross platform
analysis.

Malavolta et al. [74] extended their work by mining and classifying user reviews to understand users’ perspective of hybrid apps. Their results indicate that hybrid and native apps are balanced in terms of performance, with the exception of apps which require a closer interaction with the Android platform in which native outperformed hybrid ones. They found that hybrid apps are more prone to bugs and they attribute that due to the lack testing frameworks for cross-platform tools and hybrid apps.

Heitkötter et al. [56] conducted an experiment to evaluate various cross-platform development approaches. The authors compared Web apps, PhoneGap apps [3], Titanium apps [10] and native apps by examining a set of proposed criteria such as licenses, costs, look & feel, supported platforms and application performance. The results of their experiment indicate that the Titanium framework provides the best solution, if a native-like UI is desired and only the iOS and Android platforms need to be supported. However, if the UI requirements are flexible and more platforms need to be supported, PhoneGap is the better option. The authors found that cross platform frameworks are mature enough to develop apps that can compete with native ones.

Along the same lines, Ciman et al. [25] compared four cross-platform frameworks, MoSync [82], Titanium, jQuery Mobile [98] and Phonegap with an emphasis on the development of apps with animations. They developed an animation-intensive game using each of these four frameworks and found the Titanium framework to yield the best performance due to its native support for animations and transitions.

Angulo et al. [8] compared the user experience (UX) of two versions of the same app, one developed natively and one developed using the Titanium cross platform framework. The authors conducted a user study with 37 participants and found that users were able to complete tasks slightly faster on the native version of the app. Furthermore, they found that 71% of the participants in the iOS Titanium version of the app agreed that it behaves like a typical iOS app while 91% agreed that the Android Titanium version behaves like a native app. The authors measured user satisfaction with the System Usability Scale (SUS) [19] and found that the Titanium versions of the app scored 82.79% while the native version scored
In 2016 Willocx et al. [108] conducted a study to compare the performance of native apps and apps created using cross-platform frameworks. To compare the performance, the authors used an open source app called PropertyCross [26], which has a native implementation along with various implementations using cross-platform frameworks. Using the Android, iOS and Windows Phone native implementations along with ten cross-platform frameworks’ implementations the authors measured response times, CPU, memory, disk and battery usage. Their results indicate that Javascript-based frameworks are the most CPU intensive and have the slowest launch times. Additionally, they found that the performance of a cross-platform app is strongly correlated with the targeted platform.

### 4.2 Reviews

Cen et al. [20] utilized user reviews to assess the security risk of mobile apps. They proposed a two step approach, in which they first mine an app’s user reviews and label each review into categories related to security/privacy. They use the labeled reviews to compute a risk score for each app and then rank the apps based on their risk scores. An evaluation on two datasets has shown that their technique was able to outperform other metrics for ranking app security risk.

Gao et al. [39] proposed AR-Tracker, a framework to mine user reviews without manual human labeling which also traces the changes of user reviews over time. They compared their tool to AR-Miner [23] and were able to achieve similar results.

Along the same lines, Gu et al. [48] proposed SUR-Miner, a framework to summarize app user reviews. The framework classifies reviews into five categories and then uses two interactive visualizations to present the results. The framework achieved an f-measure of 81% and a developer survey showed that 88% agree with the tool’s findings.

In 2015 Gomez et al. [40] conducted an empirical study to mine buggy apps by examining the correlation between the permission an app uses and user reviews. The authors used the mined data to build app checkers which can predict whether an app could be buggy.
A tool, DIVERSE was developed by Guzman et al. [51] to accept developer queries and retrieve related user reviews. The authors conducted a controlled experiment and found that DIVERSE can help developers save time when analyzing user reviews and planning their next releases.

Khalid et al. [65] argued for the usefulness of app reviews for crowdsourcing by analyzing a crowdsourcing reference model. Their findings indicate that app reviews can be used for crowdsourcing and help solve a few problems such as feature requests, recommendations for developers and users and error reporting.

Similarly, Maalej et al. [72] proposed several techniques to classify app reviews into four types: bug reports, feature requests, user experiences, and ratings. Using multiple binary classifiers, as opposed to a single multiclass classifier; the authors were able to achieve a precision and recall for all four classes ranging from 71% to 97%. Furthermore, they found that review classification techniques can be enhanced by incorporating metadata such as tense, length of review and star rating.

Gao et al. [38] designed a framework PAID, to prioritize app issues for developers and help them decide on problems such as which bugs to fix or what features to add. The framework operates by tracking and classifying user comments over the release versions of the app. The authors evaluated their technique on 18 apps with 117 app versions and the results show that PAID was able to predict issues that matched the official changelogs with high accuracy.

Similarly, in 2016 Villarroel et al. [105] proposed CLAP, to automatically categorize and cluster related reviews into bug reports and new features. CLAP uses this information to make recommendations to developers on what to include/fix in their next app version. They conducted a user study and found that CLAP can accurately identify bugs and feature requests to help developers with release planning.

Liang et al. [67] examined the effect of user reviews on app sales. They used a multifacet sentiment analysis (MFSA) [68] approach to analyze user reviews and found that reviews on service quality have the strongest effect on app sales.

In 2015 Mcilroy et al. [78] conducted an empirical study on user reviews from 20 apps and found that 30% of the reviews express more than one concern about the app. Based on these findings they proposed an automated approach to assign multiple labels to reviews with a precision of 66% and recall of 65%. They demon-
strated the usefulness of their approach in app comparison and app store overview. Finally they proposed a technique to detect anomalous apps and evaluated it on 12,000 apps.

Mcilroy et al. [77] expanded their work on app reviews and examined the value of responding to user reviews. They examined responses to reviews of 10,713 apps and found out that most developers do not respond to reviews. However, they found that after a developer response, 38.7% of users increase their rating with a median of 20%.

Palomba et al. [86] proposed CRISTAL to track informative user reviews to changes in the apps source code. They conducted an evaluation on 100 open source apps and found that 49% of the requests in user reviews were implemented in new versions of the app. They found a positive relationship between implementing the user requests in reviews and the apps overall success measured in terms of ratings, thus confirming Mcilroy’s previous results.

Panichella et al. [87] combined machine learning techniques; natural language processing, text analysis and sentiment analysis to classify reviews into categories relevant to software maintenance and evolution. They evaluated their framework on 1,421 manually labeled reviews and were able to achieve a precision and recall of 75%. Their results indicate that the combined use of machine learning techniques achieves better results than using each technique separately.

Park et al. [89] conducted a study on mobile app retrieval. They proposed a topic model (AppLDA) that represents apps using the topics in user reviews and app descriptions. An evaluation on 1,385,607 reviews from 43,041 apps shows that AppLDA significantly outperforms traditional retrieval techniques.

4.3 Security

Avdienko et al. [13] mined apps for abnormal usage of sensitive data. They proposed MUDFLOW which extracts flow data information from apps in order to train a malware classifier. They used 2,950 apps to train the classifier and were able to identify 86.4% of the malicious apps and 90.1% of the apps which leak sensitive data.

Deng et al. [31] combined static and dynamic analysis to develop iRiS, a sys-
tem to detect iOS apps which use private APIs and access sensitive user information, thus violating Apple’s terms of service. Out of 2019 apps, the authors detected 146 apps which use private APIs that access sensitive user information, such as the device’s serial number.

In 2015 Huang et al. [57] developed SUPOR, a system which uses static analysis to detect UI elements that prompt the user for entry of sensitive data, such as user credentials, finance or medical data. They combined SUPOR with existing static taint analysis tools to detect apps which leak private information. They evaluated the system on 16,000 apps mined from the Google Play store and achieved an average precision and recall of 97.3% and found that 355 apps leak private user information.

Chen et al. [21] developed MassVet, which can identify malicious apps under 10 seconds and with a low false detection rate. However, Unlike other techniques which rely on heavyweight static/dynamic analysis of the app; MassVet operates by comparing the submitted app to other similar apps. A large scale evaluation on 1,165,847 apps mined from Google Play store identifies 127,429 apps as malicious.

In 2016 Dash et al. [28] proposed a purely dynamic analysis technique to classify Android malware into families of related malware. They used a hybrid approach which combines the traditional Support Vector Machines [97] classification method with Conformal Prediction [96]. Through an evaluation on 5,560 apps they were able to achieve a high accuracy of 94%.

FUSION, a bug reporting framework based on static and dynamic analysis of the app was developed by Moran et al. [81]. Taking the event driven nature of apps into account, FUSION generates reproduction steps for bugs. A user study involving 28 participants showed that FUSION allows developers to accurately reproduce bugs.

Ma et al. [71] developed a system to detect malicious apps by comparing their description and runtime behavior. Their work improves on the popular CHABADA [47] work by combining semi-supervised learning and active learning and making use of both known benign and malicious apps to detect malicious apps. An evaluation on 22,555 apps showed an f-measure of 96.02% which was 209.6% higher than CHABADA.

A large scale study to detect vulnerabilities in apps which contain web con-
tent was conducted by Mutchler et al. [83]. The authors leveraged a variety of techniques to detect vulnerabilities such as loading unsafe web content, exposing sensitive API calls and mishandling certificate errors. Inspecting 998,286 Google Play apps which contain web content reveals that 28% contain at least one vulnerability.

In 2015 Schutte et al. [92] developed ConDroid, which performs concolic execution of Android apps to observe behavior such as network traffic or dynamic code loading. Using ConDroid on a dataset of 10,000 apps, revealed that 172 suffered from a logic bomb vulnerability.

Along the same lines, Fratantonio et al. [36] implemented TriggerScope to detect logic bombs in Android apps. In addition to symbolic execution, TriggerScope uses a new static analysis technique called trigger analysis to detect hidden app behavior and report it to the user. TriggerScope was evaluated on 9,582 apps from the Google Play Store and achieved a 100% detection rate and discovering 2 previously undisclosed vulnerabilities.

Vigneri et al. [104] built a system to characterize the network behavior of Android applications and identify network communication that could leak private user information such as user tracking, spyware or excessive ad usage. Using a set of 5,000 apps from the Google Play store, the authors discovered that a large number of popular applications download excessive amounts of advertisements and that some apps establish network connections with malicious websites.

Yang et al. [111] developed AppContext which detects malicious apps by using static analysis to examine the events and conditions that cause an app to exhibit security-sensitive behaviors. Running AppContext on a dataset of 835 apps from the Google Play Store correctly identifies 192 malicious apps with a precision of 87.7% and a recall of 95%.

Zhang et al. [112] proposed the DescribeMe system to improve the security awareness of app users. The system uses a combination of static analysis techniques to automatically identify security issues with an app and output a human readable description of the results. To prove the effectiveness of their system the authors conducted a user study using Amazon's Mechanical Turk (MTurk) [6] on 100 apps and found that while their system reduced readability by 4% it reduced the downloads of malicious apps by 39%.
A system for early detection of spam mobile apps was developed by Seneviratne et al. [95]. The authors manually labeled a dataset of removed apps, and using a set of heuristics that explain why an app was removed, it was estimated that 35% of the apps were removed because they were spam. Using this data a classifier was trained to automatically detect spam apps. The classifier achieved an accuracy of 95% and estimated that 2.7% of the apps in the author’s dataset were spam.

4.4 Feature

A technique to help users find apps faster on their smartphone was developed by Lulu et al. [70]. Their technique uses app descriptions along with information from the web to represent apps based on their functionality. A user study with 40 participants revealed that the proposed technique allowed users to find apps faster and provided a more logical grouping of apps.

Berardi et al. [15] used machine learning to automatically classify apps based on their metadata. The authors used app names, descriptions, ratings and app sizes to classify apps into one of 50 categories. The classifier used the Support Vector Machine algorithm and was trained using a set of 5,993 manually labeled Android apps. The classifier achieved an f-measure of 89.5%.

A system xRank, to help advertisers find better target users was developed by He et al. [54]. As opposed to traditional approaches like using AdWords or AdSense, xRank targets users based on the apps downloaded on their phones. xRank was trained using a set of 122,875 apps from the Huawei App Store and information about 20,169,033 users. xRank was shown to improve the accuracy of various marketing tasks compared to traditional marketing approaches.

In 2015 Tian et al. [101] conducted a study to understand the differences between high and low rated apps. The authors used a dataset of 1,492 Android apps and measured the correlation between an app’s rating and its data such as size, code complexity, library dependence and UI complexity. Their findings show that the size of an app, number of promotional images, and target SDK version affect app ratings the most.

Wang et al. [107] developed a text mining approach to identify how sensitive
data is used in Android apps. Their approach decompiles apps to Java source code and searches for uses of sensitive permissions, then extracts multiple kinds of features from the code. A classifier is built using these features and evaluated in the context of geolocation and the user’s contacts permissions. Using the classifier on a set of 622 apps reveals that it can accurately, with an average of 89.5%, infer how sensitive permissions are used.

4.5 Testing

In 2015 Boushehrinejadmoradi et al. [18] developed a technique based on differential testing to detect inconsistencies and test cross platform app development frameworks. Through the use of random test generation tools, the authors executed the tests on the source and target platforms and examined the results to identify inconsistent behavior. They implemented their technique in a tool called X-Checker and applied it to the popular cross platform framework Xamarin. Their tool identified 47 bugs/inconsistencies, 12 of which have been fixed after being reported by the authors.

Along the same lines, Erfani et al. [34] proposed an automated technique to detect inconsistencies in native apps across the iOS and Android platforms. The technique uses a graph based approach to compare the dynamically extracted models of the apps and detect inconsistent behavior such as missing functionality or different data presentation. The technique was implemented through a prototype tool called CHECKCAMP and evaluated on a set of 14 industrial and open source apps. CHECKCAMP identified 54 inconsistencies with an f-measure of 92% on average. A limitation of the tool however, is the high false-positive rate in the reported data inconsistencies.

Meng et al. [79] developed an Android Testing Toolkit (ATT) to help the development of testing and analysis tools. The toolkit combines tools and APIs for device management, event generation, system profiling and program instrumentation. The authors demonstrated the use of their technique by reimplementing 3 different testing techniques using ATT.

Choudhary et al. [24] conducted a study to compare the various test input generation tools for Android. They used criteria such as code coverage, fault detection
and ease of use to compare the tools. Their experiments show that Monkey [42] remains the best existing test input generation tool, providing the best coverage, highest fault detection rate and the best support for various versions of Android.

Mao et al. [76] proposed SAPIENZ, a multi-objective approach which combines random fuzzing, systematic and search-based exploration to test Android apps. SAPIENZ aims to improve code coverage, fault detection rate and execution time. SAPIENZ was applied to the top 1,000 Google Play apps and revealed 558 new crashes, 14 of which have been reported to and fixed by the developers. Additionally, SAPIENZ significantly outperformed state of the art tools such as Monkey [42] and Dynodroid [73] in 70% of the experiments related to code coverage and fault detection.

Lu et al. [69] developed PRADA, an approach based on mining large-scale usage data to help developers prioritize Android device models to test their apps on. PRADA tries to predict the expected usage of a new app based on the usage data from a set of existing similar apps. PRADA was evaluated by using app browsing time on a set of 200 apps from the Wandoujia [106] app store covering 3.86 million users and 14.71 thousand devices. The results indicate that PRADA is able to accurately prioritize test devices 75% of the time.

Gomez et al. [41] developed a crowd sourcing based approach called MoTiF, to help developers reproduce app crashes experienced by end users. MoTiF monitors the execution traces from Android devices and identifies various crash patterns; it uses these crash patterns to generate test suites which the developers can use to reproduce the crashes quickly. MoTiF was evaluated on 5 Android apps, and successfully generated test suites which reproduced the bugs in 4 out of the 5 apps.

Deng et al. [30] adapted the popular structural testing method, mutation testing [29] to test Android apps. They defined mutation operators unique to Android apps such as replacing event handlers or deleting buttons and used the operators to develop a prototype to generate, inject and execute the mutated apps. The prototype was used to test a small Android app and successfully demonstrated the feasibility of applying mutation testing to Android apps.

A technique AGRippin, that improves on Model Based Testing techniques of Android apps was proposed by Amalfitano et al. [5]. AGRippin uses a combination of genetic and hill climbing techniques to generate test suites. An evaluation on 5
open source Android apps showed that AGRippin produced test suites which were more efficient and effective than ones produced by Model Based techniques.

In 2015 Bielik et al. [16] developed a system to detect race conditions and concurrency bugs in Android applications. The system uses execution traces to build Happens-Before graphs which are used to detect race conditions. Applying the system on 8 open-source apps revealed 15 bugs such as displaying old information and crashes after the user stops using the application. Some of these bugs were reported to and fixed by the developers. Additionally, the proposed system outperformed other race condition detection tools in terms of bug detection and usability.
Chapter 5

Conclusions and Future Work

In the first part of this thesis, we present an empirical study on cross-platform mobile apps or app-pairs. We mine 80,169 iOS and Android app-pairs and compare their app-store attributes. We find that Android apps are more expensive and receive more ratings. Price fluctuations across the platforms are common and reasons for that include, different monetizing strategies, offering different features and different costs and efforts to maintain the app. Further, we find that more than 80% of the top-rated apps are cross platform and reasons for apps only existing on one platform include, lack of resources, platform restrictions and revenue per platform. Additionally we propose a metric to measure an app’s aggregated user-perceived ratings which combines ratings and stars. We find that in 95% of the app-pairs, the Android version is perceived better by users. Reasons for apps being perceived better on one platform include the method used to implement that app, timing (release/update) and first impressions and app store support and promotional opportunities. Finally, we build three automated classifiers to classify 1.7M reviews to understand how the concerns and complaints of users vary across platforms. We find that iOS apps suffer more from critical and post update problems while Android apps exhibit more problems related to app features.

In the second part of this thesis, we expand our work on cross-platform apps by examining a special category known as hybrid apps which are built using Cross Platform Tools (CPTs). We mine 15,512 hybrid apps and analyze their attributes like the number of ratings, stars and number of downloads. We find that apps implemented using the PhoneGap CPT make up 68% of the hybrid apps and are best perceived by end-users. However, apps created using the Adobe Air CPT,
Despite having the smallest market share, have the highest number of ratings and downloads. Further, we compare hybrid and native apps and find that they are perceived similarly by end users. Finally, we look at how hybrid apps are perceived across the Android and iOS platforms and find that using a CPT does not ensure that the app will be equally perceived by users.

5.1 Future Work

As for future work, we plan to explore our data further to gain insights into the behavior of apps across different platforms. For instance, we plan to analyze the release dates of app-pairs to understand which platform developers will target first when they release a new app. Additionally, the testing and analysis of apps across multiple platforms could be explored. While our recent study [33] 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 [84]. Finally, while we combined the stars and ratings to measure how an app is perceived by users, in the future we will explore other methods such as using the source code of an app coupled with its problematic user reviews to measure reliability and user perception.

As for hybrid apps we plan to mine user reviews and automatically classify them to better understand users’ perception of such apps. Other interesting avenues of research include verifying whether CPTs output apps with the correct and desired behavior. Finally, it would be valuable to survey the developers of hybrid apps and get their feedback and comments on the subject.
Bibliography


doi:10.1145/2568225.2568276. URL 
http://doi.acm.org/10.1145/2568225.2568276.→ pages 49

doi:10.1109/ASE.2015.57. URL 
http://dx.doi.org/10.1109/ASE.2015.57.→ pages 46

doi:10.1109/SANER.2015.7081842.→ pages 13

[50] E. Guzman and W. Maalej. How do users like this feature? a fine grained sentiment analysis of app reviews. In *Requirements Engineering*


[58] Hybrid Apps Study: Toolset and dataset. 


64
doi:10.1109/MIS.2010.75  → pages [47]


doi:10.1109/COMPSAC.2015.93  → pages [49]


doi:10.1109/MobServ.2015.14  → pages [45]


Scikit Learn: Machine Learning in Python. 


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 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), ICSME ’15, pages


