Native Apps vs. Framework-Generated Apps

Shantanu Agara[1],Vinay Chavan[2], Parth Desai[3], Aditya Gupta[4], Ruby Salazar[5]

CS 540 - Advanced Topics in Software Engineering

Department of Computer Science, University of Illinois at Chicago

Abstract— **Native applications are developed exclusively for a specific platform. These apps are developed in a language compatible with the platform. Apple, for instance, prefers Objective C and Swift for iOS while Google prefers Java and Kotlin for Android. In order to make the mobile app development process quicker, easier, more convenient and less expensive, led to the appearance of cross-platform frameworks. A framework-generated cross-platform mobile application is an application that runs on multiple mobile platforms (Android, iOS). Several frameworks, like React Native developed by Facebook and Xamarin developed by Microsoft aim to provide smooth iOS and Android apps developed simultaneously using JavaScript (React Native) or C# (Xamarin) without compromising on the UX and UI experience.**

In this paper, we compare the quality of apps produced by such frameworks as Xamarin and React Native vs. apps produced natively in iOS and Android from the end-users' perspective. The basis for the comparison will be the ratings and the reviews available in the Apple and the Google markets.

I. Introduction

Millions of weekly visitors to app stores (with Google Play Store and Apple App Store as clear market dominators) show how people are using their mobiles now more frequently than the web and thus mobile apps are projected to hit $188.9 billion in revenue by 2020. However, code written for one mobile platform (e.g., Kotlin/Java code of an Android app) cannot be used on another (e.g., Objective-C/Swift of an iOS app), making the development and maintenance of native apps for multiple platforms one of the major technical challenges affecting the mobile development community. This has led top industry companies (e.g, Microsoft, Facebook Inc.) to propose development frameworks such as Xamarin (Microsoft) and React Native (Facebook) with the goal of creating cross-platform mobile apps developed simultaneously using JavaScript (React Native) or C# (Xamarin) without compromising on the UX and UI experience. The advantages of these frameworks include reducing development cost and making development twice as fast due to the absence of necessity for native developers of Android and iOS to develop apps and maintain them separately. Despite these advantages, framework-generated apps also suffer from a number of shortcomings such as restricted access to hardware features and a decrease in performance. This has led to a Native vs. Cross-platform contentious debate among developers. In order to see how both developed applications compare, this paper aims to analyze the quality of apps produced by such frameworks as Xamarin and React Native vs. apps produced natively in iOS and Android from the end-users' perspective. The basis for the comparison will be the ratings and the reviews available in the Apple and the Google markets for a total of 213 mobile applications.

The main findings of our study are: (i) end users value native apps more than apps developed using cross-platform frameworks for both iOS and Android platforms based on sentiment and statistical analysis performed on user ratings and (ii) the keyword analysis provides valuable information regarding issues inside each app framework, which could yield useful information regarding future feature design.

The rest of the paper is organized as follows. Section II presents how the dataset of apps was created. Section III discusses how the apps' metadata and reviews were mined from the play stores and preprocessed. Section IV discusses review analysis and results and Section V concludes the paper.

II. Dataset Creation

In this section, we describe the methods that were used in order to compile a framework-generated app list and native app list for both mobile platforms. This section will also cover what further methods were employed to detect Xamarin and React Native frameworks and finally, this section will end with an overview of the dataset.

A. App List Curation

We started off with creating a list of potential applications for each of the following categories:

I. Android Native apps
II. Android React Native apps
III. Android Xamarin apps
IV. iOS Native apps
V. iOS React Native apps and
VI. iOS Xamarin apps

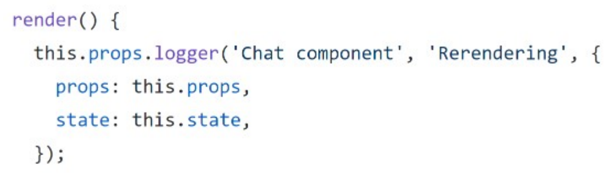
For I, we targeted apps built by Google LLC., with an assumption that all of the apps built by Google would be built natively. We also found a few apps on the Android developers - stories page [2]. For II and V, the React Native apps for both Android and iOS; we referred to a GitHub repository which maintains a curated list of open-source React Native apps [3]. We continued building this list with the help of React Native's showcase page which displays a few popular apps built using their framework [4]. To build an initial list of Xamarin apps, we referred to Xamarin's official website which lists apps developed using their framework [5]. For IV, iOS Native apps, we targeted the apps on the app store that use Swift, a study conducted by Ryan Olson[6], an iOS engineer, helped us create an initial list.

B. Framework Detection

To confirm if an Android app is implemented using the Xamarin framework, we analyzed the structure of every potential Xamarin app's apk file. Android application packages have a .apk file extension. Xamarin.Android application packages are ZIP containers and they follow the same structure and layout as native Android packages, with the following additions [7]: The application assemblies (containing Intermediate Language) are stored uncompressed inside the assemblies folder. Whenever the app is launched from the Release builds, the apk is mapped into the process startup using mmap() and the assemblies are loaded from the memory. This ensures faster app startup, as assemblies do not need to be extracted prior to execution and they are JIT compiled into native machine code whenever the app is launched.

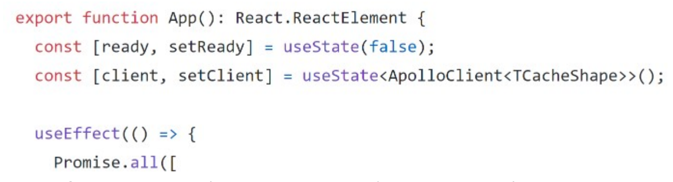
Native libraries having the Mono runtime are stored in a separate folder within the apk . In order to run a Xamarin.Android application for the targeted Android architectures such as armeabi, armeabi-v7a and x86 it must contain appropriate native libraries, specific to target platforms.

To determine whether the app is developed using React Native framework, we examined the source code of all potential React Native apps and found a bundle named com/facebook/react in all the apps developed using React Native framework. We also found javascript code for components, views, props, state and render methods in the source code confirming the apps were built with React Native. Fig. 1 and 2 show a code snippet of the render method and App component found in one of the source codes.



```
render() {
  this.props.logger('Chat component', 'Rerendering', {
    props: this.props,
    state: this.state,
  });
```

Fig. 1 Code snippet of render method found in react folder.



```
export function App(): React.ReactElement {
  const [ready, setReady] = useState(false);
  const [client, setClient] = useState<ApolloClient<TCacheShape>>();

  useEffect(() => {
    Promise.all([
```

Fig. 2 Code snippet of App component found in react folder.

The source code of all native Android and iOS apps that we considered for our analysis did not have any traces of a framework.

C. Dataset Overview

In this subsection we will provide a brief overview of the collected number of apps belonging to both mobile platforms along with the native and framework aggregates as well as show what categories these apps belonged to. As seen in Fig. 1, a total of 213 apps will be part of the dataset. Of these 213 apps, 159 apps are Android and 51 are iOS apps. The breakdown of how many belong to each framework and how many are native can also be seen in Fig. 3.

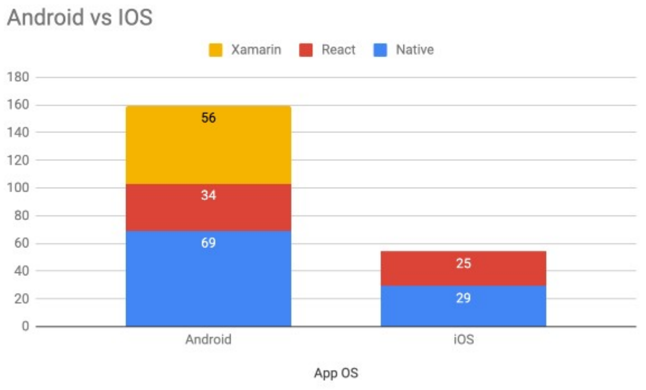


Fig. 3 A stacked bar graph displaying Android vs. iOS number of apps with each colour signifying native or framework generated.

It's important to note that no Xamarin iOS apps were considered in this study because the initial list of apps on iOS built with Xamarin was small and it was difficult to confirm if these apps actually used Xamarin. We wanted to make sure that our study does not include any false positives and hence, we made a decision to drop iOS-based Xamarin apps. In Fig. 4 we can see a further breakdown that consists of only Android apps that shows what app categories make up this dataset. The total number of categories for Android apps is 35. The most popular categories for Android native are Productivity (14) and Tools (6). The most popular categories for React native are Communications (4) and the most popular categories for Xamarin are Food & Drink (19) and Lifestyle (9).

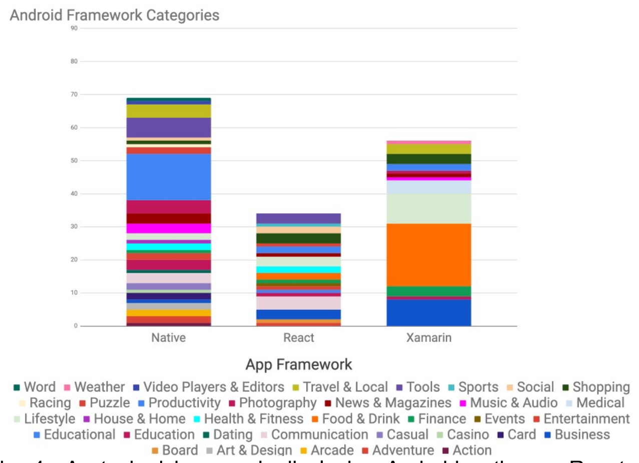


Fig. 4 A stacked bar graph displaying Android native vs React vs Xamarin number of apps with each colour signifying the app category.

Fig. 5 provides a similar breakdown belonging to iOS apps. The total number of categories for iOS apps is 16. The most popular categories for iOS native are Photos & Video (5), Social Networking (4), Travel (4) and News (4). The most popular categories for React native are Business (4) and Social Networking (4).

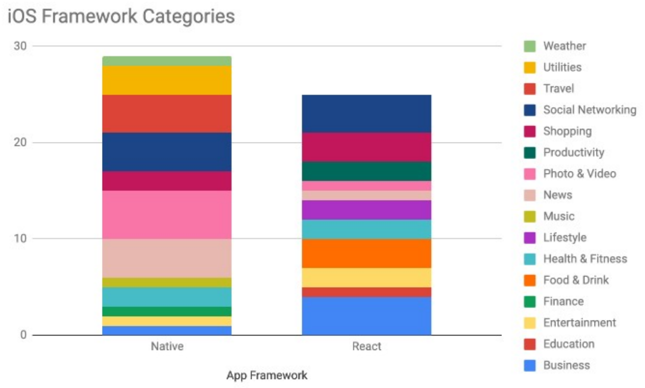


Fig. 5 A stacked bar graph displaying the number of apps for iOS native vs React with each colour signifying the app category.

III. MINING AND PREPROCESSING

In this section, we describe the methods that were used in order to scrape metadata and reviews from both app stores as well as describe the preprocessing techniques used to prepare the data for analysis.

A. *Mining*

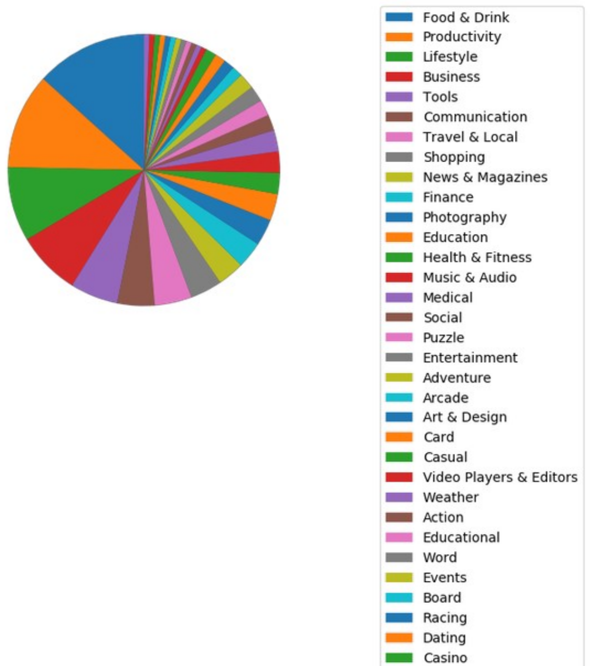
Once the app list was curated and confirmed we formed our dataset by scraping the Google Play Store and App Store. Both of these app markets render their reviews dynamically. Therefore, for data extraction, we used python scraping algorithms with Selenium and BeautifulSoup libraries. We used the Selenium Chrome Driver in order to automate “Show More Reviews” button clicks and scrolling. The targeted amount of reviews we wanted to collect was up to 5000 reviews unless they had less number of reviews. Once the page loaded with the preferred number of reviews for a particular app, we used BeautifulSoup which provides ways to navigate, search and modify the parse tree based on position, tag name, attributes, css classes, etc. Both app store scrapers are publicly available on GitHub[1]. The metadata that was collected for an app in the Google Play Store is the following: (i) name, (ii) category, (iii) overall rating valued from one to five, (iv) number of reviews, (v) number of installs, and (vi) cost (although all apps analyzed ended up being free). The metadata that was collected for an app in the App Store is the same as the Google Play Store except the number of installs was not extracted as this information is not provided by the App Store. For each review in both app markets the following was extracted: (i) user rating valued from one to five, (ii) date, and (iii) full review text. The data extracted was then merged to a Pandas DataFrame and placed in a corresponding csv file for further preprocessing and analysis.

B. *Preprocessing*

The data was processed for mining through 2 main steps 1) Text Processing and 2) Cleaning.

Text Processing involved processing the actual reviews, which are found in the review_body columns of each app’s review_appname.csv. The data was first tokenized. Tokenization is the process of chopping up each word into pieces, which are known as *tokens*. While tokenizing, we removed all punctuations, as well as special characters, including XML tags, which were left behind during mining. The next step was stemming, which involved removing the ends of words. Stemming was performed to make sure that words that are derivatives are not considered separately. For example, the words field and fielded have the same origin ‘field’ and are considered identical for our purposes. The ending ‘ed’ is removed via stemming. The next step is stopword removal. This step is performed to remove words which do not provide context to a review. The nltk library provides a predefined list of stopwords, which we used to remove almost all stopwords that affected our analysis. Then, we added more stopwords, including the app name and words that were unique to our data.

The second step was recategorization. Recategorization in our context refers to redefining the app categories, which are provided by the app store providers as well as the number of downloads. The reason for recategorization by app categories was that one, there were multiple categories which had a single app, particularly the game categories and two, that the categories for the iOS and the Android app stores were not identical. These two factors made the analysis which was carried out very biased. The solution was to recategorize the apps into 4 categories. The choice of apps per category was based on their functionality.



1 https://github.com/rsalaz9/android_review_crawler/tree/master/scappers

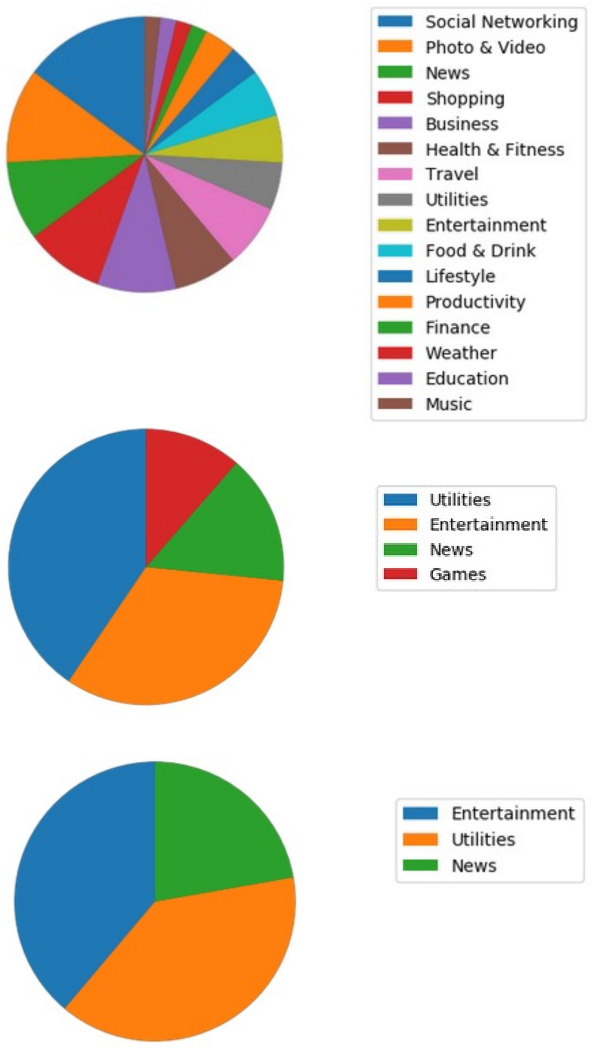
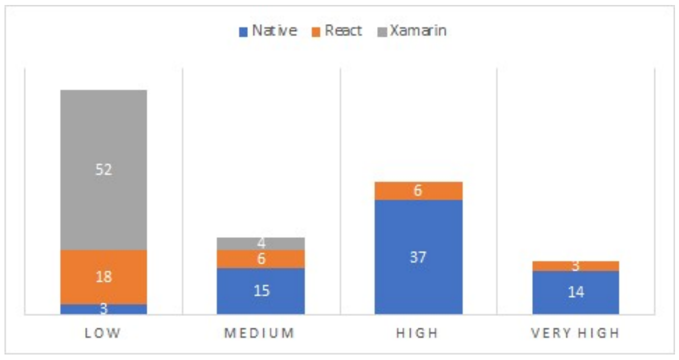


Fig. 6 Pie Charts displaying the percentage of apps divided by categories (a) Android apps before recategorization (b)iOS apps before recategorization © Android apps after recategorization and (d) iOS apps after recategorization

The categorization was carried out in the following order for Android: Utilities includes the categories of Communication, Photography, Medical, Music & Audio, Photography, Shopping, Tools, Travel & Local, Productivity, Art & Design, Video Players & Editors, House & Home. Entertainment includes the default categories of Entertainment, Food & Drink, Health & Fitness, Lifestyle, Social, Sports, Dating, Education. The news category includes the Business, Events, Finance, News & Magazines and Weather categories. Then the games category includes the Adventure, Board, Educational, Puzzle, Racing, Arcade, Action, Card, Casino, Word and Casual categories. In the case of iOS apps, there were only three categories, since no iOS gaming apps were mined for reviews. Here, the new Utilities category includes the Photo & Video, Travel, Shopping, Music and Productivity apps. The entertainment category includes the Social Networking, Health & Fitness, Entertainment, Food & Drink, Lifestyle and Education categories. And the news category included the News, Weather, Finance and Business categories.

Based on Fig 6, it is evident that the categories are significantly balanced out. However, the gaming category was missing in iOS apps because no gaming app data was mined.

Fig. 7 Number of Android apps for each framework by popularity



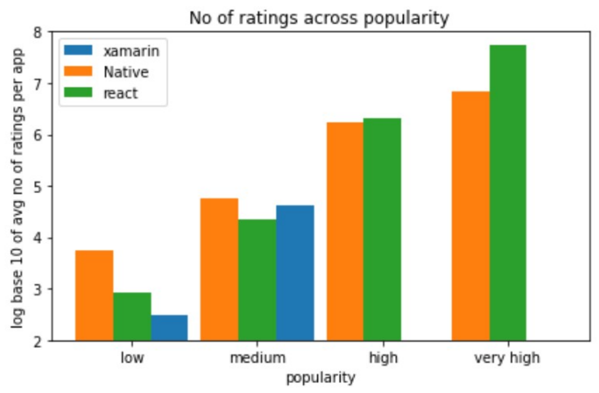
The apps were also recategorized based on popularity, which is the number of downloads (see Fig. 7). However, the data was generated only for Android since the App Store(iOS) does not provide the number of downloads. The categories are low(100-100,000+), medium(500,000-1,000,000+), high(10,000,000-100,000,000+) and very high (500,000,000-5,000,000,000+). This ensures that the very popular apps, which normally have significantly more resources devoted to them, are not compared with less popular and more niche apps.

IV. ANALYSIS

In this section, we describe the methods that were used to analyze app metadata and reviews in order to be able to compare the quality of apps produced by such frameworks as Xamarin and React Native vs. apps produced natively in iOS and Android.

A. Sentiment Analysis

Sentiment analysis is the study of emotions in data. For the app data, we used the app ratings to generate the positive, negative and neutral emotions for each reviewer. Ratings above 3 were categorized as positive, those below 3 as negative, and the ratings at 3 were categorized as neutral.



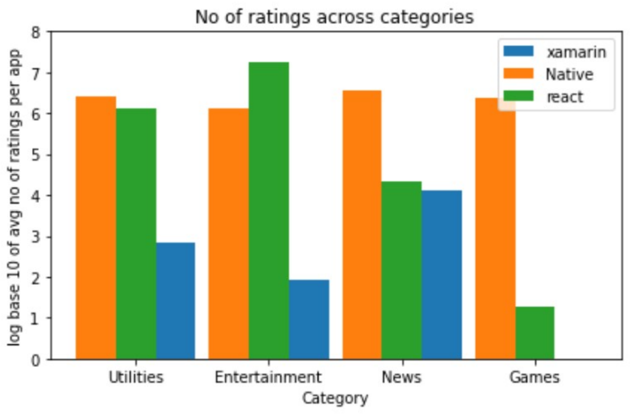


Fig 8. Distribution of apps across different categories and popularity for Android

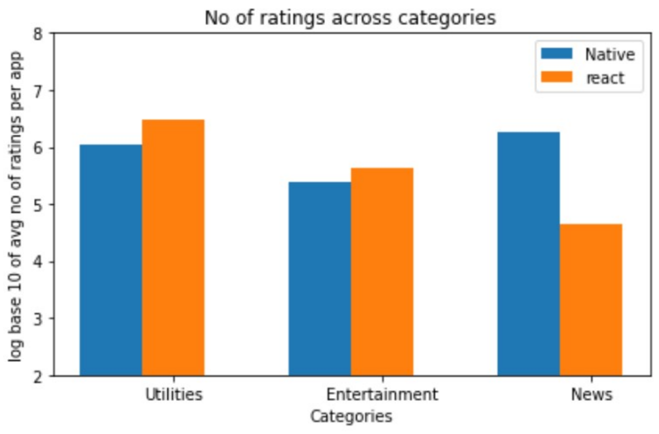


Fig 9. Distribution of iOS apps across different categories

Due to the differences in the app development, as well as other myriad reasons(such as economic prosperity of users for each ecosystem) which would make the analysis unviable, the analysis for both iOS and Android were carried out separately. The analysis was carried out based on both categories (Fig. 8) and popularity (Fig. 9). The observations were as follows:

1. Xamarin had a significantly higher ratio of negative reviews.
2. The ratio of negative reviews was higher for React in comparison to Native for all popularity levels.
3. React seems to generate a larger number of neutral reviews across all categories except in case of games.
4. In contrast with the React apps for Android, their iOS versions for all categories possessed a significantly higher number of negative reviews across all categories.
5. In iOS apps, there was a clear trend of native apps having a higher ratio of positive reviews, as compared to the React versions.

Certain observations can be attributed to smaller sample sizes. The data in case of the medium popularity for Xamarin does not refute the first observation, due to only 4 apps being mined for this category. The same can be said for observation 3, as only 4 React apps of the games category were mined.

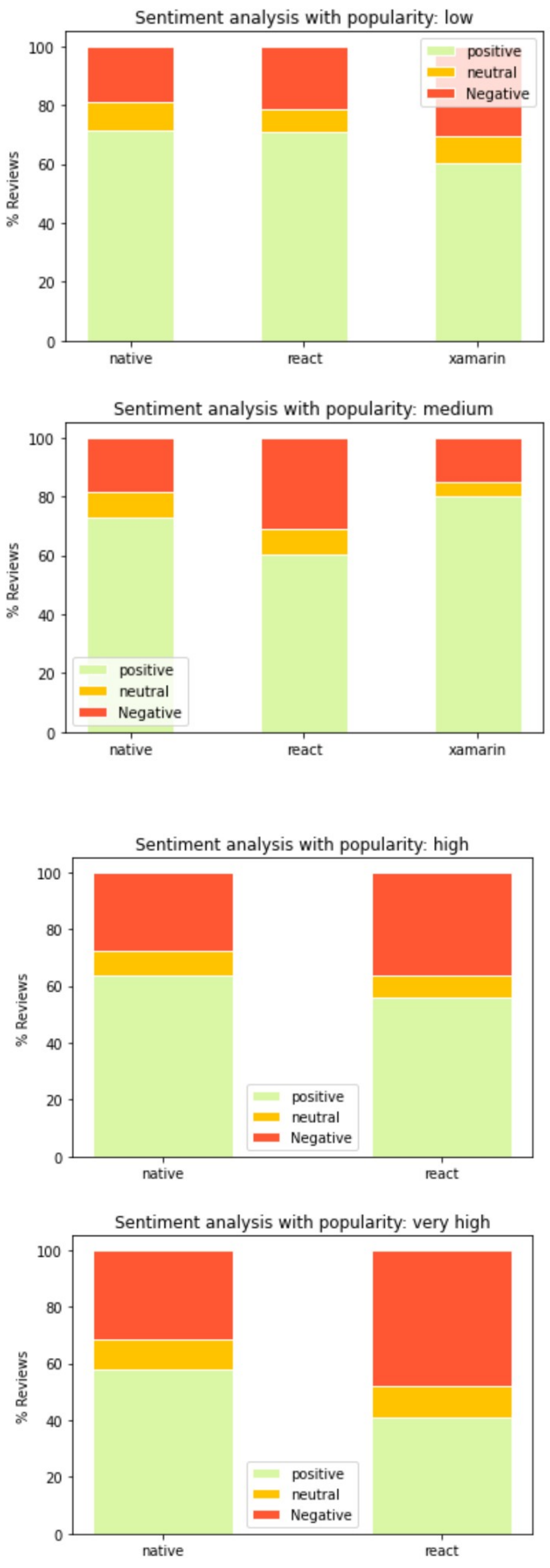


Fig. 10 Sentiment analysis of all Android apps based on (a) low (b) medium (c) high and (d) very high popularity

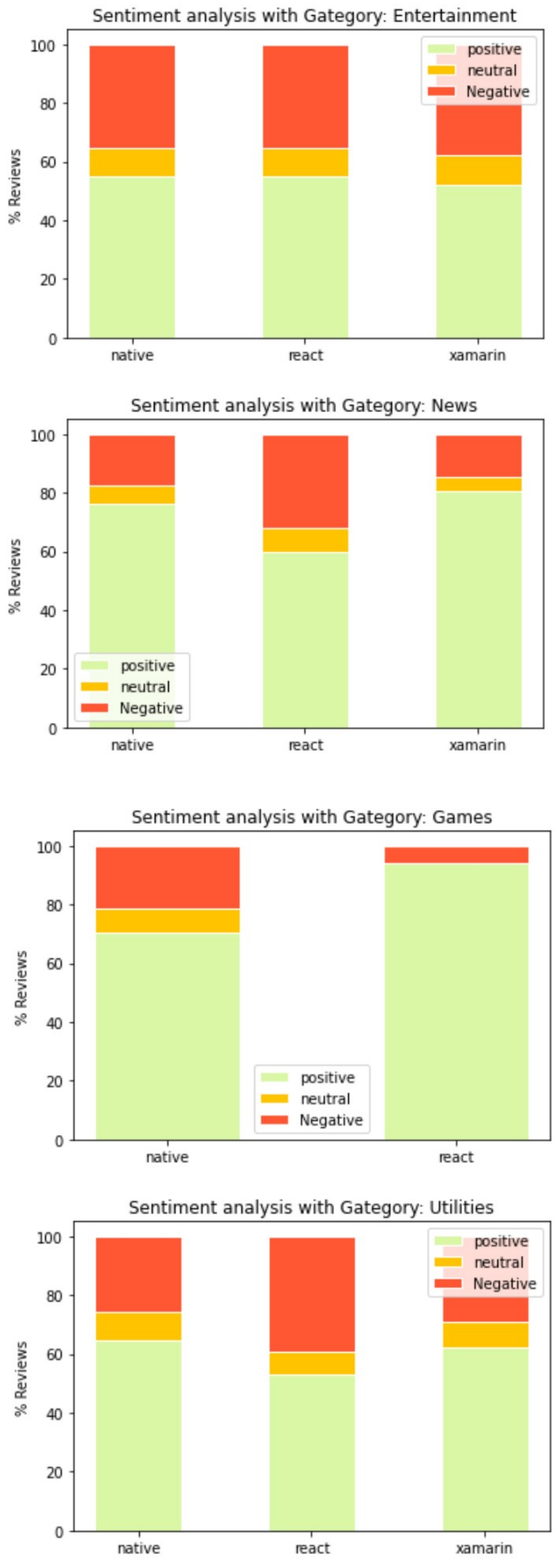


Fig. 11 Sentiment analysis of all Android apps based on (a) entertainment (b) news (c) games and (d) utilities categories

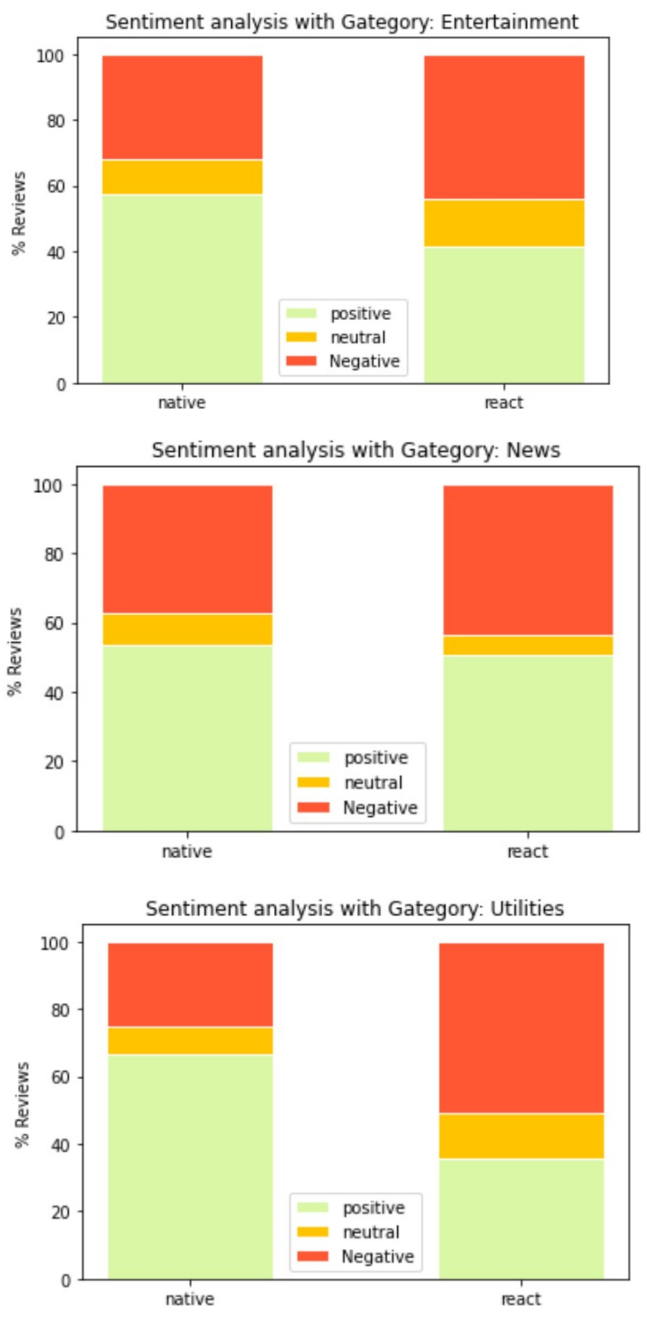


Fig. 12 Sentiment analysis of all iOS apps based on (a) entertainment (b) news (c) utilities categories

B. Rating Analysis

Rating Analysis was carried out using statistical methods. For this, we chose to calculate the mean, standard deviation and the quartile values for the overall ratings, as well as for the categories and popularity indexes for each app ecosystem.

Popularity	All		
Framework	**Native**	**React Native**	**Xamarin**
Count	300607	73611	26040
Std Dev	1.583	1.706	1.430
Mean	3.676	3.335	4.076
25%	2	1	4
50%	4	4	5
75%	5	5	5

(a)

Popularity	Low			Medium			High		very high	
Framework	**Native**	**React Native**	**Xamarin**	**Native**	**React Native**	**Xamarin**	**Native**	**React Native**	**Native**	**React**
Count	530	4215	4535	67157	17340	21505	179320	40136	53600	11920
Std Dev	1.427	1.480	1.656	1.425	1.658	1.351	1.611	1.718	1.630	1.698
Mean	3.951	3.867	3.547	3.955	3.534	4.187	3.635	3.330	3.458	2.873
25%	3	3	2	3	2	4	2	1	2	1
50%	5	5	4	5	4	5	4	4	4	3
75%	5	5	5	5	5	5	5	5	5	5

(b)

Category	Entertainment			Utilities			News			Games	
Framework	**Native**	**React Native**	**Xamarin**	**Native**	**React Native**	**Xamari**	**Native**	**React Native**	**Xamarin**	**Native**	**React**
Count	53957	24568	1283	180425	31156	3114	14524	17853	21643	51701	34
Std Dev	1.682	1.675	1.706	1.571	1.755	1.649	1.410	1.648	1.347	1.499	1.015
Mean	3.326	3.341	3.284	3.696	3.245	3.617	4.048	3.482	4.189	3.865	4.618
25%	1	1	1	2	1	2	4	2	4	3	5
50%	4	4	4	4	4	5	5	4	5	5	5
75%	5	5	5	5	5	5	5	5	5	5	5

(c)

Fig. 13. Android rating statistics across (a) all apps, (b) popularity levels and (c) app categories

Popularity	All	
Framework	**Native**	**React Native**
Count	115555	14301
Std Dev	1.644	1.716
Mean	3.570	2.940
25%	2	1
50%	4	3
75%	5	5

(a)

Category	Entertainment		Utilities		News	
Framework	**Native**	**React Native**	**Native**	**React Native**	**Native**	**React Native**

Count	28284	2381	58605	5340	28666	6580
Std Dev	1.639	1.608	1.590	1.652	1.714	1.791
Mean	3.457	2.928	3.746	2.772	3.321	3.079
25%	2	1	2	1	1	1
50%	4	3	5	2	4	4
75%	5	5	5	5	5	5

(b)

Fig. 14 iOS rating statistics across (a) all apps and (b) app categories

The results for these were as follows:

1. On average, Xamarin had the highest average ratings and the lowest deviation amongst the three Android frameworks, but the average number of ratings per app was also the lowest (465 vs 2230 for react and 4356 for native).
2. The quartile scores for React Native were the worst for both the 25 and the 50th quartile.
3. Xamarin had the worse 25th and 50th percentile scores for the low popularity, where 92.8% of its apps belong.
4. In the high and very high popularity, Native apps have better ratings than the React framework for Android.
5. The average scores are the best for the Games category in Android, whereas it is the Utilities category in iOS.
6. Across all categories, apps developed using the native framework have better average ratings than the React framework. However, no such clear observation could be drawn for Android.

C. Keyword Analysis

For keyword analysis, the primary metric is the frequency of keywords. We have utilized word clouds to provide visual representations of this metric.

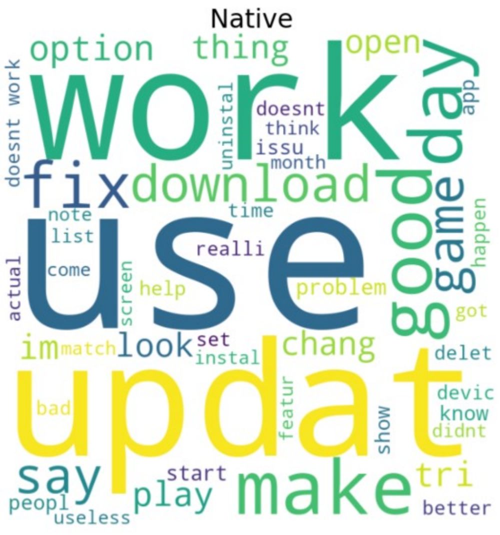
The goal of this analysis was to find specific attributes about apps developed using each framework and obtain a more concrete comparison between them. Specifically, we wanted to analyse what features users like most and what features they are facing issues with.

While analysing the positive reviews, we observed that users would use keywords like 'great app", "love the app", "very useful". There were only a few mentions of specific features they really liked. On the other hand, in negative reviews, users would specifically mention the issues they were facing.

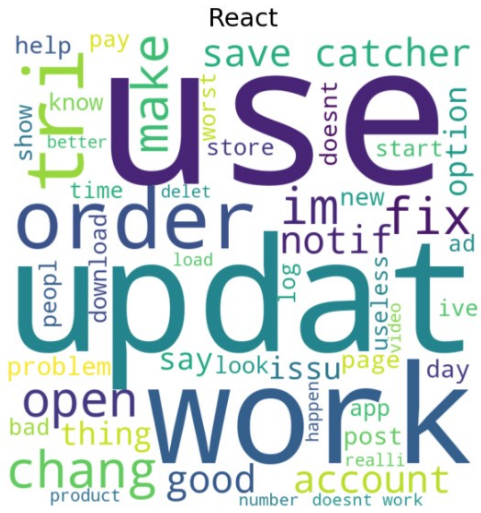
Across all frameworks, the most used words consistently are rooted with "use", "work", update", "fix". This was because the mention of phrases like "useless app", "doesn't work", "fix this issue" was most common. Also, there were frequent complaints about buggy updates.

Using word clouds, which were generated using the nltk platform, we are quickly able to trace the features which generated the most reactions amongst the reviewers. Taking the example of Xamarin (Fig. 15c), we find that the word 'login' occurs very frequently. There were 614 mentions of the word account in the 4600 negative reviews. This root when traced back to the original reviews, reveals that there are a not-insignificant number of users who faced problems while logging in while using these apps.

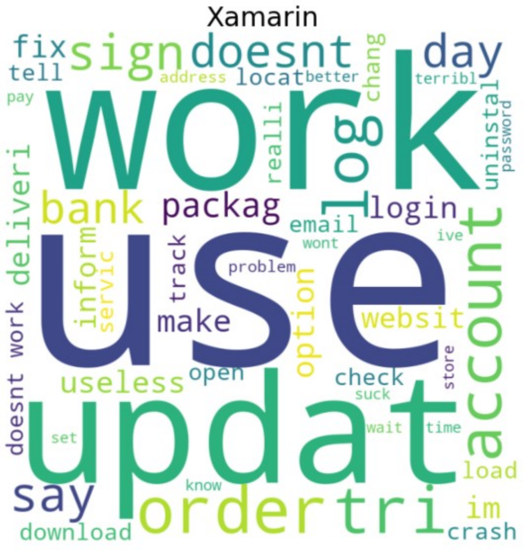
Among the reviews of the Android apps developed using the React Native framework (Fig 15b), we found a frequent mention of the word notification. Out of 26500 negative reviews of all apps we analyzed, words with root "notif" was mentioned 2408 times. Users mention issues like not getting timely notification, or their notification preferences not having effect.



(a)

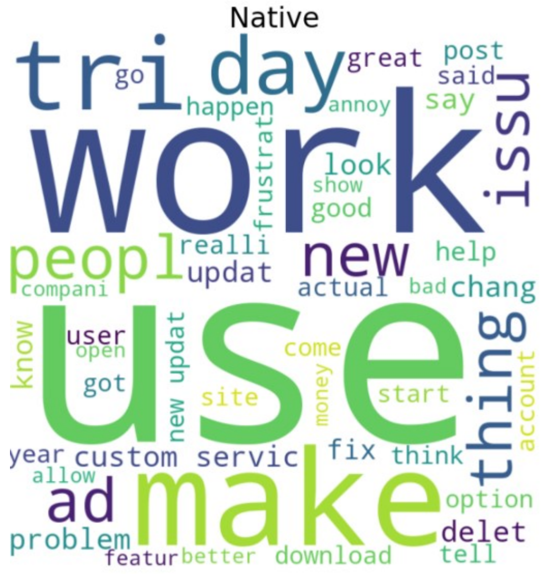


(b)

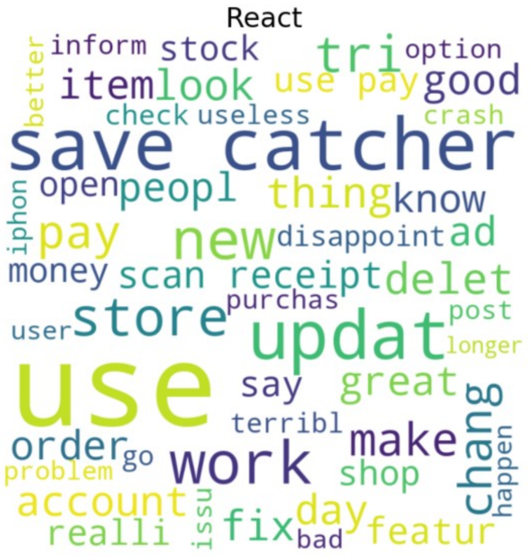


(c)

Fig.15 Word Clouds generated from the reviews of all (a) native (b) React (c) Xamarin apps for the Android platform.



(a)



(b)

Fig.16 Word Clouds generated from the reviews of all (a) native (b) React apps for the iOS platform.

V. CONCLUSIONS

For both app ecosystems, the statistical and sentiment analysis indicate that apps developed using the native frameworks have higher user ratings than those developed using hybrid frameworks. Furthermore, the keyword analysis provides valuable information regarding issues inside each app framework, which could yield useful information regarding future feature design.

REFERENCES

[1] Developer stories, latest apps. Available: https://developer.Android.com/stories/apps

[2] A curated list of open-source React Native apps. Available: https://github.com/ReactNativeNews/React-Native-Apps

[3] Who’s using React Native? Available: https://reactnative.dev/showcase

[4] Xamarin Customer Showcase: https://dotnet.microsoft.com/apps/Xamarin/customers

[5] Introduction to Information Retrieval By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze. Available: https://nlp.stanford.edu/IR-book/information-retrieval-book.html

[6] Top 100 App Store apps. Available: https://gist.github.com/ryanolsonk/e39829708f3b3e52b4e6#file-top100-csv

[7] Xamarin apk structure:

https://docs.microsoft.com/en-us/Xamarin/Android/internals/architecture