# A Performance Evaluation of Cross-Platform Mobile Application Development Approaches

Xiaoping Jia
School of Computing
DePaul University, Chicago, IL, USA
xjia@cdm.depaul.edu

Aline Ebone
Polytechnic School
Unisinos, São Leopoldo, Brazil
aebone@edu.unisinos.br

Yongshan Tan
School of Computing
DePaul University, Chicago, IL, USA
ytan17@mail.depaul.edu

## ABSTRACT

There is a wide range of different approaches and tool for cross-platform mobile application development. We studied the performance characteristic of the mobile applications developed with a number of common approaches and tools for mobile application development, including the native SDK's of Google Android and Apple iOS, and cross-platform tools of Apache Cordova, Microsoft Xamarin, and Appcelerator Titanium. The data reveal insights to the designs and trade-offs of different approaches and offer guidance in selecting the appropriate approaches based on their respective performance characteristics.

## 1 INTRODUCTION

One of the most exciting developments in computing technologies in recent years is the rapid advances in mobile computing and the tremendous growth in the popularity of mobile applications targeting smart phones, tablets, and other wearable or embedded devices. With the ever-increasing hardware capabilities of such devices, mobile software applications are becoming ever more sophisticated and complicated. At the same time, mobile applications must deal with some unique constraints and requirements, such as high responsiveness, low memory consumption, and low power consumption. To complicate the matter further, currently, there are a number of competing mobile platforms on the market, led by Android and iOS. It is highly desirable for developers to have their applications running on all popular mobile platforms. Unfortunately, the competing platforms, while they are comparable in capabilities, are very different in the programming languages and the Application Programming Interfaces used. It is rather expensive and time consuming to port mobile applications from one platform to another. There are a number of approaches to support *cross-platform* mobile application development today. They vary significantly in the technologies, the cost, and the performance characteristics of the applications produced. There are limitations and shortcomings in each of the approaches.


MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5712-8/18/05…$15.00
https://doi.org/10.1145/3197231.3197252

## 2 APPROACHES TO CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT

### 2.1 Native Applications

One way to bring a mobile application to multiple platforms is to develop a *native application* for each platform using the respective native SDK, such as Xcode for iOS, Android Studio for Android, etc. Native applications have full and direct access to the native platform functionalities and are capable of delivering the best performance and user experience on each platform. However, this approach affords minimum code reuse, and is the most expensive option in cost and time. Native applications developed using both native SDK's of Android and iOS are included in our study.

### 2.2 Cross-Platform Applications

The cross-platform mobile application development approaches allow a single code base to target multiple mobile platforms. The most significant advantages of these approaches are the cost and time savings to target multiple platforms. A representative from each of these approaches is included in our study.

*2.2.1 Packaged Web Apps/Hybrid Apps.* A popular and economical approach for cross-platform mobile applications is to develop mobile applications using the standard web technologies (HTML, CSS, and Javascript) which are supported by all mobile platforms. For this category, applications developed using Apache Cordova [1] are included in our study.

*2.2.2 Proxy Native Apps.* Appcelerator Titanium [2] is a JavaScript based framework and tool suite aims to remediate some of the shortcomings of the packaged web app approach by designing a unified and higher level, i.e., more abstract, JavaScript API for building native UI. Implementations of the unified JavaScript API

are provided for several popular mobile platforms, including Android and iOS. Applications built using this approach can look and behave exactly as native apps.

*2.2.3 Cross-Compilation with Native API Binding*. Another approach is to define API bindings to allow programs in one language to consume the libraries of a different language, and to cross-compile application source code to binaries of different target platforms. Xamarin [3] is an example of this approach, which allows mobile applications to be developed in C# and then compiled into binaries for several mobile platforms. With the native API binding and compiling source code to binaries of the target platform, this approach can build mobile apps that look and behave exactly as native apps and deliver performance close to native apps.

## 3 DESIGN OF THE EXPERIMENT

A set of mobile apps with identical features using different SDK's and tools were developed to measure their performance characteristics on the same devices under identical conditions, we have designed three benchmark mobile apps, each focusing on a different aspect of app performance. Each of the benchmark apps consists of two screens: the initial screen is a configuration screen for setting the parameters of the benchmark app. The second screen is the main screen which displays a screenful of contents of variable sizes. The size of the contents is controlled by one or more parameters of the benchmark app.

### 3.2 Implementations

- [NA] Native Android app using Java and Android Studio,
- [FA] Xamarin Forms Android app using C# and XAML,
- [XA] Xamarin Android app using C# and XML,
- [CA] Apache Cordova Android app using JS and HTML,
- [TA] Titanium Android app using JS and HTML,
- [NI] Native iOS app using Swift and Xcode,
- [FI] Xamarin Forms iOS app using C# and XAML,
- [CI] Cordova iOS app using JavaScript and HTML,
- [TI] Appcelerator Titanium iOS app using JS and HTML.

Xamarin Forms apps FA and FI share the same code base, but are packaged differently for their respective target platforms. So are the Apache Cordova apps CA and CI, and the Appcelerator Titanium apps TA and TI.

Despite our best effort to implement the benchmark apps as similarly as possible using different approaches and tools, there are some noticeable differences in the implementations due to the inherent differences in the available API and the design of the different approaches.

## 4 THE INITIAL DATA & OBSERVATIONS

*Observations on the building time*. The building time of the native apps, NA and NI, and the Xamarin apps, FA, FI, and XA, tend to grow proportionally with respect to the size of the view. The building time of the Apache Cordova apps, CA and CI, and the Appcelerator Titanium apps, TA and TI, grow at a significantly slower rate. This suggests that the native apps and the Xamarin apps behave similarly in constructing a view hierarchy that represents the entire view before any portion of the view is rendered, while the Apache Cordova apps and Appcelerator Titanium apps only construct a partial structure that represents a portion of the view that is visible.

*Observations on the rendering time*. Ideally, only the portion of the view that is visible needs to be rendered. Therefore, an optimal implementation would exhibit a near constant rendering time regardless of the size of the view. Only the Android native app, NA, and the Xamarin Android app, XA, exhibit near constant rendering time with respect to the size of the view. The iOS native app NI, Apache Cordova Android apps, CA and CI, and the Appcelerator Titanium apps, TA and TI, behave similarly with a slow rate of increase of rendering time with respect to the size of the view. The Xamarin Forms apps, FA and FI, are significantly slower for larger views compared to the last group of apps. The Apache Cordova apps, CA and CI, exhibit meaningfully different patterns of rendering time on the Android and iOS platforms. This difference may be attributed to the differences in the underlying HTML rendering engines and the JavaScript engines of the respective platforms.

*Observations on the total UI response time*. The total UI response time of the native apps, NA and NI, the Xamarin Android app, XA, and the Appcelerator Titanium apps, TA and TI, exhibit similar patterns with slow rate of growth with respect to the size of the view. The Xamarin Forms apps, FA and FI, are significantly slower for larger views compared to the last group of apps. The Apache Cordova apps, CA and CI, exhibit significantly different patterns of UI response time on the Android and iOS platforms.

*Observations on memory usage*. The memory usage of all apps generally increases proportionally to the size of the UI, with the exception of the Apache Cordova apps, CA and CI. The memory usage of the Apache Cordova apps remains roughly constant regardless of the size of the UI. This suggests that the Apache Cordova apps do not maintain a persistent structure representing the UI after the rendering is complete. The may also be the reason that Apache Cordova apps are able to support the UI of the largest sizes compared to other approaches.

*Observations on app size*. The sizes of the native apps, NA and NI, are the smallest on both platforms. The sizes of cross-platform apps are significantly larger than the sizes of the native apps on both Android and iOS platforms.

## 5 CONCLUSIONS

Our experiment shows that there are significant differences in the performance characteristic of the benchmark apps developed using different approaches. The data collected from the experiment offers insights to the designs and trade-offs of the different approaches. These insights offer guidance to the practitioners when selecting among the different approaches.

## REFERENCES

[1] Apache Software Foundation. Apache Cordova, 2018. https://cordova.apache.org/.
[2] Appcelerator Inc. Titanium, 2018. http://www.appcelerator.com/.
[3] Xamarin Inc. Xamarin Platform, 2018. https://www.xamarin.com/.