Exploring the Development of Micro-Apps: A Case Study on the BlackBerry and Android Platforms

Mark D. Syer[1], Bram Adams[1], Ying Zou[2] and Ahmed E. Hassan[1]
Software Analysis and Intelligence Lab (SAIL)[1], Software Reengineering Research Group[2]
School of Computing[1], Department of Electrical and Computer Engineering[2], Queen's University, Canada
{mdsyer, bram}@cs.queensu.ca, ying.zou@queensu.ca, ahmed@cs.queensu.ca

***Abstract*—The recent meteoric rise in the use of smartphones and other mobile devices has led to a new class of applications, i.e., micro-apps, that are designed to run on devices with limited processing, memory, storage and display resources. Given the rapid succession of mobile technologies and the fierce competition, micro-app vendors need to release new features at break-neck speed, without sacrificing product quality. To understand how different mobile platforms enable such a rapid turnaround-time, this paper compares three pairs of feature-equivalent Android and Blackberry micro-apps. We do this by analyzing the micro-apps along the dimensions of source code, code dependencies and code churn. BlackBerry micro-apps are much larger and rely more on third party libraries. However, they are less susceptible to platform changes since they rely less on the underlying platform. On the other hand, Android micro-apps tend to concentrate code into fewer files and rely heavily on the Android platform. On both platforms, code churn of micro-apps is very high.**

***Index Terms*—mobile platforms; micro-apps; Android; BlackBerry;**

I. Introduction

Mobile devices have changed the software development world by the rapid emergence of a new class of software applications that has opened up a multi-billion dollar market [1]. Micro-apps, commonly referred to as apps (e.g., BlackBerry *App* World, iPhone *App* Store), distinguish themselves from applications that run on typical desktops or servers by their limited functionality, low memory and CPU footprint, touch interfaces, and limited screen sizes and resolutions. Similar to web applications [2], micro-apps are rapidly developed by small teams who may only have limited experience with software development [3]–[6].

Although micro-apps have been around for over a decade, micro-app development did not take off until 2008, when Apple opened the Apple App Store [1], [7]. The total number of stores selling (or hosting) micro apps has increased from less than 5 in 2008 to over 50 in 2011. Downloads of micro-apps have followed the same trends, rising from 7 billion downloads in 2009 to 15 billion in 2010 and a projected 50 billion in 2012 [1]. The revenues from micro-app downloads have also followed similar trends, rising from $4.1 billion in 2009 to $6.5 billion in 2010 and $16.7 billion in 2012 (projected) [1].

The purpose of this paper is to explore the new world of micro-apps, since such apps are expected to be one of the major challenges for software maintenance and program understanding in the near future. This software, and the hardware it relies on, is constantly and rapidly evolving. When micro-apps first rose to prominence in 2008 there were very few mobile platforms. Now, three years later, there are several major mobile platforms, each of which have taken turns as the most popular one [8]–[10]. Developers need to target multiple platforms in response to shifting consumer preferences. However, amidst market pressure and with limited resources and experience, how companies do (and should do) this is an open research question.

As a first step towards addressing this question, we start with an exploratory study to compare the differences between micro-apps for different platforms: are there clear differences in code characteristics, dependencies and churn? We have studied micro-apps that have feature-equivalent versions for different platforms. In total, six micro-apps and two platforms were selected. Our study addresses the following three research questions:

- *RQ1: How different are the code characteristics between platforms?* – Less code is required to implement a feature on the Android platform. BlackBerry micro-apps include and customize more third party source code.
- *RQ2: How different are the number and type of dependencies between platforms?* – Android micro-apps rely much more on the underlying platform than BlackBerry micro-apps do. Over 50% of the dependencies on the BlackBerry platform can be attributed to user interface APIs. Micro-apps written for either the Android or BlackBerry platforms rely on the Java APIs for at least one third of their dependencies.
- *RQ3: How different is the amount of code churn between platforms?* – The maintained third party library code changes very little. Code churn is very high on both platforms.

This paper is organized as follows: Section II motivates our case study of the development of Android and BlackBerry micro-apps. Section III describes the setup of our case study and Section IV describes and discusses the results of our case study. Section V outlines the threats to validity. Finally, Section VI concludes the paper.

II. Motivation and Related Work

The number of purchases of micro-apps has seen an explosive growth in the past few years. This growth is expected to

continue well into the future [3], [11]. Despite this growth, the distribution of the number of micro-apps is highly non-uniform across the different popular mobile platforms (Apple's iPhone, Google's Android and Research In Motion's BlackBerry). Table I shows the total number of micro-apps available for each platform as well as the average number of micro-apps installed on each mobile device in September 2010 [12]–[15]. Since then, 100,000 micro-apps have been added to the Apple App Store, 220,000 to the Android Market and 5,000 to the BlackBerry App World [12]–[14].

TABLE I
NUMBER OF MICRO-APPS BY PLATFORM [12]–[15].

Company - Platform	Number of Micro-Apps Available	Average Number of Micro-Apps Purchased Per Device
Apple - iPhone	250,000	40
Google - Android	80,000	25
RIM - BlackBerry	10,000	14

The number of micro-apps available on each platform is affected by many factors, including marketing, public perception and the overall development experience [9], [16]. Notwithstanding the considerable differences presented in Table I, four of the top five most popular micro-apps on each platform are actually the same. This can be seen in Table II, which shows the most popular micro-apps by platform based on a survey of 4,000 micro-app users in August 2010 [15]. Therefore, in order to reach the largest consumer base, micro-app developers need to develop for each of these mobile platforms.

TABLE II
MOST POPULAR MICRO-APPS BY PLATFORM [15].

iPhone	Android	BlackBerry
1. Facebook	1. Google Maps	1. Facebook
2. Weather Channel	2. Facebook	2. Weather Channel
3. Google Maps	3. Weather Channel	3. Google Maps
4. iPod/iTunes	4. Pandora	4. Pandora
5. Pandora	5. YouTube	5. Twitter

However, companies have a hard time porting and maintaining their micro-apps on multiple mobile platforms. First, the explosion of new mobile devices, operating systems and frameworks has resulted in a highly fragmented market [17]. Developers need to make their code aware of different features and quirks of the supported devices, and update their micro-app for every new major device or new version of the operating system. Second, development tools have been released freely to the general public so that anyone can develop a micro-app, even without prior development experience [3]. However, to our knowledge there have been no detailed studies on the micro-app development or maintenance processes. A good understanding of these processes is necessary to grasp the speed and scale of micro-app development as well as the need for mechanisms to defend against platform changes and maintain backwards comparability.

Gasimov et al. have surveyed micro-apps and classified them into general categories. The authors have also surveyed the development tools available to micro-app developers [17].

To aid in cross platform development, Wu et al. use a mobile web application engine that can run C++ code on the Android platform through the Java Native Interface [18].

Teng and Helps have designed a project for a junior level operating systems course in which 35 students were asked to develop a micro-app. The authors surveyed the students who completed the project to evaluate the overall development and learning experience [19]. The authors determined that the majority of students who took the course feel that micro-app development is the important component of an information technology curriculum.

As a first step toward understanding the development and maintenance of micro-apps, this paper presents an exploratory study of micro-app development for two popular mobile platforms. In particular, we study the source code, dependency and code churn properties of micro-apps on the Android and BlackBerry platforms.

III. CASE STUDY SETUP

This section outlines our approach to explore the development of micro-apps. First, we selected micro-apps that have feature-equivalent versions for the Android and BlackBerry platforms. Second, we measured the source code, dependency and churn properties of theses micro-apps. We then compared the measurements across the subject micro-apps.

A. Application Selection

We selected applications for our case study based on the following criteria. First, the selected micro-apps must be open source, as we require access to the source code repository. Second, we require micro-app pairs that have feature-equivalent versions that run on different platforms. This requirement allows us to directly compare the effort it takes to implement equivalent functionality on the two platforms, possibly written by different developers (or companies). Third, we require that the micro-apps were developed in the same programming language. This requirement simplifies our case study, since it is hard to compare the source code characteristics of a micro-app written in Objective-C for the Apple iPhone platform versus a micro-app written in Java for the Google Android or RIM BlackBerry platforms.

We selected the Android and BlackBerry platforms as the focus of our study, because (1) they are two of the most popular mobile platforms and (2) micro-apps for these platforms are written mostly in Java [3], [8], [9].

Three micro-apps, WordPress, Google Authenticator and Facebook SDK, were selected for our case study. To ensure that the micro-app pairs are feature equivalent, we verified feature differences using feature lists on the micro-app webpages, change logs for each release and feature requests in the forum or issue tracking systems.

WordPress is one of the most popular content management systems in use today. The WordPress micro-app is open-source, available on the Android and BlackBerry platforms

and the features of both versions are nearly identical [20]–[22]. The WordPress micro-apps allow users to manage their blog or web page from their mobile device. Source code for the WordPress for Android micro-app was first committed to the repository in September 2009, while code for the BlackBerry micro-app was first committed to the repository in April 2009.

Google Authenticator is a micro-app that allows users to generate 2-step verification codes on their mobile devices without an Internet connection. This adds an extra layer of security to a user's Google Account (e.g., Gmail) by requiring the user to have access to his/her phone (in addition to the typical username and password) [23]. Both versions of the Google Authenticator micro-apps are developed by Google. Hence, developers for both the Android and BlackBerry versions of the Google Authenticator micro-app share the same source code repository and bug database. Source code for both versions of the Google Authenticator micro-app was first committed to the repository in March 2010.

The Facebook SDK is an open source project that allows developers to integrate Facebook's functionality into their own applications [24], [25]. The Facebook SDK for BlackBerry was developed by Research in Motion, whereas the Facebook SDK for Android was developed by Facebook. Source code for the Facebook SDK for Android was first committed to the repository in May 2010. Source code for the Facebook SDK for BlackBerry was first committed to the repository in July 2010.

The source code for each version of these micro-apps is available in the repositories listed in Table III. We perform our analysis on the source code in the repository up to, and including, the last commit which was tagged as a release.

TABLE III
MICRO-APP REPOSITORIES.

WordPress		
Android	android.svn.wordpress.org	1.4.0
BlackBerry	blackberry.svn.wordpress.org	1.4.6.2
Google Authenticator		
Android	google-authenticator.googlecode.com/hg	0.54
BlackBerry	google-authenticator.googlecode.com/hg	1.1.2
Facebook SDK		
Android	github.com/facebook/facebook-android-sdk.git	1.5.0
BlackBerry	facebook-bb-sdk.svn.sourceforge.net	0.4.5

B. Source Code Properties

We used the Understand tool by SciTools [26] to extract the metrics in Table IV for each micro-app. Understand is a static analysis toolset for measuring and analyzing the source code of small- to large-scale software projects written in a number of programming languages.

TABLE IV
SOURCE CODE VOLUME METRICS.

Metric	Definition
Files	Total Number of Files Containing Source Code
Classes	Total Number of Classes
Lines Code	Total Number of Lines of Code

We measured the source code volume metrics for the entire micro-app, and for two subsets of the micro-app base: micro-app specific source code and third party library source code.

Third party libraries consist of reusable software components developed and maintained by developers unaffiliated with the micro-app. For example, CWAC (CommonsWare Android Components) is a collection of open source libraries specifically developed to help Android micro-app developers tackle common and recurring issues [27]. Micro-apps often include, customize and maintain the source code of third party libraries, therefore it is important to study the project-specific source code metrics and the maintained third party library source code independently.

In order to identify third party libraries, we examine the projects' directory structure looking for utility directories or directories commonly associated with third party libraries (e.g., src/com/ on the BlackBerry platform). In all six micro-apps, the third party libraries are included as `.java` files. Therefore, we are able to examine each source code file for license agreements, disclaimers, documentation or links to other projects in the source code comments.

After we classified each source code file as either third party or project-specific, we compared the source code volume metrics for both groups of files to determine how much of the micro-app the developers have to develop and maintain themselves (i.e., everything other than the third party code).

C. Dependency Properties

Similar to desktop and web applications, micro-apps make use of APIs that provide access to functionality that the developers would otherwise have to implement themselves. Three types of APIs are provided to developers: the Java API, the platform API (i.e., Android- or BlackBerry-specific APIs) and third party libraries. Android developers have access to nearly all of the Java 2 Standard Edition APIs, whereas BlackBerry developers have access to the Java 2 Micro Edition.

We used the Understand tool introduced in Section III-B to extract, for each class in the micro-app, a list of classes on which the class depends. These dependencies were classified into one of the following categories based on the class name:

- **Language dependency** - dependency on a class that is part of the Java platform (e.g., `java.io.IOException` or `java.lang.Thread`).
- **User Interface dependency** - dependency on a class that is part of the device platform and that is responsible for the user interface (e.g., `android.view` or `net.rim.device.api.ui`).
- **Platform dependency** - dependency on a class that is part of the device platform and not responsible for the user interface (e.g., `android.app.Activity` or `net.rim.device.api.system.EventLogger`).
- **Third Party dependency** - dependency on a class that is part of a third party library.
- **Project dependency** - dependency on some class in the micro-app code base other than a third party class.

Listing 1. Hello, World - An Android Developer's First Micro-App [28].

```
1 import android.app.Activity;
2 import android.os.Bundle;
3 import android.widget.TextView;
4
5 public class HelloAndroid extends Activity{
6   public void onCreate(Bundle savedInstanceState){
7     super.onCreate(savedInstanceState);
8     TextView tv = new TextView(this);
9     tv.setText("Hello, Android");
10    setContentView(tv);
11  }
12 }
```

Understand extracts and counts the following types of class dependencies:

- Calls - call to a method in another class
- Casts - cast to an object type defined in another class
- Creates - creation of an object whose type is defined in another class
- Extends - extending another class
- Implements - implementing an interface
- Sets - setting a variable or object defined in another class
- Typeds - use of an object type defined in another class
- Uses - use of a variable or object defined in another class

As an example of this analysis, consider the "Hello, World" code for the Android in Listing 1 [28]. From this example, Understand extracts the class dependencies in Table V.

TABLE V
LISTING 1 CLASS DEPENDENCIES

Dependency	Cause	Line
app.Activity	HelloAndroid **Extends** Activity	5
os.Bundle	savedInstanceState **Typeds** Bundle	6
widget.TextView	tv **Typeds** TextView	8
	HelloAndroid.onCreate **Creates** TextView	8
	HelloAndroid.onCreate **Calls** TextView	8
	HelloAndroid.onCreate **Calls** setText	9

In this paper, we count the total and unique number of dependencies on each dependency category. For example, we can summarize the dependency information of Listing 1 as in Table VI. Such a class dependency summary is a measure of how strongly a micro-app is tied to Java, the underlying platform, the UI, third party libraries or itself.

TABLE VI
LISTING 1 CLASS DEPENDENCY SUMMARY

Dependency Class	Number of Dependencies	
	Total	Unique
Language	0	0
User Interface	3	1
Platform	6	3
Third Party	0	0
Project	0	0

We define the "platform dependency ratio" as the ratio between the number of platform and user interface dependencies, and the total number of dependencies. A low platform dependency ratio indicates that developers do not rely significantly on the platform APIs. For example, their micro-app may be simple or self-contained, or the platform may be too difficult to use. Such micro-apps can be easily ported to other platforms. Conversely, a high platform dependency ratio indicates that micro-app developers heavily exploit platform APIs. However, this leads to platform "lock-in", which complicates porting to other platforms and potentially introduces instability due to the rapid evolution of mobile device platforms. For example, Listing 1 has a platform dependency ratio of 100%, i.e. it is highly tied to the Android platform and might need to be rewritten completely to port it to another platform.

The effort required to port a micro-app from one platform to another depends on a number of factors (e.g., the number and complexity of features and the platform dependency ratio). In our specific case study, we study three pairs of feature equivalent micro-apps. Therefore, we expect that the platform dependency ratio is a good measure of platform lock-in.

D. Code Churn Properties

Source code is constantly changing throughout the development process in response to maintenance and evolution activities. Code churn measures how much source code changes over time. We measure code churn using the following metrics:

- Number of files changed per change set.
- Number of project specific files changed per change set.
- Number of third-party files changed per change set.
- Number of lines changed per change set.
- Number of project-specific lines changed per change set.
- Number of third party lines changed per change set.

For our metrics, we ignored the initial commit, since this would heavily skew the churn metrics, as well as change sets that did not change any Java source code file. In addition, we measure the number of non-white space lines of code. We did not include changes to comment lines, since these are hard to detect from the change set diffs. We calculated our line churn metrics as the sum of all the added and deleted lines, for each file, for each commit.

Given the wide variety of source code repository formats, we used the following tools/commands to extract the number of changes, changed files and changes lines from the following repository formats:

- Subversion (svn) - We used statsvn [29] to extract the total number of changes and total number of changed lines for each file and directory in a project. Statsvn is an open source tool for generating project development metrics that characterize developer activity, project growth and code churn.
- GIT - we used `git log --numstat`, a standard git command [30], on the entire repository to extract the number of lines added and deleted from each file during each commit.
- Mercurial (hg) - we used `hg churn -f '\%s'`, a standard mercurial extension [31], on each file in the repository to extract the number of lines modified in each file in each commit.

IV. Case Study Results

This section presents the results of our case study on the three pairs of micro-apps selected in Section III-A.

RQ1: How different are the code characteristics between platforms?

Motivation: Source code volume metrics have been shown to be highly correlated to the complexity of a software system [32], [33]. The complexity of a software system measures the difficulty of understanding, evolving and maintaining the system. We measure and compare the source code volume metrics for each pair of micro-apps to determine if differences between the Android and BlackBerry platforms require developers on either platform to write more complex source code when implementing similar functionality.

We also measure and compare the source code volume metrics for the micro-app and third party libraries. Reuse of third party libraries has many possible benefits including reduced development time and increased quality. However, third party software has potential drawbacks, as micro-app developers must ensure their copy of the third party API is always in sync and up to date with the most recent updates and bug fixes, although they may not be familiar with the third party library code. In addition, micro-app developers can be negatively impacted if support for the third party library or parts of its functionalities are is abandoned.

Approach: We used the methodology presented in Section III-B to extract source code volume metrics from each micro-app. Table VII presents the results of these metrics and the percentage of increase of each metric for the BlackBerry version relative to the Android version for each micro-app pair. After identifying which source code files belong to third party code, and which files contain actual micro-app code, we extracted the source code volume metrics of both groups of files. Table VIII presents the values of these metrics and the percentage (in parentheses) of the total code base for each category of each micro-app pair.

TABLE VII
Global Source Code Volume Metrics and Difference Relative to Android.

WordPress			
Metric	Android	BlackBerry	Difference
# Files	55	241	+338%
# Classes	360	575	+60%
# Lines Code	15,928	35,775	+125%
Google Authenticator			
Metric	Android	BlackBerry	Difference
# Files	10	31	+210%
# Classes	26	61	+135%
# Lines Code	1,344	3,322	+147%
Facebook SDK			
Metric	Android	BlackBerry	Difference
# Files	6	57	+850%
# Classes	12	77	+542%
# Lines Code	777	5,070	+553%

TABLE VIII
Breakdown of Source Code Volume Metrics Across Project-Specific Code.

WordPress			
Metric	Android	BlackBerry	Difference
# Files	40 (73%)	211 (88%)	528%
# Classes	331 (92%)	543 (94%)	164%
# Lines of Code	12,948 (81%)	30,764 (86%)	237%
Google Authenticator			
Metric	Android	BlackBerry	Difference
# Files	10 (100%)	17 (55%)	170%
# Classes	26 (100%)	47 (77%)	181%
# Lines of Code	1,344 (100%)	1,421 (43%)	106%
Facebook SDK			
Metric	Android	BlackBerry	Difference
# Files	6 (100%)	21 (37%)	350%
# Classes	12 (100%)	33 (43%)	275%
# Lines of Code	777 (100%)	1,723 (34%)	222%

TABLE IX
Breakdown of Source Code Volume Metrics Across Third Party Code.

WordPress			
Metric	Android	BlackBerry	Difference
# Files	15 (27%)	30 (12%)	200%
# Classes	29 (8%)	32 (6%)	110%
# Lines of Code	2,980 (19%)	5,011 (14%)	168%
Google Authenticator			
Metric	Android	BlackBerry	Difference
# Files	0 (0%)	14 (45%)	—
# Classes	0 (0%)	14 (23%)	—
# Lines of Code	0 (0%)	1,901 (57%)	—
Facebook SDK			
Metric	Android	BlackBerry	Difference
# Files	0 (0%)	36 (63%)	—
# Classes	0 (0%)	44 (57%)	—
# Lines of Code	0 (0%)	3,347 (66%)	—

To better understand the distribution of project-specific file sizes, we visualize the file size, in terms of the number of lines of code, using bean plots. Bean plots are an alternative to box plots to summarize and compare the distribution of different sets of data [34]. Figure 1, 2 and 3 also show the median file size of each micro-app (solid black line).

Results: Table VII shows that a micro-app written for the BlackBerry platform contains two (+125%) to more than six (+553%) times as many lines of code as the equivalent Android micro-app. The differences in number of files are even larger (+850%). This difference is not merely due to differences in coding style of the developers developing the Android and BlackBerry apps, since the Google Authenticator micro-apps (developed by the same company) also show these differences. If source code volume is a good indicator of development effort, more effort seems to be needed on the BlackBerry platform. However, since most BlackBerry micro-apps contain both their own source code, as well as the source code for third party libraries, it is necessary to break down the volume metrics across micro-app and third party code (Table VIII).

Table VIII shows that in two of the three micro-app pairs the BlackBerry micro-apps contain more lines of code in third party libraries than in project-specific source code (57% in

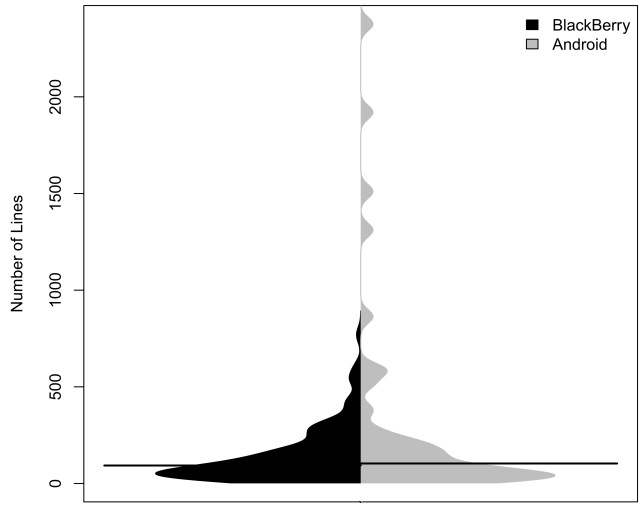

Fig. 1. Distribution of file sizes across the WordPress Micro-App project-specific files.

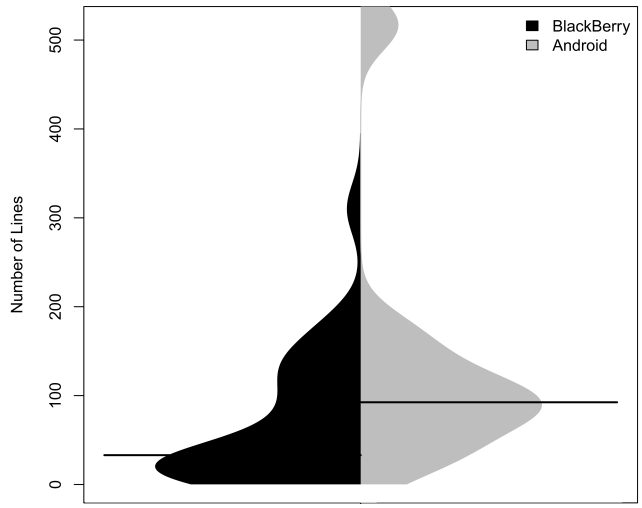

Fig. 2. Distribution of file sizes across the Google Authenticator Micro-App project-specific files.

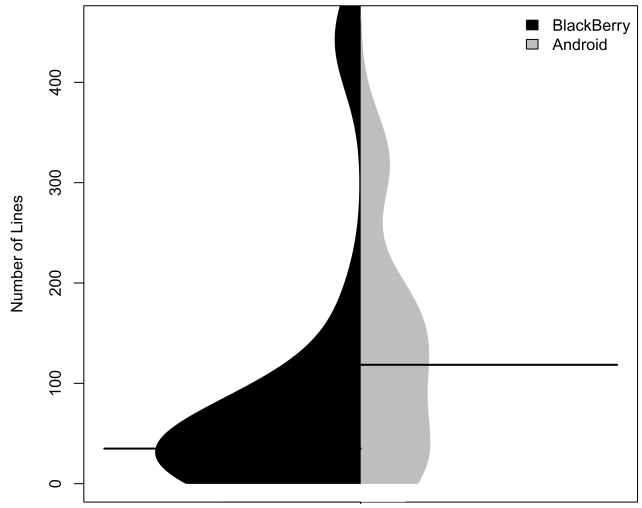

Fig. 3. Distribution of file sizes across the Facebook SDK project-specific files.

Google Authenticator and 66% in the Facebook SDK). On the other hand, in the Android micro-app's third party libraries are either not included at all (Facebook SDK and Google Authenticator) or make up approximately 19% of the lines of code (WordPress). When looking only at the project-specific code, these BlackBerry micro-apps are still up to two times as large as the corresponding Android micro-app.

From Figure 1, 2 and 3, Android micro-app developers tend to write much larger files (with respect to lines of code) than BlackBerry micro-app developers. First, from Figure 1 and Figure 2, Android micro-apps have more outliers, i.e., more files that are significantly larger than the median file size. This is particularly true in the WordPress for Android micro-app, where a significant amount of WordPress code is concentrated in a few files. Second, from Figure 2 and Figure 3, the median file size of an Android micro-app is more than twice the median file size of the feature-equivalent BlackBerry micro-app.

Note, however, that the area of bean plots is standardized to 1 (i.e., the bean plots cannot be used to compare the total code size of the micro-apps in each pair).

> Less source code is required for Android micro-apps than feature equivalent BlackBerry micro-apps. Android micro-app developers typically write larger source code files and tend to concentrate more code into fewer files. BlackBerry micro-apps include more third party libraries in their code base.

RQ2: How different are the number and type of dependencies between platforms?

Motivation: We study the API usage properties of each micro-app to uncover how micro-apps depend on language, platform, user interface and third party APIs, and on their own classes. Since micro-app developers need to port their micro-apps to multiple platforms, a high platform dependency ratio (defined in Section III-C) negatively impacts the porting process by increasing the amount of code that needs to be rewritten).

Approach: We use the methodology presented in Section III-C to extract the number of dependencies for each class in the micro-app. Table X presents our measurements for these key metrics and the percentage of the total number of dependencies (in parentheses) for each category of dependencies and each micro-app pair. Table X also presents the percentage of increase in the metrics for the BlackBerry version relative to the Android version of each micro-app. Finally, Table XI presents the platform dependency ratio, defined in Section III-C, for each micro-app.

Results: Table X and Table XI expose several interesting trends. (1) Android micro-apps rely much more on platform and user interface APIs than their BlackBerry equivalents (platform dependency ratios of 41%, 51% and 34% on the Android platform compared to 15%, 12% and 6% on

TABLE X
SOURCE CODE DEPENDENCY METRICS.

WordPress			
Dependency	Android	BlackBerry	Difference
Language	5860 (43%)	6720 (33%)	+15%
Platform	3593 (27%)	600 (3%)	-83%
User Interface	1961 (15%)	2473 (12%)	+26%
Third Party	365 (3%)	665 (3%)	+82%
Project Specific	1724 (13%)	10132 (49%)	+488%
Google Authenticator			
Dependency	Android	BlackBerry	Difference
Language	312 (33%)	949 (58%)	+204%
Platform	269 (28%)	43 (3%)	-84%
User Interface	223 (23%)	154 (9%)	-31%
Third Party	0 (0%)	192 (12%)	—
Project Specific	156 (16%)	300 (18%)	+92%
Facebook SDK			
Dependency	Android	BlackBerry	Difference
Language	252 (42%)	1176 (38%)	+367%
Platform	170 (29%)	79 (3%)	-54%
User Interface	31 (5%)	99 (3%)	+219%
Third Party	0 (0%)	1101 (36%)	—
Project Specific	140 (24%)	606 (20%)	+333%

TABLE XI
MICRO-APP PLATFORM DEPENDENCY RATIOS.

Micro-App	Android	BlackBerry
WordPress	41%	15%
Google Authenticator	51%	12%
Facebook SDK	34%	6%

the BlackBerry platform). (2) BlackBerry micro-apps depend heavily on project-specific classes, far more than they rely on the BlackBerry platform (49% of WordPress dependencies, 18% of Google Authenticator dependencies and 20% of Facebook SDK dependencies compared to 3% non-User Interface dependencies). (3) BlackBerry micro-apps rely on the underlying platform primarily for the user interface libraries. In the WordPress and Google Authenticator BlackBerry micro-apps, 80% of the platform dependencies are on user interface libraries. (4) For Android, the Android and Java APIs appear to provide most of the dependencies of the micro-apps. Even excluding the user interface APIs, over 25% of the Android micro-app dependencies are on the Android platform, leading to a relatively high platform dependency ratio (41% for WordPress, 51% for Google Authenticator and 34% for Facebook SDK). (5) Finally, in all three Android micro-apps, third party dependencies account for fewer dependencies than any other dependency category.

From Table X, it seems that the extent to which each micro-app depends on third party libraries seems to fluctuate from 0% in the Google Authenticator micro-app and Facebook SDK for Android to 36% in the Facebook SDK for BlackBerry. Manual analysis shows that these third party libraries are especially used for implementing functionality that is missing from a language or platform API. Examples of missing functionality on the BlackBerry platform are the Java Script Object Notation protocol and regular expression support, and on both platforms the module for XML-Remote Procedure Calls.

Java Script Object Notation (JSON) is a light weight data interchange format for language-independent client-server communication [35]. Within the BlackBerry version of the WordPress micro-app, JSON is used for geocoding and fetching of page statistics from the WordPress back-end. JSON is included in the Android API `org.json` [36], but on the BlackBerry platform, prior to version 5.0.0, micro-app developers needed to include their own implementation of the JSON format [36]–[38]. Although JSON is included in version 6.0.0 of the BlackBerry platform, micro-app developers still need to maintain a third party implementation of JSON to preserve backwards compatibility. [38].

Regular expressions are typically used for search and replace operations in strings and extraction of substrings. Regular expression functionality is included in the standard Java library `java.util.regex` [36], which is not available on the J2ME platform that is supported by the BlackBerry platform. The WordPress for BlackBerry micro-app uses the Jakarta Regexp regular expression package from the Apache Jakarta Project [39] to determine the number of characters in a comment, post or page, before posting to a web site or blog.

XML-RPC is a lightweight mechanism for exchanging data and invoking web services. XML-RPC is used by both the Android and BlackBerry versions of the WordPress micro-app. The WordPress for BlackBerry micro-app uses the kXML-RPC implementation, a J2ME XML-RPC implementation built on top of the kXML parser [40]. The WordPress for Android micro-app uses the android-xmlrpc implementation, a very thin XML-RPC client library for the Android platform [41].

Apart from missing functionality, third party libraries are also used as an alternative to poorly implemented functionality in the language or platform APIs. One example of poorly implemented functionality on the Android platform is the visualization of lists of thumbnails off the Internet. The third party Thumbnail module from the CommonsWare Android Components library allows to load and cache thumbnail images transparently in the background to avoid tying up the user interface thread [27]. The module has been included in the WordPress for Android micro-app. Since this module requires the use of the Cache module, the latter module has also been included in the WordPress for Android micro-app [27]. In this case, including one third party library requires the inclusion of a second third party library.

Android micro-apps rely primarily on the Android APIs, whereas BlackBerry micro-apps rely on Java libraries and project-specific classes in the micro-app. Android micro-apps contain little to no third party libraries. More than half of the dependencies on the BlackBerry platform are on user interface APIs. Android micro-apps have a much higher platform dependency ratio than feature equivalent BlackBerry micro-apps.

RQ3: How different is the amount of code churn between platforms?

Motivation: Given the rapid pace and high pressure of the micro-app development market, we want to characterize the effort needed to develop each micro-app. We also explore the code churn properties of the third party libraries to determine the amount of effort needed to maintain them, i.e., do micro-app developers highly customize such libraries, or mainly clone them.

Approach: We use the methodology presented in Section III-D to extract the code churn properties for each micro-app. Table XII presents the values of these metrics and the percentage of the total number of changes (in parentheses) for each class of each micro-app pair.

TABLE XII
CODE CHURN METRICS.

WordPress		
Metric	Android	BlackBerry
Total # File Changes	660	2760
Average # File Changes/File	12.00	11.45
# Third Party File Changes	47 (7%)	59 (2%)
# Project File Changes	613 (93%)	2701 (98%)
Total # Line Changes	23276	46823
Average # Line Changes/Lines	1.04	0.89
# Third Party Line Changes	245 (1%)	648 (1%)
# Project Line Changes	23031 (99%)	46175 (99%)
Google Authenticator		
Metric	Android	BlackBerry
Total # File Changes	12	12
Average # File Changes/File	1.20	0.39
#Third Party File Changes	0 (0%)	0 (0%)
#Project File Changes	12 (100%)	12 (100%)
Total #Line Changes	306	94
Average # Line Changes/Lines	0.16	0.02
# Third Party Line Changes	0 (0%)	0 (0%)
# Project Line Changes	306 (100%)	94 (100%)
Facebook SDK		
Metric	Android	BlackBerry
Total # File Changes	329	38
Average # File Changes/File	54.83	0.67
# Third Party File Changes	0 (0%)	5 (13%)
# Project File Changes	329 (100%)	33 (87%)
Total # Line Changes	2979	473
Average # Line Changes/Lines	1.80	0.05
# Third Party Line Changes	0 (0%)	23 (5%)
# Project Line Changes	2979 (100%)	450 (95%)

We also visualize the line churn using a box plot. Box plots graphically depict the smallest observation, lower quartile, median, upper quartile, and largest observation using a box. Circles correspond to outliers. Figure 4, 5 and 6 depict the line churn characteristics across all commits.

We also examine the growth of the micro-apps in size (lines of code) over time. Figure 7 shows this evolution for the WordPress BlackBerry micro-app over the project's lifetime (from May 2009 to March 2011).

Results: From Table XII, we can see that although Android micro-apps have fewer commits to their repositories, the average number of times a file is changed is much higher for Android micro-apps. For example, the average number

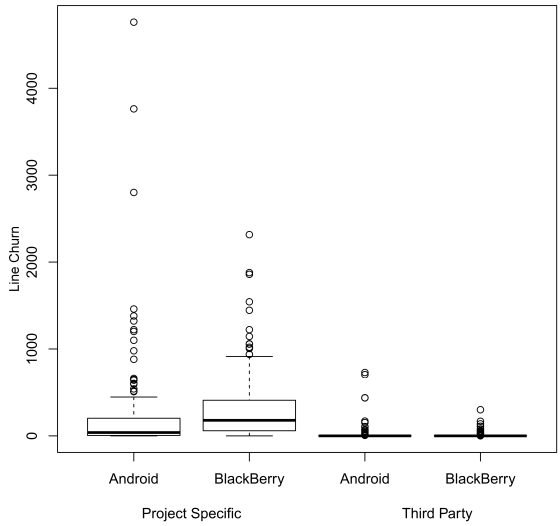

Fig. 4. Line Churn Characteristics of the WordPress Micro-App.

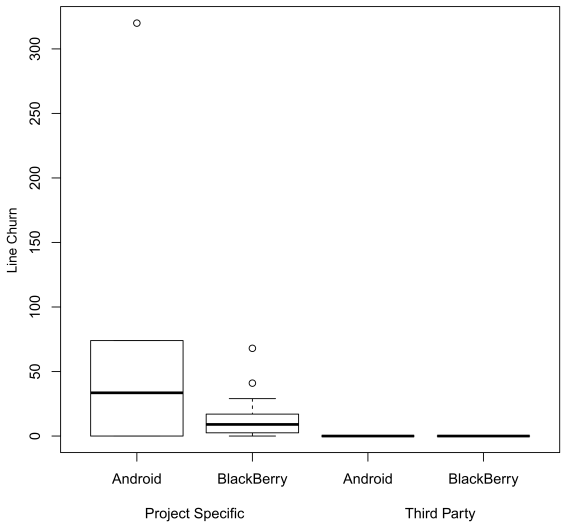

Fig. 5. Line Churn Characteristics of the Google Authenticator Micro-App.

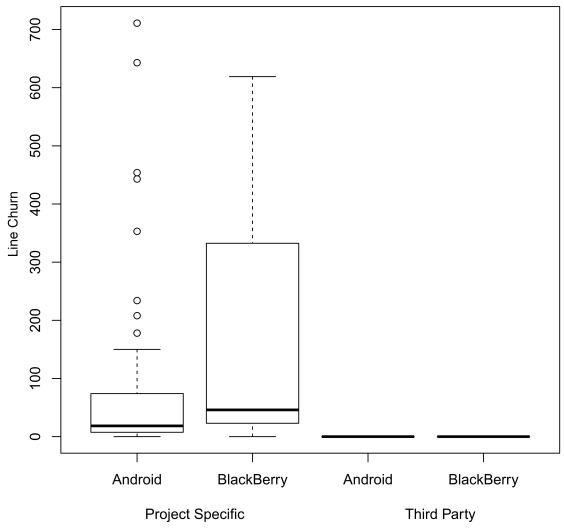

Fig. 6. Line Churn Characteristics of the Facebook SDK.

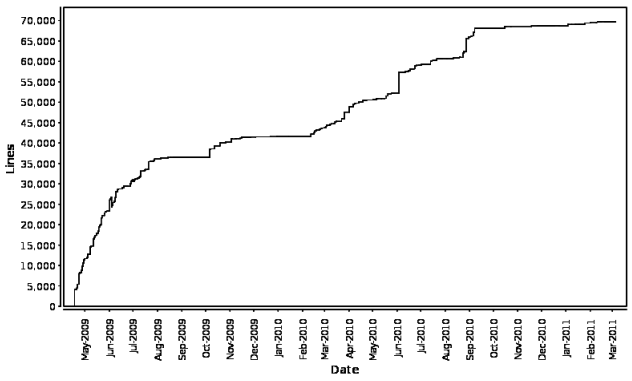

Fig. 7. Size (lines of code) of the WordPress for BlackBerry micro-app from May 2009 to March 2011

of times a file is changed in Android Facebook SDK is 54.83 compared to 0.67 for the BlackBerry micro-app. In this case, Table XII and Figure 6 show that Android micro-apps see many small changes, whereas BlackBerry micro-apps see fewer larger changes. On the other hand, Figure 4 and Figure 6 show that changes to BlackBerry micro-apps typically affect more lines of code, even though Android micro-apps contain more outliers. This indicates that although the size of most changes to Android micro-apps is relatively small, there are a number of relatively large changes.

From Table XII, Figure 4, 5 and 6, third party source code, on either platform, experiences very little code churn after the initial import. This suggests that these libraries are mostly just copied to make the projects self-contained, rather than to heavily customize them. Only for the WordPress micro-app, many large changes are made. We checked the repository and found that 80% of these changes correspond to refactoring or fixing of defects in the kXML-RPC third party library. Without access to the third party library source code micro-app developers would not be able to fix or refactor these libraries themselves.

Figure 7 shows the growth of the WordPress for BlackBerry micro-app (lines of code) over time. The other five micro-apps show a similar pattern, except for the micro-apps with shorter project histories. Those micro-apps show very little growth after the initial commit.

> Source code files in Android micro-apps change more frequently than source code files in BlackBerry micro-apps, but typically see smaller changes. Third party libraries typically change very little.

V. Threats to Validity

The studied micro-apps represent a small subset of the total number of micro-apps available on the Android and BlackBerry platforms. In addition, we did not consider micro-app games, which are the most commonly downloaded micro apps [15], [42], since we were unable to acquire a pair of feature-equivalent micro-app games. Finally, the Google Authenticator micro-app and the Facebook SDK are rather small, approximately 5,000 lines of code. However, we do not know whether these are typical sizes for a micro-app or an outlier. The results of our case study may not generalize to other micro-apps or platforms.

We investigated feature equivalence between each micro-app pairs using feature lists on the micro-app webpages, change logs for each release and feature requests in the forum or issue tracking systems. However, we did not verify that the functioning versions (i.e., installed and operating on a mobile device) had these features. This may have introduced false positives into the Application Selection process (i.e., two micro-apps that are thought to be feature equivalent, are, in fact, not feature equivalent). In addition, although two micro-apps may be feature equivalent, the features may have been implemented very differently (e.g., simple and straight-forward user interface compared to a more complicated interface).

The identification of third-party libraries in each micro-app was done using heuristics and manual analysis. It is possible that some third-party libraries were misidentified using this approach.

The Facebook SDK for BlackBerry was developed by Research In Motion (the company behind the BlackBerry) itself. This may have introduced bias into our study of platform dependencies, since the developers have intimate knowledge of the BlackBerry platform and may be biased towards relying more on BlackBerry APIs. However, the results in Table X seem to contradict this. Similarly, the Google Authenticator micro-apps for Android and BlackBerry were both developed by Google (the company behind Android). Since these developers were simultaneously developing the same micro-app for the Android and BlackBerry platforms, they may have been biased towards using Java APIs (as opposed to device-specific APIs), as well as Android APIs. Table X suggests that such a bias is possible.

Given that micro-apps like the ones that we analyzed are typically only a couple of years old, they do not have the stable project histories of long-lived, commonly studied projects like Linux and Apache. This may have biased our code churn metrics. Given this short history, it is not yet known which micro-apps are (or will be) either successful or representative of good development style.

Given the rapid pace of micro-app development and platform evolution, the micro-apps in our case study are likely to evolve considerably in the near term. The micro-app pairs may no longer be functionally equivalent and may not even be maintained any further.

Finally, one of the authors of this paper holds an NSERC/RIM Industrial Research Chair in Software Engineering. Despite this, we believed we have objectively analyzed and compared the Android and BlackBerry platforms and presented the results fairly and without bias.

VI. Conclusions

This paper presented an exploratory study of micro-apps on two popular mobile platforms, as a first step toward understanding the development and maintenance process of micro-apps. In particular, we studied the source code, dependency and code churn properties of three pairs of feature-equivalent micro-apps on the Android and BlackBerry platforms in order to address the question of how micro-app developers can target multiple platforms with limited resources.

Micro-apps written for the BlackBerry platform are more than twice the size of feature-equivalent Android micro-apps. Missing functionality in the BlackBerry and Java 2 ME APIs has forced BlackBerry micro-app developers to rely on third party libraries that have increased the size (lines of code) of the micro-app.

BlackBerry micro-apps rely less on BlackBerry-specific APIs and more on Java APIs and other classes in the micro-app. On the other hand, Android micro-app developers leverage more Android and Java SE APIs. However, heavy reliance on the Android platform has led to a greater degree of platform lock-in in these Android micro-apps. Therefore, developers who wish to target both the BlackBerry and Android platforms should write their micro-apps for the BlackBerry platform then port their micro-apps to the Android platform.

While micro-apps on both platforms experience a high degree of churn due to constant and rapid evolution, included third party libraries experience very little churn and therefore require little effort by micro-app developers to maintain.

We intend to further explore the development and maintenance properties of additional micro-apps. Since this paper has focused more on the development characteristics of micro-apps, we intend to explore the maintenance characteristics of micro-apps by examining quality metrics in the micro-app and the relationship between software defects and the source code, dependency and churn metrics that we studied in this paper.

References

[1] C. Sharma, "Sizing Up the Global Apps Market," Chetan Sharma Consulting. [Online]. Available: blog.getjar.com/developer/sizing-up-the-global-apps-market

[2] A. Hassan and R. Holt, "Architecture recovery of web applications," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2002, pp. 349–359.

[3] M. Butler, "Android: Changing the mobile landscape," *IEEE Pervasive Computing*, vol. 10, no. 1, pp. 4–7, Jan. 2011.

[4] S. Lohr. Google's Do-It-Yourself App Creation Software. [Online]. Available: nytimes.com/2010/07/12/technology/12google.html

[5] H. Wen. The ascendance of App Inventor. [Online]. Available: radar.oreilly.com/2011/06/google-app-inventor-programmers-mobile-apps.html

[6] D. Gavalas and D. Economou, "Development platforms for mobile applications: Status and trends," *IEEE Software*, vol. 28, no. 1, pp. 77–86, jan 2011.

[7] "Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011," Gartner Inc. [Online]. Available: gartner.com/it/page.jsp?id=1529214

[8] "Apple Leads Smartphone Race, while Android Attracts Most Recent Customers," Nielsen Company. [Online]. Available: blog.nielsen.com/nielsenwire/online_mobile/apple-leads-smartphone-race-while-android-attracts-most-recent-customers

[9] "Who is Winning the U.S. Smartphone Battle?" Nielsen Company. [Online]. Available: blog.nielsen.com/nielsenwire/online_mobile/who-is-winning-the-u-s-smartphone-battle

[10] "U.S. Smartphone Battle Heats Up: Which is the Most Desired Operating System?" Nielsen Company. [Online]. Available: blog.nielsen.com/nielsenwire/online_mobile/us-smartphone-battle-heats-up

[11] "comScore Reports May 2010 U.S. Mobile Subscriber Market Share," ComScore Inc. [Online]. Available: comscore.com/Press_Events/Press_Releases/2010/7/comScore_Reports_May_2010_U.S._Mobile_Subscriber_Market_Share

[12] Android Market. [Online]. Available: https://market.android.com

[13] BlackBerry App World. [Online]. Available: appworld.blackberry.com

[14] Apple App Store. [Online]. Available: apple.com/iphone/apps-for-iphone

[15] "Games Dominate Americas Growing Appetite for Mobile Apps," Nielsen Company. [Online]. Available: blog.nielsen.com/nielsenwire/online_mobile/games-dominate-americas-growing-appetite-for-mobile-apps

[16] J. Murai. You Win RIM! (And Open Letter To RIM's Developer Relations). [Online]. Available: blog.jamiemurai.com/2011/02/you-win-rim

[17] A. Gasimov, C.-H. Tan, C. W. Phang, and J. Sutanto, "Visiting mobile application development: What, how and where," in *Proceedings of the International Conference on Mobile Business and Global Mobility Roundtable (ICMB-GMR)*, Jun. 2010, pp. 74–81.

[18] Y. Wu, J. Luo, and L. Luo, "Porting mobile web application engine to the android platform," in *Proceedings of the International Conference on Computer and Information Technology (CIT)*, Jul. 2010, pp. 2157–2161.

[19] C.-C. Teng and R. Helps, "Mobile application development: Essential new directions for IT," in *Proceedings of the International Conference on Information Technology: New Generations (ITNG)*, Apr. 2010, pp. 471–475.

[20] Usage Statistics and Market Share of Content Management Systems for Websites. W3Techs - Web Technology Surveys. [Online]. Available: w3techs.com/technologies/overview/content_management/all

[21] WordPress for Android. [Online]. Available: android.wordpress.org

[22] WordPress for BlackBerry. [Online]. Available: blackberry.wordpress.org

[23] Google Authenticator. [Online]. Available: code.google.com/p/google-authenticator

[24] Facebook SDK for Android. [Online]. Available: github.com/facebook/facebook-android-sdk

[25] Facebook SDK for BlackBerry. [Online]. Available: us.blackberry.com/developers/started/facebook.jsp

[26] Understand Your Code. [Online]. Available: scitools.com

[27] CommonsWare Android Components. [Online]. Available: commonsware.com/cwac

[28] Hello, World - Android Developers. [Online]. Available: developer.android.com/resources/tutorials/hello-world.html

[29] StatSVN - Repository Statistics. [Online]. Available: statsvn.org

[30] git-log. [Online]. Available: kernel.org/pub/software/scm/git/docs/git-log.html

[31] ChurnExtension - Mercurial. [Online]. Available: mercurial.selenic.com/wiki/ChurnExtension

[32] R. Lind and K. Vairavan, "An experimental investigation of software metrics and their relationship to software development effort," *Transactions on Software Engineering*, vol. 15, no. 5, pp. 649–653, May 1989.

[33] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Towards a theoretical model for software growth," in *Proceedings of the International Workshop on Mining Software Repositories (MSR)*, 2007, pp. 21–28.

[34] P. Kampstra, "Beanplot: A boxplot alternative for visual comparison of distributions," *Journal of Statistical Software, Code Snippets*, vol. 28, no. 1, pp. 1–9, Nov. 2008.

[35] Java Script Object Notation. [Online]. Available: json.org

[36] Android Package Index. [Online]. Available: developer.android.com/reference/packages.html

[37] BlackBerry JDE 5.0.0 API Reference. [Online]. Available: blackberry.com/developers/docs/5.0.0api/index.html

[38] BlackBerry JDE 6.0.0 API Reference. [Online]. Available: blackberry.com/developers/docs/6.0.0api/index.html

[39] Jakarta Regexp. [Online]. Available: jakarta.apache.org/regexp

[40] android-xmlrpc. [Online]. Available: code.google.com/p/android-xmlrpc

[41] kXML-RPC. [Online]. Available: kxmlrpc.objectweb.org

[42] "The State of Mobile Apps," Nielsen Company. [Online]. Available: blog.nielsen.com/nielsenwire/online_mobile/the-state-of-mobile-apps