The Use of Cross-Platform Frameworks for Google Play Store Apps

Andreas Biørn-Hansen and Tor-Morten Grønli
Kristiania University College,
Dep. Technology, Oslo, Norway
andreas@flytit.no; tor-morten.gronli@kristiania.no

Tim A. Majchrzak
University of Agder,
Kristiansand, Norway
timam@uia.no

Hermann Kaindl
TU Wien, Inst. of Computer Tech.,
Vienna, Austria
hermann.kaindl@tuwien.ac.at

Gheorghita Ghinea
Dept. of Computer Science, Brunel University London
Uxbridge, London, United Kingdom
George.Ghinea@brunel.ac.uk

Abstract

In this paper, we describe the harnessing and analyses of a large sample ($n = 661\,705$) of Android apps and associated metadata available on the Google Play Store. The analyses and scrutiny are in the context of cross-platform mobile development, as we report on the technologies used to develop apps for the Android ecosystem. Specifically, we quantify the use of 13 technical frameworks for cross-platform development, identify their distribution across Google Play Store categories, present an overview of framework usage from 2008 to 2019, app file size (`.apk` size), and lastly discuss our findings in the context of current industry trends and directions. Our findings indicate that cross-platform apps account for approximately 15% ($n = 99\,304$) of the dataset, and that all overarching development approaches are present.

Keywords: Cross-platform development, mobile apps, Google Play Store, mobile computing

1. Introduction

For more than a decade, the Google Play Store has been the leading (and official) software application repository for the Android mobile operating system. For industry, practitioners and hobbyist developers, having a product (an app) available on the app marketplaces means participating in a multi-trillion dollar economy [1]. The growth in users across demographic and geographic groups [2] suggest continuous economical growth for all stakeholders. In addition to acting as the official repository for mobile apps for the Android platform and the go-to marketplace for end-users, the Google Play Store is also a massive repository of binary files and metadata for researchers, enabling scrutiny of real-world technical solutions.

In this study, we investigate the use and presence of cross-platform development frameworks in mobile apps available on the Google Play Store. Our motivation is to report on the state of "cross-platform development" in published Android apps, an umbrella term covering a wide array of development approaches and technical frameworks and tools [3, 4]. Such technologies attempt to bridge the heterogenous nature of the leading mobile platforms, Android and iOS, by providing a uniform set of abstractions and APIs to developers [5]. The overarching idea in cross-platform development is to write the majority of an app's code in a platform-agnostic fashion, abstracting from the platforms, allowing a single codebase to run on multiple platforms with little to no platform-specific code [6].

While the study described here focuses on the Android ecosystem through analysis of published apps on the Google Play Store marketplace, apps are also developed for the Apple ecosystem and their Apple App Store marketplace. These two leading mobile platforms are heterogenous in terms of app development programming languages and user interface (UI) design guidelines. Hence, developing an app publishable to both platforms requires two separate codebases [7]. Inherently, all code related to business logic and user interface design would require a complete rewrite for each supported platform. According to the literature, mobile software developers identify reuse of code across multiple platforms to be a challenge, along with fragmentation of user interface design standards and programming languages [8].

The native development approach describes the platform providers' intended set of practices and procedures for developing platform-specific apps, i.e., apps that cannot be executed on another platform than what it was developed for. Exampels for the cross-platform spectrum are hybrid, interpreted, and cross-compiled [3]. The hybrid approach relies on an embedded Web browser in a native app to display HTML, CSS and JavaScript-based user interfaces, and for communicating with underlying platform and

URI: https://hdl.handle.net/10125/80276
978-0-9981331-5-7

hardware features. The interpreted approach typically ships a JavaScript engine along with a native app, allowing to render native user interface components. Lastly, the cross-compiled approach is more of a catch-all category for frameworks not fitting into the previously mentioned approaches, where a single codebase might be compiled into platform-specific binaries, or frameworks renders user interfaces without intercepting with native UI components.

We formed three research questions (RQs) related to cross-platform mobile development, `.apk` analysis and metadata from the Google Play Store:

Research Question 1 (RQ_1): "What is the distribution of cross-platform development frameworks on the Google Play Store?"

Research Question 2 (RQ_2): "What is the distribution of cross-platform framework usage across Google Play Store categories?"

Research Question 3 (RQ_3): "How has the usage of cross-platform frameworks for deployed apps changed over more than a decade?"

The remainder of this paper is structured as follows. In Section 2, we present related studies. In Section 3, we discuss our data gathering and processing pipeline, alongside our framework identification algorithm. In Section 4, we present our findings, and discuss the results. In Section 5, we conclude the study.

2. Related Work

The rapid development of the mobile computing field, and the enabling factors provided through it has led to considerable academic and industry interest in research and development of mobile apps. Studies from recent years have, for instance, focused on the impact of a programming language on app quality in a study on Java versus Kotlin for development of Android apps [9], assessing and comparing code smells in iOS and Android apps [10], and providing guidelines for ensuring development of energy-efficient apps [11]. In the mobile computing sub-field of cross-platform mobile development, there is an established body of knowledge, as surveyed by [3].

Although app store analysis in our context is relatively new, descriptive studies on cross-platform mobile development have been published and cited frequently since the earliest frameworks were released. For instance, the study by Heitkötter et al. [7] shaped much of the research to follow due to actionable suggestions for future research, and due to having laid the foundation for framework comparisons. Also Charland and LeRoux [12] discussed early on the possibilities of Web-based mobile apps as a potential challenger to native apps, and placed cross-platform development on the research agenda.

From a user perspective, Mercado et al. [13] leverage user reviews mined from the Google Play Store and the Apple App Store to investigate the impact on user-perceived quality of apps developed using cross-platform frameworks. Malavolta et al. [14] also scrutinize the user perspective in their study, although looking more specifically at user perception of hybrid mobile apps, also based on reviews mined from the Google Play Store. Similarly to our study, they provide an overview of cross-platform framework usage and distribution across the Play Store categories, although with a more limited dataset of $n = 11\,917$ apps and a partially different set of cross-platform frameworks (Cordova, Titanium, PhoneGap, Sencha, Kivy, Rho Mobile, UIU and Enyo).

From a broader perspective, Viennot et al. [15] conducted a measurement study of the Google Play Store, describing their Web crawler architecture along with results on such as native library usage, advertisement libraries, graphics engine usage, and briefly on cross-platform framework usage. Their results indicate that 9.2% of non-popular apps are built using cross-platform technologies, while 2.6% of popular apps are built the same way. In our study, we follow up and elaborate on cross-platform framework usage, using Viennot et al.'s [15] study as a historical reference (2013), but we do not differentiate between popular and non-popular apps.

3. Research Design

Our study is exploratory, with analyses based on quantitative data. The procedure for conducting the analysis is threefold: First, we detail on the process of gathering the data. Then, we describe the framework identification algorithm. Lastly, we revisit the RQs named in the introduction and present a hypothesis.

3.1. Data Gathering

To gain access to the Android installation files (`.apk` 's), we were granted access to the AndroZoo dataset made available by researchers from the University of Luxembourg (as described in detail by Allix et al. [16]). Gathering data through AndroZoo instead of the Google Play Store increases the reproducibility and availability of the dataset and results, as Google Play Store mining is unnecessarily complex and time-consuming [16] when more accessible services exist, especially if these were developed specifically to benefit researchers. Due to the structure and

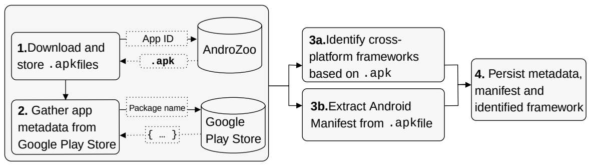
App ID
AndroZoo
1.Download and store .apkfiles
.apk
2. Gather app metadata from Google Play Store
Package name
Google Play Store
{ ... }
3a.Identify cross-platform frameworks based on .apk
3b.Extract Android Manifest from .apkfile
4. Persist metadata, manifest and identified framework

Figure 1. Process of data gathering Store

services provided by AndroZoo, researchers interested in replicating or continuing our current work may study our data gathering process and open-sourced dataset as discussed below. We filtered the AndroZoo dataset on apps originally deployed and made available in the Google Play Store, as the dataset also contains apps from various other sources including, but not limited to PlayDrone, AppChina and F-Droid. A filter for removing duplicate apps was applied, as AndroZoo may host multiple versions of the same app. The dataset consisting of $n = 661\,705$ compiled (`.apk`) apps requires 9.2TB of hard drive space. We have open-sourced the identification algorithm and dataset with extracted data (`.csv`) through GitHub[1], along with necessary instructions. After downloading the `.apk` files, we gathered associated metadata from the Google Play Store and piped the `.apk` installation files through our cross-platform framework identification algorithm (described in Section 3.2). The whole data gathering process is illustrated in Figure 1.

3.2. Framework Identification Algorithm

To identify the usage of a cross-platform framework – or the absence of one – based on an `.apk` file, we developed a pattern matching algorithm looking for specific string-based values, files, folders, etc. As the technical landscape of cross-platform development is vast, with one study identifying more than 60 individual frameworks [3], it was necessary to choose a specific set of frameworks for which we can create condition-based rules. In Table 1, we list the 13 frameworks that we created rules for to analyze the `.apk` files. All four major development approaches are accounted for in the list. In terms of sample size, Martin et al. [17] found that studies on app store analyses wherein apps are analysed have a median sample size of 1 679 and a mean of 44 807 apps. Our sample size of $n = 661\,705$ `.apk` files puts our study in the upper percentile of assessed apps [17].

Our framework identification algorithm is inspired by Ali and Mesbah's [18] study on characterizing hybrid apps based on an algorithm accounting for three frameworks, which we in our study extend to 13 frameworks across three development approaches. To identify the various frameworks, our algorithm traverse

[1]https://github.com/andreasbhansen/phd-thesis-contributions/

Table 1. Technologies included in our algorithm

Technology	Appr.	Language	Release	Ruleset
Native*	Native	Numerous	2007/08	-
Adobe Air	Interp.	Numerous	2008	(1)
NativeScript	Interpr.	JavaScript	2014	(3)
Qt (Mobile)	Interpr.	C++/QML	2013	(3)
Fuse	Interpr.	JavaScript/C#	2012	(3)
Titanium	Interpr.	JavaScript	2009	(3)
React Native	Interpr.	JavaScript	2015	(2 & 3)
Weex	Interpr.	JavaScript	2016	(3)
Codename One	Cross-c.	Java/Kotlin	2012	(3)
Flutter	Cross-c.	Dart	2017	(3)
Xamarin	Cross-c.	C#	2011	(3)
Capacitor	Hybrid	JavaScript	2017	(2 & 3)
Cordova/PhoneGap	Hybrid	JavaScript	2009	(2 & 3)
Ionic *(Cordova-based)*	Hybrid	JavaScript	2013	(2 & 3)

a directory containing N `.apk` files, searching for information for and in meta tags, manifest files and and binary files. The following three rule sets were developed to identify the various frameworks.

1. Combination of string search in `AndroidManifest.xml` and file/folder search in the `.apk` 's `/assets/` folder.
2. File/folder search in the `.apk` 's `/assets/` folder.
3. String search in `AndroidManifest.xml`.

We validated the algorithm by downloading and testing `.apk` files associated with apps from each framework's showcase page. The version of the algorithm used to extract the dataset presented in this study managed to correctly identify all the showcase apps we tested with.

To categorize apps as belonging to the native development approach, our requirement was that the `.apk` did not match any of the rules specified for the frameworks included. We describe these as *Unidentified*, as our algorithm could neither identify the use of a framework nor the lack thereof.

3.3. Hypothesis

In addition to the three research questions given above, one hypothesis was also formed. The hypothesis is based on compiled app (`.apk`) file size, and how the use of a cross-platform framework impacts this non-functional requirement. Corbalán et al. [19] investigate the file size impact in an experiment using self-developed Android apps, indicating great size variance depending on which technology and framework was used. The native development approach generated the smallest `.apk` files, while the NativeScript framework generated files more than an order of magnitude larger for the same type of app. We build on and extend their work by investigating file sizes in our $n = 661\,705$ `.apk` dataset. The importance of keeping the compiled binary `.apk` size as small as possible is stressed by the Google Play Store team's own research, reported by Tolomei [20]. Their findings indicate a 1% decrease in app downloads per 6MB increase in `.apk` size. Additionally, apps beyond 100MB in size are additionally prone to see a cancellation of the Play Store download by an increase of 30% compared to apps less than 100MB in size. Based on Corbalán et al.'s [19] work, the importance of small `.apk` sizes as stressed by Google Play Store, and as cross-platform frameworks typically include additional interpreters and runtimes in the packaged `.apk` files, we formulate our hypothesis as follows.

Hypothesis 1 (H_1): "Apps developed using the native approach generate smaller `.apk` files than those using cross-platform development frameworks."

4. Findings and Discussion

In the following, we summarize our findings before discussing them. The subsections to follow draw from the overview presented in Figure 2.

4.1. Framework Distribution of Apps

We first revisit RQ_1 and ask for the distribution of development frameworks. Although industry outlets have set their focus on more recent frameworks including React Native and Flutter (e.g., [21, 22]), Figure 2 shows indications of numerous frameworks having considerably larger market shares of published apps on the Google Play Store as per our sample dataset. In terms of development approaches, native apps account for the majority of the analyzed apps, according to Figure 4. In numbers, the analyzed dataset consists of $n = 562\,401$ native apps ($\approx$ 85%) and $n = 99\,304$ ($\approx$ 15%) cross-platform apps. Further investigations of the development approaches revealed that each of the top three cross-platform frameworks listed in Figure 2 belongs to a separate approach, hybrid (Cordova), cross-compiled (Xamarin) and interpreted (Adobe Air), respectively. Grouping the counts of apps per development approach (see Table 1) rather than framework, we found that the hybrid approach accounts for $n = 48\,371$ apps, interpreted for $n = 27\,468$ apps, and cross-compiled for $n = 23\,484$ apps. The hybrid approach is, thus, the largest of the development approaches in terms of published apps on the Google Play Store in our large sample.

Specifically the number of hybrid apps is in rather stark contrast to Gartner's prediction from 2013 that by 2016, a majority of mobile apps will be based

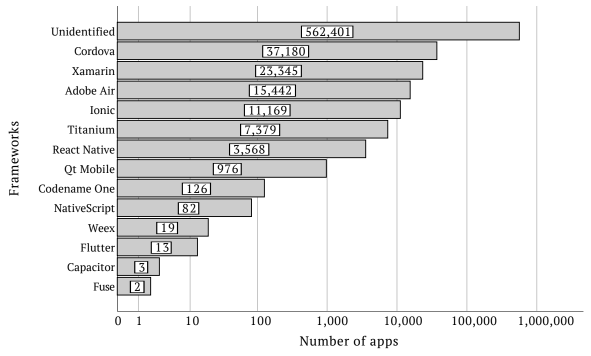

Figure 2. Log-scaled distribution of cross-platform frameworks and native development

on cross-platform technologies [23]. To the best of our knowledge, Gartner has not provided a similar prediction for 2020 or later. Nevertheless, according to related studies, these numbers are somewhat higher than what has previously been reported by Viennot et al. [15] (9.20% of non-popular apps specifically), although the number of hybrid apps identified is larger in our study (8.67%) than according to Malavolta et al. [14], where the hybrid approach accounted for 3.73% of the apps analyzed. This discrepancy could be the result of framework identification technique or sample size, as Malavolta et al. [14] analyzed $n = 11\,917$ apps, while we have analyzed $n = 661\,705$. It may also be an indication of a growing industry interest in the hybrid development approach.

4.2. Framework Distribution Across Categories

Investigating the distribution of cross-platform framework usage across app categories on Google Play Store can provide an indication of how practitioners and industry have made use of these technologies for targeting various types of apps, for instance, particular presence of cross-platform frameworks in one category, and their absence in another one. Thus, in RQ_2 we asked for this distribution.

The aggregated results are presented in Figure 3 and Table 2. The former is a log-scaled visualisation of the use of development technologies across the top four Play Store categories, the latter a contingency table displaying tabularized distribution of apps across development technologies grouped by Play Store categories. Due to the large amount of Play Store categories, we visualise only the top four Google Play Store categories based on the number of apps, regardless of the development approach, which according to Table 2 (column "$\sum$ w/ native") are Games[2] ($n = 136\,316$),

[2]The Games category is an aggregation of all the sub-categories on Play Store within gaming, for instance adventure, puzzle and strategy.

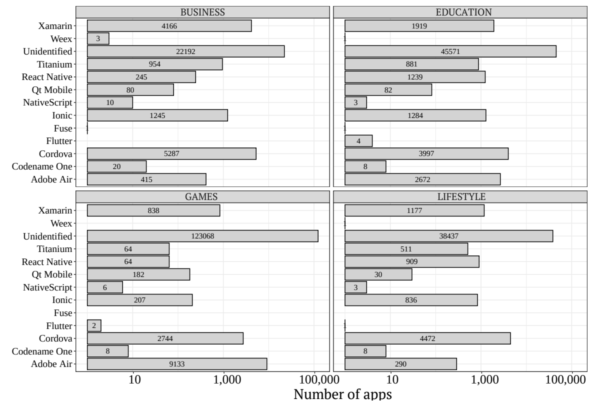

Figure 3. Log-scaled distribution of native and cross-platform frameworks

Education ($n = 57\,662$) and Lifestyle ($n = 46\,675$).

Table 2 (column "$\sum$ w/o native") shows that Games ($n = 13\,248$), Business ($n = 12\,426$) and Education ($n = 12\,091$) are the top three Play Store categories with the most identified apps. The Lifestyle category was replaced with Education when excluding native development and instead focusing on cross-platform apps, see the top four categories in Figure 3.

Of the $n = 13\,248$ games identified, Adobe Air ($n = 9\,133$) and Cordova ($n = 2\,744$) together account for 89.65% of the cross-platform apps in the category. Although Adobe Air is marketed as a cross-platform framework for both app and game development, 59.19% of the identified Adobe Air apps are in the games category, a higher percentage than any other framework. This indicates that it is a technology preferred by game developers more than by non-game app developers.

Regarding the Business category, business-to-business (B2B) apps might prioritize differently in terms of user experience and functionality than apps developed for public end-users and consumers, according to Anglin and Telerik [24]. Given that the hybrid approach, which Cordova belongs to, is frequently perceived as being close to incapable of achieving predictable native-like user experience across platforms and older devices (see, e.g., [25, 26, 27]), the prioritization of apps in the Business category may skew towards functionality over user experience, hence Cordova's popularity. According to previous research on Google Play Store category distribution of cross-platform apps, Malavolta et al. [14] had the Business category listed as the seventh most cross-platform populated Play Store category. As their study is from 2015, an explanation of this discrepancy could be a shift towards heavier investments in business digitization, leading to an increase in B2B and line of business apps, and thus in the use of Cordova. However, the results presented by Ali and Mesbah [18] in 2016 align much closer to the distribution that we present in Table 2, with the Business category as the predominant outlet for hybrid apps. According to our results, the hybrid-based Cordova framework is the single most widely used cross-platform framework within the Business category, accounting for 42.55% of cross-platform apps in this category. Note that the far right sum ($\sum$) columns in Table 2 differ from the numbers of identified cross-platform apps due to parsing and extraction issues with a small number of apps' Play Store category.

By inspecting the Google Play Store's page on top apps in the Education category, we find Massive Open Online Courses (MOOC) from MIT, Udemy and Coursera, alongside apps for Learning Management Systems (LMS), interactive language courses such as Duolingo, and note-taking apps. The strong presence of cross-platform apps in this category could indicate that cross-platform frameworks can cater to varying degrees of complexity and requirements, as the aforementioned apps range from predominantly list view-based apps (note taking) to more performance-demanding products focusing on video rendering and animated content.

Certain categories stand out due to the relatively considerable absence of cross-platform apps. Personalization – for instance, custom Android launchers, custom keyboards, and icon and wallpaper packs – is one extreme case where cross-platform frameworks account for 0.41% of the category's identified apps. Those types of apps require a higher degree of platform-specific native code than, for instance, a Business category app, as the former's predominant focus is on modifying the underlying Android system front-end. Another category lacking the presence of cross-platform apps is Photography, wherein such apps account for 2.74%. It could be argued that these need platform-specific native code for communication with platform and device features, for instance to perform face recognition and tracking with superimposed filters in real-time.

4.3. Framework Distribution Over Time

A challenge with analyzing framework distribution and usage over time stems from a side-effect of obfuscation during `.apk` compilation [28]. Certain measures and obfuscation techniques will render an app's compilation time (Dalvik executable files' creation dates, `DEX_DATE`) unusable as the date is set to the 1970s, 1980s or an arbitrary year, for instance 2001, 2046 or 2059 which were all observed in the dataset. Prior to downloading `.apk` files from the AndroZoo

Table 2. Distribution of apps per framework grouped by Play Store category

Category	Adobe Air	Capa-citor	Codename One	Cordova	Flutter	Fuse	Ionic	Native-Script	Qt Mobile	React Native	Tita-nium	Weex	Xamarin	Native*	$\sum$ w/ native	$\sum$ w/o native
ART AND DESIGN	44	0	2	61	0	0	20	0	7	5	7	0	51	3843	4040	197
AUTO AND VEHICLES	16	0	1	144	0	0	48	0	4	10	46	0	286	2483	3038	555
BEAUTY	12	0	0	86	0	0	21	0	2	15	17	0	110	1573	1836	263
BOOKS AND REFERENCE	255	0	6	1667	0	0	840	11	14	35	183	1	641	35888	39541	3653
BUSINESS	415	0	20	5287	0	1	1245	10	80	245	954	3	4166	22192	34618	12426
COMICS	36	0	0	22	0	0	17	0	0	0	1	0	21	1725	1822	97
COMMUNICATION	86	0	4	618	0	0	253	4	23	34	85	0	565	9590	11262	1672
DATING	7	0	0	118	0	0	7	0	1	2	4	0	22	412	573	161
EDUCATION	2672	0	8	3997	4	1	1284	3	82	1239	881	1	1919	45571	57662	12091
ENTERTAINMENT	697	0	5	2052	0	0	525	4	33	86	203	2	1019	33029	37655	4626
EVENTS	16	0	0	112	0	0	65	3	2	29	46	0	271	671	1215	544
FINANCE	93	0	3	1850	0	0	443	2	7	69	755	1	1237	13125	17585	4460
FOOD AND DRINK	90	0	1	766	0	0	338	1	5	36	307	0	559	5632	7735	2103
GAMES	9133	0	8	2744	2	0	207	6	182	64	64	0	838	123068	136316	13248
HEALTH AND FITNESS	182	3	7	1397	1	0	629	1	32	59	1013	0	958	14024	18306	4282
HOUSE AND HOME	16	0	0	141	0	0	49	1	8	14	28	0	141	1963	2361	398
LIBRARIES AND DEMO	15	0	0	61	0	0	29	1	7	5	19	0	60	1439	1636	197
LIFESTYLE	290	0	8	4472	1	0	836	3	30	909	511	1	1177	38437	46675	8238
MAPS AND NAVIGATION	20	0	1	649	0	0	183	1	20	25	106	1	433	6502	7941	1439
MEDICAL	109	0	4	1086	0	0	299	4	22	44	395	0	761	6685	9409	2724
MUSIC AND AUDIO	167	0	3	831	0	0	174	1	42	18	147	0	409	35705	37497	1792
NEWS AND MAGAZINES	81	0	2	791	0	0	368	3	19	74	174	1	534	15872	17919	2047
PARENTING	28	0	0	33	0	0	13	0	2	3	1	0	44	489	613	124
PERSONALIZATION	10	0	0	40	0	0	23	0	0	1	8	0	48	31290	31420	130
PHOTOGRAPHY	53	0	0	87	0	0	26	0	11	12	13	0	70	10890	11162	272
PRODUCTIVITY	181	0	13	1352	1	0	568	6	112	80	224	2	1737	15250	19526	4276
SHOPPING	29	0	5	998	0	0	374	5	4	77	168	3	952	8447	11062	2615
SOCIAL	54	0	5	658	0	0	325	1	7	77	140	0	477	7173	8917	1744
SPORTS	138	0	0	1302	0	0	351	4	16	66	190	0	949	10528	13544	3016
TOOLS	275	0	12	1260	0	0	582	4	149	78	203	1	1657	34412	38633	4221
TRAVEL AND LOCAL	142	0	6	2243	3	0	913	2	30	144	435	2	1119	17582	22621	5039
VIDEO PLAYERS	59	0	1	125	1	0	33	0	18	8	29	0	45	4209	4528	319
WEATHER	9	0	1	107	0	0	74	1	5	5	18	0	62	2442	2724	282
$\sum$	15430	3	126	37157	13	2	11162	82	976	3568	7375	19	23338	562141	661392	99251

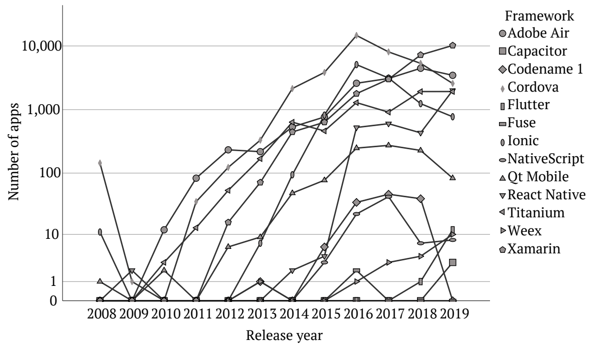

Figure 4. Log-scaled framework distribution over time from 2008 to 2019.

repository, we applied a filter to only download files with a reasonable (i.e., non-obfuscated) date between 2008 and 2019.

In Figure 4, a log-scaled distribution of framework usage between 2008 and 2019 is depicted. We excluded native apps from the figure to focus exclusively on cross-platform presence. We can thereby answer RQ_3, which asked about the usage pattern over time.

2016 was particularly interesting according to Figure 4, as all the assessed cross-platform frameworks increased in adoption that year. One possible explanation is the industry-generated hype surrounding the first release of React Native in 2015 [29], an implementation that became more stable in 2016. The increased interest in cross-platform development could have led industry practitioners to also look for alternatives to React Native, resulting in higher adoption and usage across all frameworks.

Up until 2018, our dataset indicates that Cordova was for five consecutive years the dominating cross-platform development framework. Cordova peaked in terms of deployed apps in 2016 ($n = 16\,225$). However, in 2018 Xamarin surpassed Cordova for the first time since 2011. Based on data from the Stack Overflow developer surveys from 2017 and 2018, Xamarin is more appreciated than Cordova, with a slight increase in developer appreciation for both frameworks. The Xamarin growth in 2018 according to our data could possibly be related to the Microsoft acquisition of Xamarin in 2016. The decreasing use of Cordova is reflected in the decline of desktop installs of the Cordova tool, as illustrated by the download statistics provided by the npm-stat website [30].

Based on discussions from industry outlets, Facebook's React Native has gained traction among practitioners [21]. As shown in Figure 4, however, we found an earlier implementation of the interpreted approach to still see more usage in published apps – namely the Titanium framework from Appcelerator. While React Native had a spike after its release in 2015, the Titanium framework was still found to be more popular for published apps up until 2019, when React Native saw a considerable increase in adoption, from $n = 431$ in 2018 to $n = 2\,004$ in 2019, whereas Titanium in 2019 was at $n = 1\,922$.

Figure 4 also shows that Cordova was used in 144 of the sample size apps in 2008, an outlier as compared to the other frameworks. The increase in use of Cordova has continued since then, with the exception of 2009 and 2010. Interestingly, for several of the frameworks with high volume, 2018 / 2019, shows a decline. It will be interesting to see whether this trend continues, which could be an indication of a swing in the pendulum between native apps and cross-plattform apps.

4.4. Impact on `APK` File Size

Prior to investigating the potential impact that cross-platform frameworks have on `.apk` size, we conducted a Levene's Test to check for homogeneity of variance. The test reported of statistically significant variance, thus violating the assumption of homogeneity of variance in ANOVA, at $F(13, 661691) = 318.927, p < .0005$. Due to the amount of outliers (see Figure 5), high standard deviation and sample variance of identified apps between the categories of development approaches (see Table 3), ranging from two (2) apps in the case of Fuse up to 37 180 apps in the case of Cordova, and further 562 401 unidentified/native apps, it was deemed infeasible to conduct hypothesis testing using an analysis of variance. We focus instead on reporting and interpreting results based on descriptive statistics to discuss Hypothesis 1 (H_1), and highlight mean ($\bar{x}$) `.apk` size alongside standard deviation (σ), maximum and minimum values in Table 3 accompanied by a boxplot of `.apk` size per cross-platform framework.

The mean `.apk` size generated by the various cross-platform frameworks can have immediate implications for decision making and requirements engineering, for instance regarding the likelihood of a user actually downloading an app [20]. Developing apps for storage-constrained devices requires caution and thought, and the appropriateness of an app's file size depends on factors such as primary market (e.g., Western countries versus developing regions) and significance for end-user (e.g., daily use versus one-time use). Developing an app meant for daily use may grant the developer fewer constraints in terms of file size, while an app you may only need once, for instance a public transportation app needed during a holiday, can face additional scrutiny by the end-user for being unnecessarily large.

We thus formulated H_1, suggesting that native apps have smaller `.apk` file sizes. The results presented in Table 3 and Figure 5 indicate that the vast majority

Table 3. Mean `.apk` size per framework

Technology	N	**Mean (kB)**	**SD (σ)**	**Max (kB)**	**Min (kB)**
Xamarin	23 345	34 142,45	20 579,42	198 528,01	2 510,52
Weex	19	33 439,57	26 852,28	97 842,74	930,33
Flutter	13	31 657,93	26 088,96	94 889,04	7 200,16
Titanium	7 379	28 380,40	24 104,87	102 230,71	804,60
React Native	3 568	27 239,81	14 494,94	112 823,37	3 734,31
Adobe Air	15 442	23 556,74	17 359,39	107 526,09	32,64
Capacitor	3	21 841,82	75,41	21 885,39	21 754,74
Qt Mobile	976	20 706,93	14 849,85	104 086,80	73,58
NativeScript	82	19 314,41	12 166,07	85 992,08	6 741,01
Ionic	11 169	15 273,53	14 355,10	104 716,90	315,74
Unidentified	562 401	13 776,04	17 490,73	176 575,43	3,38
Fuse	2	12 675,19	2 584,90	14 502,99	10 847,39
Cordova	37 180	12 527,55	13 151,81	106 686,63	95,72
Codename One	126	8 011,64	7 934,04	38 516,81	1 895,09
Total	*661 705*	*14 924,16*	*17 953,00*	*198 528,01*	*3,38*

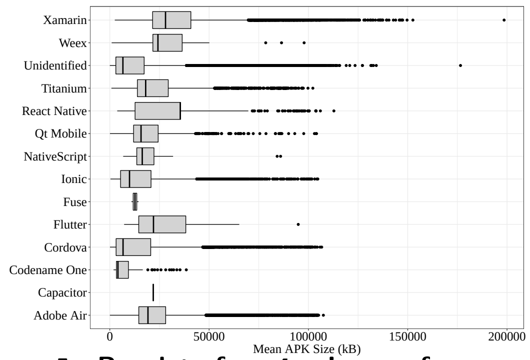

Figure 5. Boxplot of `.apk` size per framework.

of the frameworks generate apps with a higher mean `.apk` size than the native development approach. Three frameworks were found to have lower means: Fuse, Cordova/PhoneGap and Codename One.. The remaining frameworks were shown to produce apps of a higher mean file size than the native approach. Both Fuse and Codename One have a significantly lower number of identified apps used for comparison, while Cordova/PhoneGap has a more comparable size. On grounds of the results presented in Table 3 we can reject Hypothesis 1 (H_1); native apps are not inherently smaller in file size than cross-platform apps

On the opposite side of Codename One is Xamarin with the highest mean `kB` size. Looking to the overview of number of apps per framework per Play Store category in Table 2, we find that Xamarin is particularly present in the Business category ($n = 4\,166$). While it could be that apps in this category are in general larger in size, also the presence of Cordova in the Business category is substantially larger than in any other category ($n = 5\,287$), yet has a mean `kB` size close to three times smaller than the average Xamarin app. Our findings contradict those presented by Corbalán et al. [19], where three types of apps were developed in and compared across five cross-platform frameworks.

Results are more in line with those presented

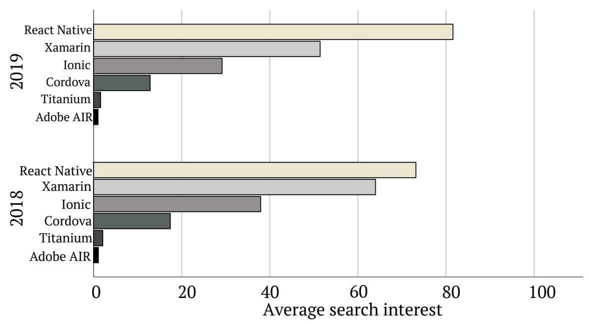

Figure 6. Search interest for selected frameworks. Data source: Google Trends

by Willocx et al. [31], who compared the native development approach to apps generated using PhoneGap and Xamarin. They found that Xamarin for Android generates apps about five times the size of the native baseline, and three times the size of PhoneGap.

4.5. Trends from an Industry Perspective

By looking to Google Trends, we can compare the search interest for cross-platform frameworks with the framework distribution over time as shown in Figure 4. By doing so, we can identify possible discrepancies in framework adoption versus what is frequently searched for on Google. Raw data from Google Trends does not include search volume, but instead a range from 0 to 100 per keyword illustrating search interest relative to the other keyword [32]. The Google Trends data was harnessed and processed using the open source Python library `gsvi` [33]. Due to the possibility of our keywords' ambiguity, for instance the meaning of Ionic which could refer to the app development framework as well as to the chemical process "ionic bonding", we filtered the results from Google Trends based on the category "Programming" (code 31).

The results are illustrated in Figure 6, in which we compare the search trends for the top six frameworks from 2018 and 2019 as displayed in Figure 4, specifically Xamarin, Ionic, Cordova, React Native, Adobe Air, and Titanium. We can see that Google Trends provide indications that React Native has been searched for more often than the other frameworks listed both in 2018 and 2019. The two other interpreted approach frameworks, i.e., the React Native alternatives Titanium and Adobe Air are to be found at the bottom of the chart. We found that React Native, Ionic and Cordova all saw an increase in search interest from 2018 to 2019, while Xamarin and Titanium both decreased. Adobe Air saw little search interest in both years.

To compare these trends to our own findings, we have tabulated and ranked the frameworks from

Table 4. Comparison of Google Trends from 2018 and 2019 (Fig. 6) to our findings (Fig. 4).

#	Google	Results 2018	Results 2019
1	React Native	Xamarin (7 219)	Xamarin (10 178)
2	Xamarin	Cordova (5 329)	Adobe Air (3 457)
3	Ionic	Adobe Air (4 461)	Cordova (2 573)
4	Cordova	Titanium (1 921)	React Native (2 004)
5	Titanium	Ionic (1 241)	Titanium (1 922)
6	Adobe Air	React Native (431)	Ionic (772)

Figure 4 and Figure 6, as can be seen in Table 4. Particularly React Native is seeing most industry interest relative to the other frameworks based on the Google Trends data, while based on our data it was the sixth most adopted framework in 2018 and fourth in 2019. This could indicate that mobile developers are interested and curious regarding new technologies, although the interest does not necessarily translate into actual adoption and use for published projects. From the survey questionnaire by Biørn-Hansen et al. [34], framework matureness was highlighted as the third (of eight) most frequently perceived challenge with cross-platform development, which could provide some context to our findings. It is also noteworthy that Adobe Air, the third (2018) and second (2019) most used framework in published Android apps according to Figure 4, does not see much search interest at all, neither during 2018 nor 2019, as seen in Figure 6. This could indicate that there is a stable community of Adobe Air developers who keep on publishing apps, while the more trendy frameworks including React Native are less stable but generated comparably more search interest.

Another industry-based source of data for gauging of framework interest, is the State of JavaScript survey from 2018 ($n > 20\,000$ respondents) [22]. In their survey, React Native is the clear outlier in terms of developer interests, with 53.2% of respondents stating they are interested in learning it. While React Native enjoys the interest of developers, NativeScript is on the other side of the presented quadrant, between the categories "Assess" and "Avoid" due to varying satisfaction. Both Cordova and Ionic are also listed in the survey and quadrant, but both frameworks get a significant percentage of votes for the survey option "Heard of it, not interested". This contrast our presented findings on framework adoption. We also note the absence of Xamarin, Adobe Air and Titanium Appcelerator from the survey. We suggest that for future surveys, similar to what was conducted by Greif [22], a more comprehensive list of technologies for participants to choose between could lead to a more detailed and accurate view of the use of technologies.

4.6. Implications

The motivation for conducting this study was to uncover the state of cross-platform mobile development and its presence in Android apps on the Google Play Store. We find that industry "framework hype" does not necessarily correspond with which technologies are adopted by practitioners. The discrepancy as discussed in Section 4.5 displays considerable differences between trends and adoption. Our results can assist in processes of decision making requiring or benefiting from empirically informed knowledge, and push towards a better understanding of how current cross-platform usage is for published apps. Developers can draw from Table 2 to better understand the use of cross-platform frameworks across the Google Play Store categories, and make informed decisions based on the state of practice. This could be opting for a hybrid-based framework for developing business applications, Adobe Air for game development projects, or Xamarin for productivity and tooling apps. Combining the state of practice with results on generated mean `.apk` file size from Section 4.4 can aid practitioners in the development of storage-aware cross-platform apps.

We can draw similar implications for research, where this overview of framework usage could assist in deciding on technologies to investigate and evaluate in future work. If the purpose of a study is to assess cross-platform mobile development frameworks that are commonly made use of in real-world apps, the overview presented in this study can provide an indication of which technologies to include. Also, as a result of our findings and analyses, several new questions arise for researchers to investigate. Specifically for investigation of cross-platform development, our open-sourced dataset containing package names and AndroZoo app IDs of identified cross-platform apps can assist in further scrutiny.

4.7. Limitations and Validity

We have focused on the Android ecosystem and how cross-platform frameworks play a role in the development of Android apps. Looking at the Apple and iOS ecosystem is a natural next step, but was out of scope for this current investigation. To the best of our knowledge, no service similar to AndroZoo exists for gathering and distributing iOS apps, so it would require an even grander undertaking to harvest and mine a similar dataset of iOS binaries and metadata.

We also acknowledge that false positive and false negative errors are likely to have occurred during data gathering using our cross-platform framework identification algorithm. While we ensured consistent identification results using a set of known cross-platform and natives apps, we cannot be sure of the percentage of such errors without conducting a significantly more complex study involving de-compilation and de-obfuscation of the `.apk` files to gain access to traversable and searchable source code.

Another possible limitation is related to the sudden decrease in published apps in 2009 and 2010, as depicted in Figure 4, which could be the case of missing data or challenge related to the identification algorithm. For certain cross-platform frameworks, for instance early versions of Cordova and PhoneGap (same technology, different distributions), finding `.apk` files from over a decade ago which have not been updated since, in order to validate the algorithm, was deemed infeasible. For these cases, we took to the open source software repository GitHub to find pre-2010 Cordova and PhoneGap app projects. These code bases were used as the foundation for extending the identification algorithm's pre-defined rules to also be able to account for early framework versions.

Finally, the count of unidentified apps is a limitation, although the lack of identifiability is a finding in itself.

5. Conclusion

We investigated patterns and trends of cross-platform framework usage in published apps in the Google Play Store, the Android app marketplace. Our findings indicate that cross-platform apps account for approximately 15% of the dataset. We found that the hybrid development approach has for years been the most popular path for developing cross-platform apps, only to be overtaken by the cross-compiled approach in 2018. In terms of Google Play Store categories, specifically Business ($\sim$36%), Finance ($\sim$25%) and Education ($\sim$21%) saw a great deal of cross-platform apps, unlike categories such as Photography ($\sim$2,44%) and Personalization ($\sim$0,41%), which both saw minimal use of cross-platform frameworks. We also found that native apps are not inherently smaller in `.apk` size than cross-platform apps.

For future work, we will continue including more cross-platform frameworks and development approaches in the identification algorithm. Additionally, working out a set of rules for properly identifying native apps, thus ruling out the use of a cross-platform framework, is an immediate priority. Looking to the Apple iOS ecosystem would also be beneficial for an even more thorough understanding of the state of cross-platform frameworks across the major mobile platforms.

References

[1] Statista, "Global app economy size 2021." https://www.statista.com/statistics/267209/global-app-economy/, 2018.

[2] S. Kemp, "Digital 2020: April global statshot." https://datareportal.com/reports/digital-2020-april-global-statshot, 2020.

[3] A. Biørn-Hansen, T.-M. Grønli, and G. Ghinea, "A survey and taxonomy of core concepts and research challenges in Cross-Platform mobile development," *ACM Comp Surv*, vol. 51, no. 5, pp. 108:1–108:34, 2019.

[4] C. Rieger and T. A. Majchrzak, "A Taxonomy for App-Enabled Devices: Mastering the Mobile Device Jungle," in *Revised Selected Papers WEBIST 2017*, vol. 322 of *LNBIP*, pp. 202–220, Springer, 2018.

[5] S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in *Proc. 6th Balkan Conf. in Informatics*, BCI '13, pp. 213–220, ACM, 2013.

[6] C. Rieger and T. A. Majchrzak, "Towards the definitive evaluation framework for cross-platform app development approaches," *JSS*, vol. 153, pp. 175–199, 2019.

[7] H. Heitkötter, S. Hanschke, and T. A. Majchrzak, "Evaluating Cross-Platform development approaches for mobile applications," in *Webist*, LNBIP, pp. 120–138, Springer, 2012.

[8] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, and S. Ge, "An empirical study of investigating mobile applications development challenges," *IEEE Access*, vol. 6, pp. 17711–17728, 2018.

[9] B. Góis Mateus and M. Martinez, "An empirical study on quality of android applications written in kotlin language," *EMSE*, no. 24, 2019.

[10] S. Habchi, G. Hecht, R. Rouvoy, and N. Moha, "Code smells in iOS apps: How do they compare to android?," in *Proc. 4th MOBILESoft*, pp. 110–121, IEEE/ACM, 2017.

[11] L. Cruz and R. Abreu, "Performance-Based guidelines for energy efficient mobile applications," in *Proc. 4th MOBILESoft*, pp. 46–57, IEEE/ACM, 2017.

[12] A. Charland and B. LeRoux, "Mobile application development: Web vs. native," *Queueing Syst.*, vol. 9, no. 4, p. 20, 2011.

[13] I. T. Mercado, N. Munaiah, and A. Meneely, "The impact of cross-platform development approaches for mobile applications from the user's perspective," in *WAMA'16*, pp. 43–49, ACM, 2016.

[14] I. Malavolta, S. Ruberto, T. Soru, and V. Terragni, "End users' perception of hybrid mobile apps in the google play store," in *IEEE MS*, pp. 25–32, IEEE, 2015.

[15] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *2014 ACM METRICS*, vol. 42, pp. 221–233, ACM, 2014.

[16] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo: collecting millions of android apps for the research community," in *Proc. 13th Int. Conf. on MSR*, pp. 468–471, ACM, 2016.

[17] W. Martin, F. Sarro, Y. Jia, Y. Zhang, and M. Harman, "A survey of app store analysis for software engineering," *IEEE Trans. Softw. Eng.*, vol. 43, no. 9, pp. 817–847, 2017.

[18] M. Ali and A. Mesbah, "Mining and characterizing hybrid apps," in *WAMA'16*, pp. 50–56, ACM, 2016.

[19] L. Corbalán, P. J. Thomas, L. Delia, G. Cáseres, P. Pesado, and others, "A study of non-functional requirements in apps for mobile devices," in *Cloud comp. and big data*, vol. 1050, pp. 125–136, Springer, 2019.

[20] S. Tolomei, "Shrinking APKs, growing installs." https://medium.com/googleplaydev/shrinking-apks-growing-installs-5d3fcba23ce2, 2017.

[21] B. Skuza, A. Mroczkowska, and D. Włodarczyk, "Flutter vs react native – what to choose in 2019?." https://www.thedroidsonroids.com/blog/flutter-vs-react-native-what-to-choose-in-2019, 2019.

[22] S. Greif, R. Benitte, and M. Rambeau, "Mobile & desktop - overview." https://2018.stateofjs.com/mobile-and-desktop/overview/, 2018.

[23] Gartner, "Gartner says by 2016, more than 50 percent of mobile apps deployed will be hybrid." http://www.gartner.com/newsroom/id/2324917, 2013.

[24] T. Anglin and Telerik, "Web, native and cross-platform - three approaches to mobile app development," tech. rep., Telerik, 2014.

[25] V. Ahti, S. Hyrynsalmi, and O. Nevalainen, "An evaluation framework for Cross-Platform mobile app development tools: A case analysis of adobe PhoneGap framework," in *17th CompSysTech*, pp. 41–48, ACM, 2016.

[26] S. Dhillon and Q. H. Mahmoud, "An evaluation framework for cross-platform mobile application development tools," *Softw. Pract. Exp.*, vol. 45, no. 10, pp. 1331–1357, 2015.

[27] R. Nunkesser, "Beyond Web/Native/Hybrid: A new taxonomy for mobile app development," in *5th MOBILESoft*, pp. 214–218, IEEE/ACM, 2018.

[28] G. Kambourakis, A. Shabtai, C. Kolias, and D. Damopoulos, *Intrusion Detection and Prevention for Mobile Ecosystems*. CRC Press, 2017.

[29] S. Alpert, "Introducing react native." https://reactjs.org/blog/2015/03/26/introducing-react-native.html, 2015.

[30] P. Vorbach, "Cordova download statistics." https://npm-stat.com/charts.html?package=cordova&from=2014-01-01&to=2019-09-12, 2019.

[31] M. Willocx, J. Vossaert, and V. Naessens, "A quantitative assessment of performance in mobile app development tools," in *IEEE MS*, pp. 454–461, IEEE, 2015.

[32] Google, "FAQ about google trends data." https://support.google.com/trends/answer/4365533?hl=en.

[33] A. Pirchner, "gsvi." https://pypi.org/project/gsvi/, 2020.

[34] A. Biørn-Hansen, T.-M. Grønli, G. Ghinea, and S. Alouneh, "An empirical study of Cross-Platform mobile development in industry," *Wirel Commun Mob Comput*, vol. 2019, 2019.