

# COMS E6156 Project Report

## Evaluating iOS app development frameworks

Swift Reaction to Xamarin

Pratyush Nalam, pn2278 | Gaurav Ragtah, gr2511

May 5, 2016

## 1 Problem Statement

Smartphones and app stores are a multi-billion dollar industry. 61% of the phone-owning population in the United States uses a smartphone. An estimated 1 billion smartphones are projected to be sold in the next couple years. In this huge market, Android and iOS are the main competitors, and most apps are built to run on one of these two platforms. By the nature of competition and rivalry, these two operating systems have an exclusionary approach to one another, resulting in a 2-fragmented app developer experience. Various mobile development frameworks for cross-platform app solutions have sprung up to target this opportunity. In this project, we focus on iOS and explore the new, native Apple Swift language bundled with XCode and the hot, upcoming cross-platform app development framework, Xamarin. We wish to compare the two holistically as mobile app frameworks, and also weigh in on whether non-native and cross-platform mobile development is a feasible option today.

## 2 Background

### 2.1 Launch of the iPhone and App Store

Steve Jobs introduced the iPhone to the world in 2007 with much fanfare. While there were other smartphones in that era, the iPhone had one major differentiating feature – a capacitive touchscreen. While competitors still had resistive touchscreens that led to a poor user experience while using styluses, the iPhone’s finger-friendly capacitive touchscreen massively simplified operation of the phone. This combined with the marketing juggernaut of Apple saw the iPhone absolutely take off.

In 2008, with the iPhone 3G and iPhone OS 2.0, Apple introduced the App Store, an online storefront for third-party developers to produce and publish their own apps for owners of the iPhone to use. Combined with the proliferation of 3G in the United States and its dropping to a favorable price point, iPhone’s popularity skyrocketed. More and more people started developing apps for the iPhone.

### 2.2 Android and the platform problem

Google launched Android, a competing mobile operating system, at around the same time and in the following years, gained massive popularity due to it being available at relatively

lower price points. The other factor was heavy backing from Verizon, in order to counter its competitor's hold (AT&T) on the iPhone. With its rapidly rising market share, developers began flocking to the platform and mobile app development was seeing the rise of a big issue.

Independent developers, who had limited resources, were forced to compromise as they sought to make apps for both the iPhone and Android platforms. The fact that they used different languages – iOS used Objective-C and Android used Java – complicated the problem. For larger companies as well, reaching both the Android and iOS user bases was important because Android covered a much larger number of users and provided good traction for apps, while iOS users tended to be more likely to buy paid or freemium apps and provide an easy revenue source. The situation was ripe for exploitation by an efficient, cross-platform mobile development solution.

## 2.3 Early solutions

Around 2010, a new class of cross-platform frameworks hit the market. This was the time of web apps and an increased focus on web development, and unsurprisingly, the earliest frameworks were all based on JavaScript. Cordova and PhoneGap were some of the first to hit the market. This phenomenon was based on the assumption that since JavaScript was one of the most widely used languages – most repositories on the popular source code repository website, GitHub are written in JavaScript – it was surely the best suited for cross-platform development going forward. It had a huge developer community and extensive documentation as well as tutorials. JavaScript and web-apps were touted as the future. In fact, Facebook rewrote its apps for iOS and Android as web-apps, further lending credibility to this trend.

Soon, however, things took a turn for the worse. Apps developed using these frameworks had extremely poor UI/UX and this frustrated users, who could clearly see degraded experiences from the native ones. API access wasn't complete either and this led to suboptimal compromises in developing such apps. Users were unhappy, and this led to developers switching back to writing native apps for the fragmented market. Indeed, Facebook notably rewrote its aforementioned iOS and Android apps using native libraries to address widespread user criticism.

## 2.4 The rise of “hybrid” frameworks

This brought the situation back to square one. The problem of building “once” (at a high level, at least) for multiple platforms still remained. JavaScript frameworks – at least in their current version, were not working out and a huge need was felt for an alternative kind of system.

This led to a new “hybrid” approach being tried out. On the backend, the system ties in natively with low-level and system API calls, and on the front end, cross-platform UI and framework components that are familiar to developers are used. In this way, it is kind of a “best-of-both-worlds” approach: familiar components with native-level performance to make life easier for developers.

Xamarin is one such “hybrid” cross platform app-development framework, and in the next section, we will explore Xamarin as well as native Swift.

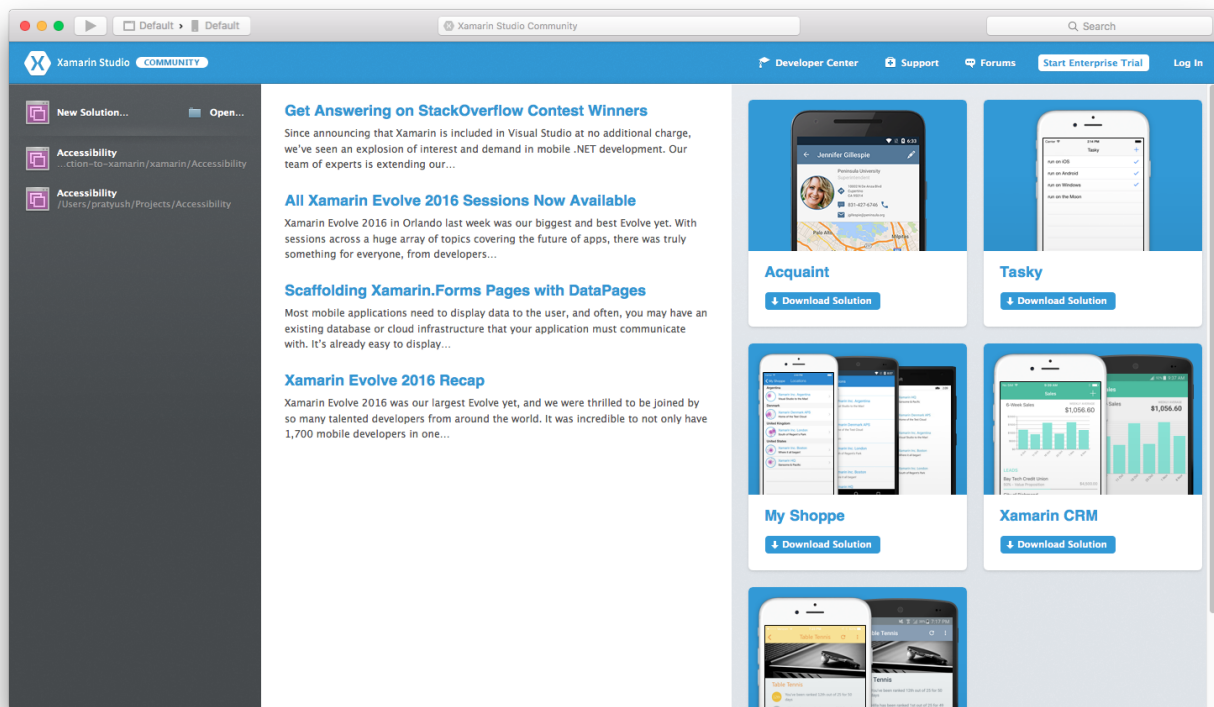


Figure 1: Xamarin Studio for OS X

## 3 Evaluating Frameworks

### 3.1 Xamarin

#### 3.1.1 Introduction

Xamarin is a cross-platform app development framework which follows the “hybrid” philosophy as described in the previous section. Its language of choice is C#. Xamarin decided to go the C# route because C# and, in turn, .NET has a huge ecosystem of developers and an extensive support documentation structure already in place that provides great leverage. It also wanted to attract the huge demographic of Windows desktop developers to mobile app development by promising them the same kind of tools and frameworks at their disposal. It supports developing apps for iOS, Android, and Windows (the new app development model introduced in 2012). It was recently acquired by Microsoft and was subsequently made completely free for individuals (important to note that it was already free for students). The app written in C# is translated to native iOS-level executable code.

#### 3.1.2 Installation

Installing Xamarin was a breeze on both OS X and Windows. We installed Xamarin Studio for both operating systems and its plugins from <https://www.xamarin.com/platform>. This installs the various C# libraries and compilers needed to run iOS apps. It obviously requires Xcode to be installed on the system as Xcode contains all the various simulators, assembly packages, and toolkits required to actually build and run iOS apps. Figure 1 shows Xamarin Studio for OS X on startup.

```
// you can also set the desired accuracy:
iPhoneLocationManager.DesiredAccuracy = 5; // 1000 meters/1 kilometer
// you can also use presets, which simply evaluate to a double value:
//iPhoneLocationManager.DesiredAccuracy = CLLocation.AccuracyNearestTenMeters;

// handle the updated location method and update the UI
if (UIDevice.CurrentDevice.CheckSystemVersion (6, 0)) {
    iPhoneLocationManager.LocationsUpdated += (object sender, CLLocationUpdatedEventArgs e) => {
        UpdateLocation (mainScreen, e.Locations [e.Locations.Length - 1]);
    };
} else {
    #pragma warning disable 618
    // this won't be called on iOS 6 (deprecated)
    iPhoneLocationManager.UpdatedLocation += (object sender, CLLocationUpdatedEventArgs e) => {
        UpdateLocation (mainScreen, e.NewLocation);
    };
    #pragma warning restore 618
}
```

Figure 2: Code snippet showing the calls to the location API

```
static public void UpdateLocation (IMainScreen ms, CLLocation newLocation)
{
    var watch = System.Diagnostics.Stopwatch.StartNew ();

    ms.LblAltitude.Text = newLocation.Altitude.ToString () + " meters";
    ms.LblLongitude.Text = newLocation.Coordinate.Longitude.ToString () + "°";
    ms.LblLatitude.Text = newLocation.Coordinate.Latitude.ToString () + "°";
    ms.LblCourse.Text = newLocation.Course.ToString () + "°";
    ms.LblSpeed.Text = newLocation.Speed.ToString () + " meters/s";
    ms.mapView.ShowUserLocation = true;

    watch.Stop ();
    Console.WriteLine (watch.ElapsedTicks);
}
```

Figure 3: Code snippet showing the profiling code inserted to calculate the time taken to make the API calls

### 3.1.3 App

In our app itself, we created a basic MapView to poll the device's current location via GPS, and also other parameters like heading with respect to North that are obtained from the readings of the gyrometer, inclinometer, and magnetometer sensors using Apple's Core Motion packages and the MapKit APIs which were fully implemented in Xamarin. We added some extra profiling code in order to calculate how much time it took for the location to be polled. Figures 2, 3, 4, 5 show code snippets, the actual app itself, and the 'info.plist' file which enumerates various app parameters.

### 3.1.4 Performance

The results of these calculations are shown in Figure 6. Calculating the ticks elapsed allows us to calculate time taken in relation to the CPU clock speed thus eliminating the factor of the CPU/emulator speed in the calculations. We have also enumerated below some additional results

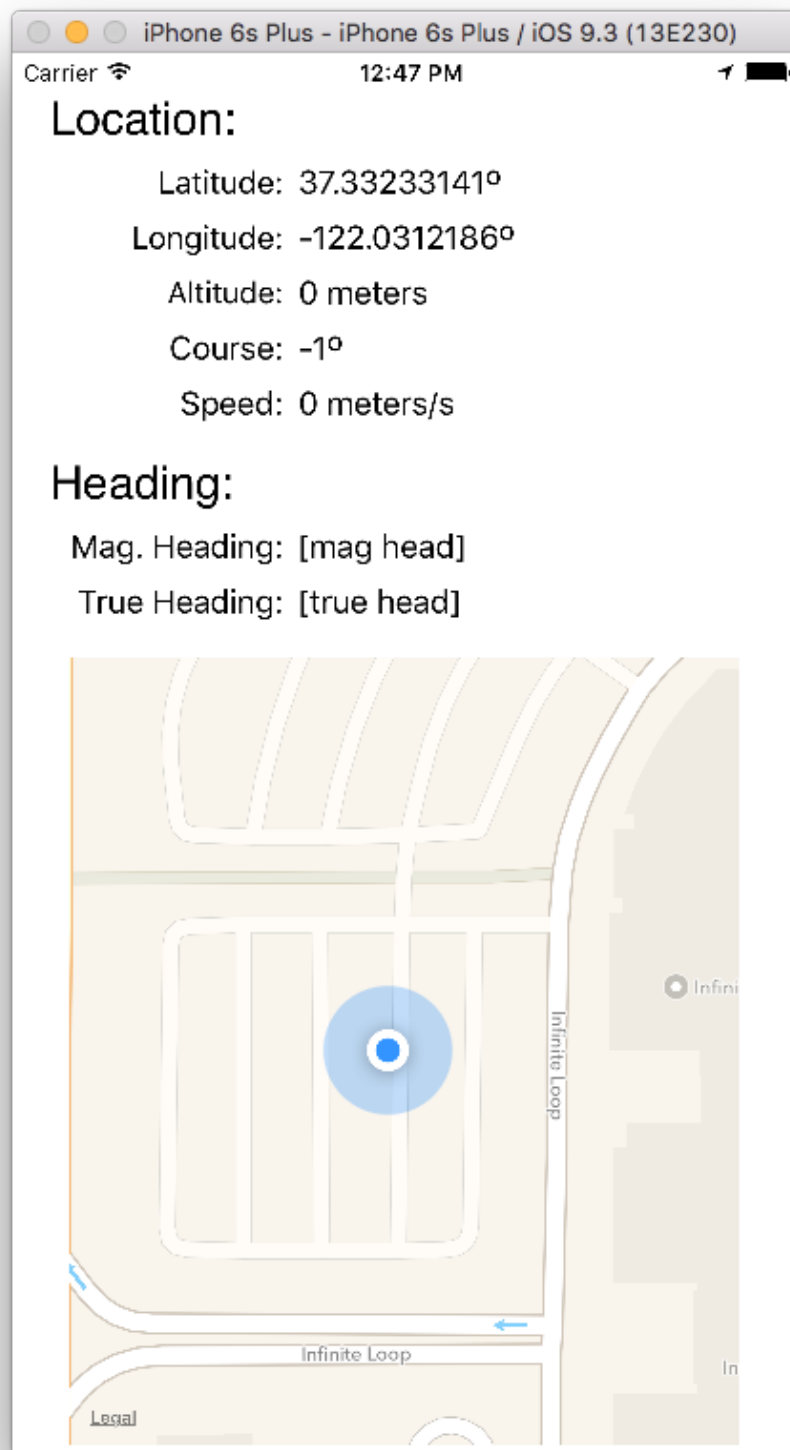


Figure 4: The app developed in Xamarin running on a simulator configured for the iPhone 6S Plus running iOS 9.3

Application Name:

Bundle Identifier:

Version:  Build:

Devices:

Deployment Target:

---

**iPhone / iPod Deployment Info**

Main Interface:

Device Orientations: ☒ Portrait  
☐ Upside Down  
☐ Landscape Left  
☐ Landscape Right

---

**Status Bar Styles**

Style:

Visibility: ☐ Hide during application launch

Figure 5: info.plist for the app

which we obtained.

**Profiling Output:**

```
Location update took 38487 ticks
Location update took 585 ticks
Location update took 976 ticks
Location update took 688 ticks
Location update took 1045 ticks
Location update took 1061 ticks
Location update took 1461 ticks
Location update took 933 ticks
Location update took 883 ticks
Location update took 1565 ticks
Location update took 1601 ticks
Location update took 2790 ticks
Location update took 963 ticks
Location update took 1721 ticks
Location update took 1924 ticks
Location update took 1279 ticks
Location update took 1772 ticks
Location update took 1694 ticks
Location update took 1155 ticks
Location update took 1585 ticks
Location update took 941 ticks
Location update took 939 ticks
Location update took 1731 ticks
Location update took 1007 ticks
Location update took 2120 ticks
Location update took 1326 ticks
Location update took 1021 ticks
Location update took 1062 ticks
Location update took 1424 ticks
Location update took 1746 ticks
Location update took 1001 ticks
Location update took 1182 ticks
Location update took 1815 ticks
Location update took 1615 ticks
Location update took 1527 ticks
Location update took 964 ticks
Location update took 1071 ticks
Location update took 1387 ticks
Location update took 1651 ticks
Location update took 1598 ticks
Location update took 1654 ticks
Location update took 1463 ticks
Location update took 1593 ticks
Location update took 1717 ticks
```

```

2016-05-10 12:05:52.848 Accessibility[13975:264957] 1281
2016-05-10 12:05:53.848 Accessibility[13975:264957] 1264
2016-05-10 12:05:54.850 Accessibility[13975:264957] 1865
2016-05-10 12:05:55.852 Accessibility[13975:264957] 975
2016-05-10 12:05:56.854 Accessibility[13975:264957] 1909
2016-05-10 12:05:57.857 Accessibility[13975:264957] 878
2016-05-10 12:05:58.861 Accessibility[13975:264957] 1072
2016-05-10 12:05:59.865 Accessibility[13975:264957] 1211
2016-05-10 12:06:00.870 Accessibility[13975:264957] 1434
2016-05-10 12:06:01.874 Accessibility[13975:264957] 1207
2016-05-10 12:06:02.878 Accessibility[13975:264957] 1044
2016-05-10 12:06:03.883 Accessibility[13975:264957] 1230
2016-05-10 12:06:04.887 Accessibility[13975:264957] 1206
2016-05-10 12:06:05.890 Accessibility[13975:264957] 1262

```

Figure 6: Ticks elapsed every time the location API is called

```

Location update took 1563 ticks
Location update took 925 ticks
Location update took 1279 ticks
Location update took 1289 ticks
Location update took 972 ticks
Location update took 940 ticks

```

### 3.1.5 Advantages

There were multiple advantages in using Xamarin for developing this app.

- Xamarin is a cross-platform app development framework. This meant that with a few changes, we could easily adapt the app to make it run on Android, as the backend code is all C#.
- The performance of Xamarin apps is comparable, if not similar, to their Swift counterparts. This is a huge win for cross-platform development frameworks.
- The UI components used by Xamarin closely resemble their native counterparts in look, feel, and functionality. This is very important, since user adoption validates the mobile app, and being virtually indistinguishable from a natively developed app in look and performance is a huge milestone.
- Thanks to C#, Xamarin enjoys strong community and documentation support. Also, with its recent success and many big companies starting to use Xamarin, it has become easy and convenient to get development or debugging support when needed.
- With Xamarin, it is also possible to do iOS app development on Windows. Once the Windows machine being used for development is connected to a Mac on the network via ssh, when you press build, the app is sent to the Mac and compiled and tested locally on that machine. This is especially useful for teams, wherein instead of buying a Mac for everyone, there can be one or few Macs for the team which can be used to scale up better for development and testing.



### 3.1.6 Drawbacks

It was not all rosy with Xamarin, however. Some nagging drawbacks we observed were as follows:

- The UI builder was quite clunky and buggy. Components would sometimes not get selected, or mapped correctly to the views, or a bunch of them would get selected together. Working with to design the user flows and experience was a real headache.
- Compared to the Windows version, the IDE didn't feel as fluid on OS X. This is definitely something to be improved upon, and seemed ironic that Apple iOS app development was better on Windows.
- Getting the app to run on a real device for testing involved a lot of work. Xcode itself imposes a lot of restrictions and using a third-party framework further increased the restrictions.

## 3.2 Swift (and Xcode)

### 3.2.1 Introduction

Swift is the native mobile development language provided by Apple to build apps for iOS devices, and is the “official” and “supported” way to do so, through the Xcode IDE. It directly compiles to native iOS-compatible executables. It is a new language and was introduced quite recently to replace the old Objective-C as a development language and framework for the future.

### 3.2.2 Installation

Swift and its components are installed by downloading Xcode - Apple's official developer IDE. We installed Xcode - <https://itunes.apple.com/us/app/xcode/id497799835> from the Mac App Store. This also installs all the required libraries, simulators, and toolchains that are necessary to build apps for iOS and OS X. A point to note is that Apple's ecosystem is set up such that Xcode is needed even if one doesn't use Swift to build the app (for instance, even if you are using Xamarin, Xcode needs to be installed). Figure 7 shows Xcode on startup.

### 3.2.3 App

Our app developed in Swift was similar to the Xamarin version. We created a basic MapView to poll the device location through GPS and also obtained other information like heading that, again, are obtained from the readings of the gyrometer, inclinometer, and magnetometer sensors. This was handled by Apple's Core Motion and MapKit APIs. We set up extra profiling code in order to calculate how much time it took for the location to be polled. Figures 8, 9, 10 show code snippets, the actual app itself, and the 'info.plist' file which enumerates various app parameters.

### 3.2.4 Performance

The results of these calculations are shown in Figure 11. Similar to Xamarin, we calculate the ticks elapsed as this allows us to calculate the time taken in relation to the CPU clock speed and thus eliminate the factor of the CPU/emulator speed in the calculations. We have also enumerated below some additional results which we obtained.

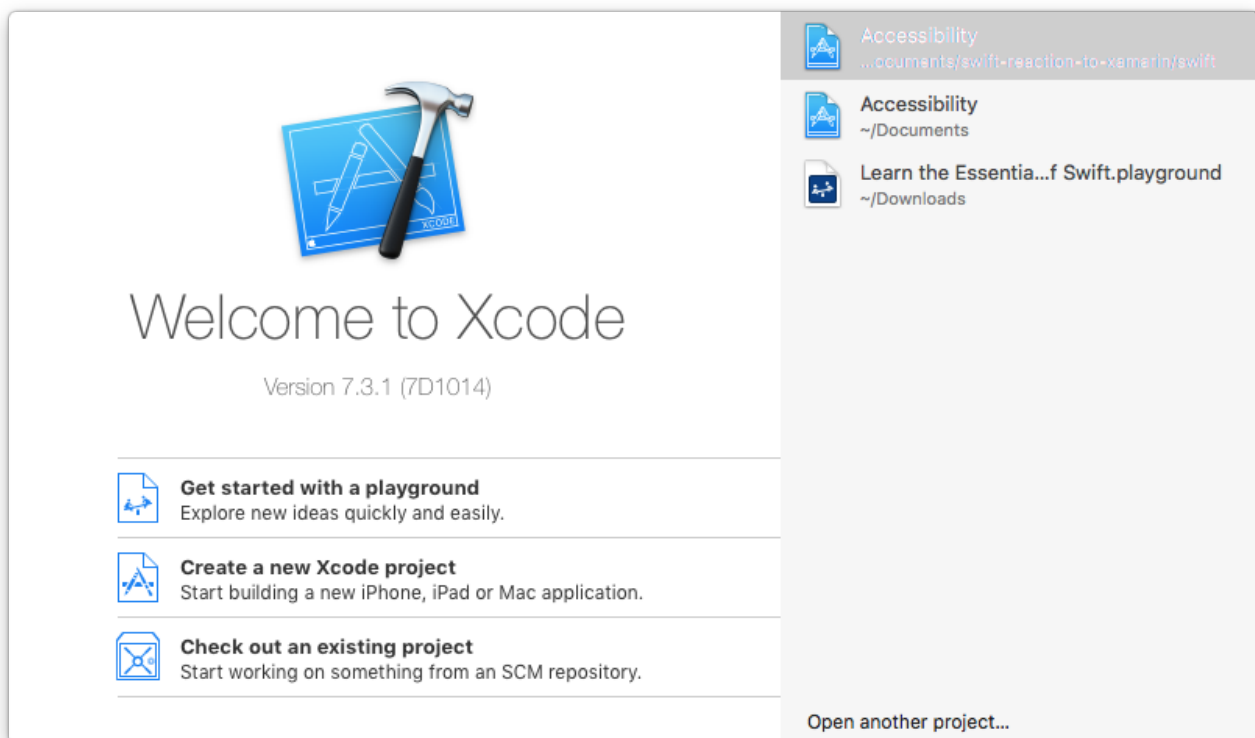


Figure 7: Xcode

```
func locationManager(manager:CLLocationManager, didUpdateLocations locations:[CLLocation]) {
    var t = clock()

    theLabel.text = "\\(locations[0])"
    myLocations.append(locations[0] )

    let spanX = 0.007
    let spanY = 0.007
    let newRegion = MKCoordinateRegion(center: theMap.userLocation.coordinate, span: MKCoordinateSpanMake(spanX, spanY))
    theMap.setRegion(newRegion, animated: true)

    if (myLocations.count > 1){
        let sourceIndex = myLocations.count - 1
        let destinationIndex = myLocations.count - 2

        let c1 = myLocations[sourceIndex].coordinate
        let c2 = myLocations[destinationIndex].coordinate
        var a = [c1, c2]
        let polyline = MKPolyline(coordinates: &a, count: a.count)
        theMap.addOverlay(polyline)
    }

    t = clock() - t

    print("Location update took \\(t) ticks, or \\(Double(t) / Double(CLOCKS_PER_SEC)) seconds of CPU time")
}
```

Figure 8: Code snippet showing the calls to the location API and the profiling code inserted to calculate the time taken to make the API calls

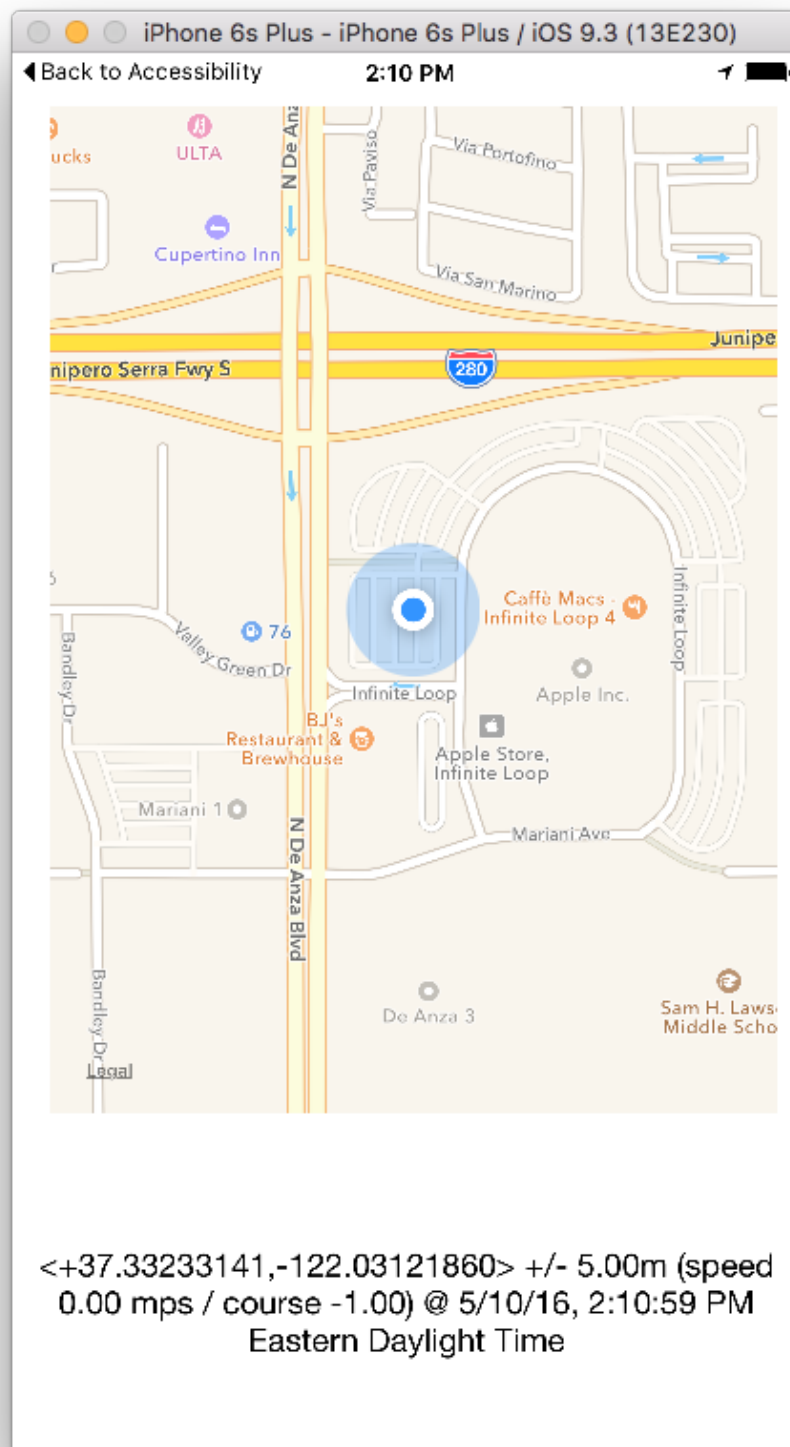


Figure 9: The app developed in Swift on Xcode running on a simulator configured for the iPhone 6S Plus running iOS 9.3

▼ Information Property List	Dictionary	(17 items)
NSLocationAlwaysUsageDescription	String	
NSLocationWhenInUseUsageDesc...	String	
Application Category	String	
Localization native development re...	String	en
Executable file	String	\$(EXECUTABLE_NAME)
Bundle identifier	String	\$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	\$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▼ Required device capabilities	Array	(1 item)
Item 0	String	armv7
▼ Supported interface orientations	Array	(3 items)
Item 0	String	Portrait (bottom home button)
Item 1	String	Landscape (left home button)
Item 2	String	Landscape (right home button)

Figure 10: info.plist for the app

### Profiling Output:

Location update took 31185 ticks, or 0.031185 seconds of CPU time  
 Location update took 1846 ticks, or 0.001846 seconds of CPU time  
 Location update took 939 ticks, or 0.000939 seconds of CPU time  
 Location update took 867 ticks, or 0.000867 seconds of CPU time  
 Location update took 846 ticks, or 0.000846 seconds of CPU time  
 Location update took 1143 ticks, or 0.001143 seconds of CPU time  
 Location update took 633 ticks, or 0.000633 seconds of CPU time  
 Location update took 1038 ticks, or 0.001038 seconds of CPU time  
 Location update took 1326 ticks, or 0.001326 seconds of CPU time  
 Location update took 1199 ticks, or 0.001199 seconds of CPU time  
 Location update took 1427 ticks, or 0.001427 seconds of CPU time  
 Location update took 1107 ticks, or 0.001107 seconds of CPU time  
 Location update took 1745 ticks, or 0.001745 seconds of CPU time  
 Location update took 798 ticks, or 0.000798 seconds of CPU time  
 Location update took 1375 ticks, or 0.001375 seconds of CPU time  
 Location update took 808 ticks, or 0.000808 seconds of CPU time  
 Location update took 1610 ticks, or 0.00161 seconds of CPU time  
 Location update took 808 ticks, or 0.000808 seconds of CPU time  
 Location update took 733 ticks, or 0.000733 seconds of CPU time  
 Location update took 1157 ticks, or 0.001157 seconds of CPU time  
 Location update took 783 ticks, or 0.000783 seconds of CPU time

Location update took 1365 ticks, or 0.001365 seconds of CPU time  
Location update took 1297 ticks, or 0.001297 seconds of CPU time  
Location update took 916 ticks, or 0.000916 seconds of CPU time  
Location update took 818 ticks, or 0.000818 seconds of CPU time  
Location update took 978 ticks, or 0.000978 seconds of CPU time  
Location update took 1494 ticks, or 0.001494 seconds of CPU time  
Location update took 1726 ticks, or 0.001726 seconds of CPU time  
Location update took 1301 ticks, or 0.001301 seconds of CPU time  
Location update took 1296 ticks, or 0.001296 seconds of CPU time  
Location update took 810 ticks, or 0.00081 seconds of CPU time  
Location update took 1188 ticks, or 0.001188 seconds of CPU time  
Location update took 783 ticks, or 0.000783 seconds of CPU time  
Location update took 940 ticks, or 0.00094 seconds of CPU time  
Location update took 1195 ticks, or 0.001195 seconds of CPU time  
Location update took 829 ticks, or 0.000829 seconds of CPU time  
Location update took 898 ticks, or 0.000898 seconds of CPU time  
Location update took 1165 ticks, or 0.001165 seconds of CPU time  
Location update took 798 ticks, or 0.000798 seconds of CPU time  
Location update took 851 ticks, or 0.000851 seconds of CPU time  
Location update took 924 ticks, or 0.000924 seconds of CPU time  
Location update took 792 ticks, or 0.000792 seconds of CPU time  
Location update took 907 ticks, or 0.000907 seconds of CPU time  
Location update took 1059 ticks, or 0.001059 seconds of CPU time  
Location update took 734 ticks, or 0.000734 seconds of CPU time  
Location update took 987 ticks, or 0.000987 seconds of CPU time  
Location update took 974 ticks, or 0.000974 seconds of CPU time  
Location update took 779 ticks, or 0.000779 seconds of CPU time  
Location update took 1050 ticks, or 0.00105 seconds of CPU time  
Location update took 993 ticks, or 0.000993 seconds of CPU time

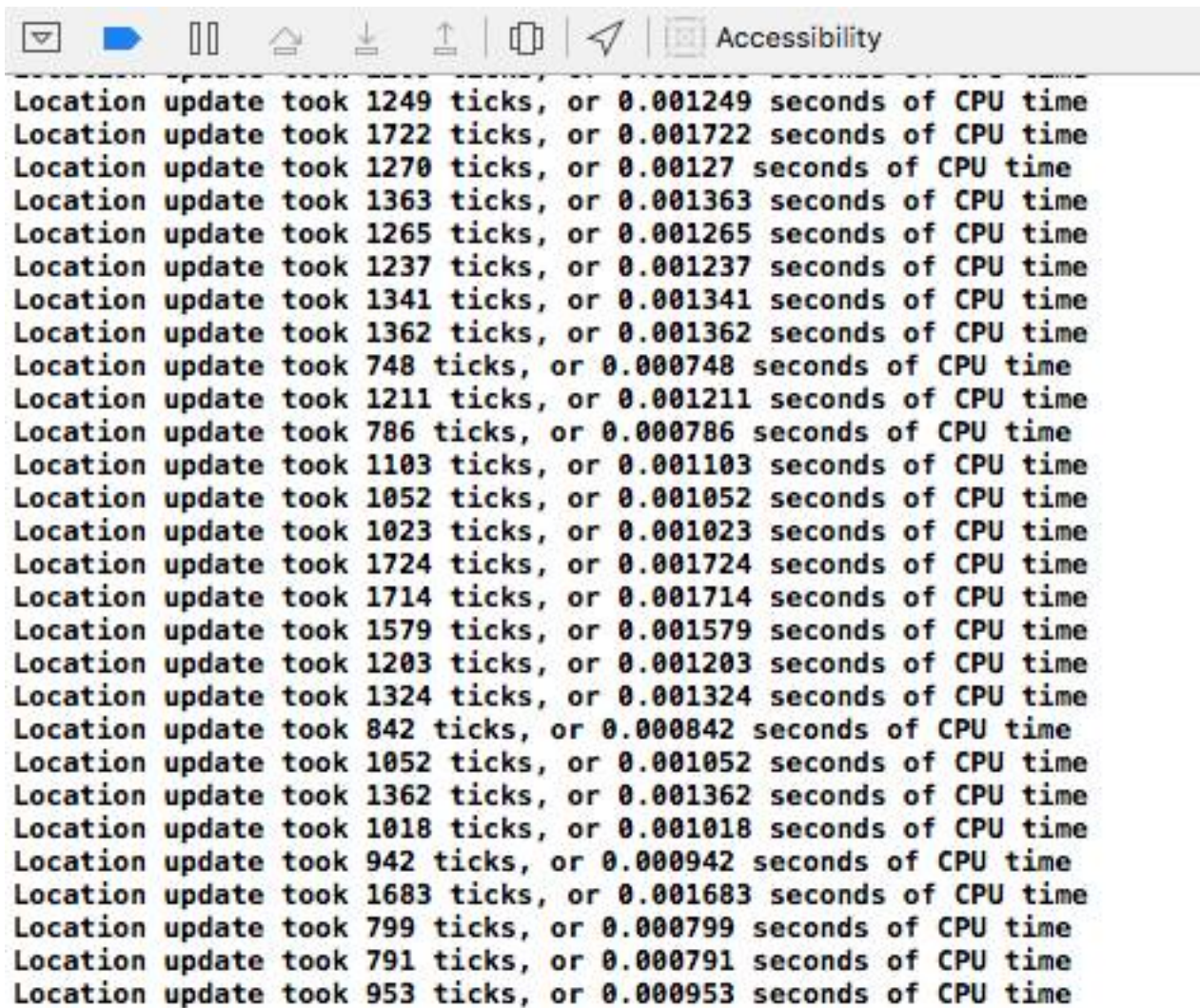
### 3.2.5 Advantages

There were several advantages in picking Swift for app development.

- Swift is native code. There is no translation involved and this leads to the best performance theoretically possible.
- Swift is a new language and special focus was given to code readability. More importantly, there are no semi-colons.
- Handling of null pointers is safer and better implemented in Swift. Also, it has strong typing and generics.
- Thanks to being native, Swift apps enjoy better memory management than their Xamarin counterparts.

### 3.2.6 Drawbacks

However, Swift also suffers from a few key drawbacks.



Event	Ticks	CPU Time (seconds)
Location update	1249	0.001249
Location update	1722	0.001722
Location update	1270	0.00127
Location update	1363	0.001363
Location update	1265	0.001265
Location update	1237	0.001237
Location update	1341	0.001341
Location update	1362	0.001362
Location update	748	0.000748
Location update	1211	0.001211
Location update	786	0.000786
Location update	1103	0.001103
Location update	1052	0.001052
Location update	1023	0.001023
Location update	1724	0.001724
Location update	1714	0.001714
Location update	1579	0.001579
Location update	1203	0.001203
Location update	1324	0.001324
Location update	842	0.000842
Location update	1052	0.001052
Location update	1362	0.001362
Location update	1018	0.001018
Location update	942	0.000942
Location update	1683	0.001683
Location update	799	0.000799
Location update	791	0.000791
Location update	953	0.000953

Figure 11: Ticks elapsed every time the location API is called



- It is iOS only. Since it is provided directly by Apple, it is not going to be cross platform anytime soon.
- Due to it being a very recent language, there is not much documentation and community support available and debugging can be quite a challenge.
- Due to it being so new, Apple is still iterating on various aspects of the language spec and we often encounter tutorials or documentation for older versions which is an irritating, as more often than not, backward compatibility is absent.
- Compiler and run-time error messages are still either minimal or very cryptic and more work is needed in this area.

### 3.3 User Experience

We loaded these apps onto a friend's iPhone and asked 15 iPhone owners among our friends to try them out. Our friends were tasked to identify, not guess, which app was native. Not a single one was able to identify the iOS app. Next, we let them guess (to allow for intuition and subconscious factors), and the split was about half and half – 7 friends guessed the Xamarin app was native, while the other 8 guessed correctly. This underscores the success of Xamarin in developing iOS apps. Look and fluidity of the app was comparable in both versions.

### 3.4 Performance

As we have seen, performance of both the frameworks is quite similar, although Swift edges out Xamarin. Xamarin gave us an average elapsed ticks value of 2093.82. Swift gave us an average elapsed ticks value of 1664.4. However, this is a trivial difference for most real-world use cases, since we're talking about 0.002093 s vs 0.001664 s – insignificant to human users. This close-to-native performance has led to a huge momentum and traction for Xamarin as the choice for cross-platform development.

### 3.5 React Native

We had initially also attempted developing this app in React Native, given the media hype around it. However, this was quite a challenge. React Native is an iOS/Android app development framework developed by Facebook which is based on their own JS library called react.js. It is quite non-trivial for beginners to get started, even with previous mobile development or javascript experience. It also suffers from poor documentation due to it being a very new framework. There are very few code samples and tutorials, making it even harder to get started. Moreover, all components of the API are not yet mapped to native and it remains in active development phase. Given this, it made more sense to just compare Xamarin and Apple Swift.

## 4 Division of Labor

Gaurav mainly worked on the Swift native app while Pratyush worked on the Xamarin equivalent. We pair programmed and collaborated for set up, testing, and debugging etc of both apps. Each of us went through and ran the two apps locally on our machines to familiarize ourselves with the code and for sanity checks. We also both individually tried building a React Native version, but the efforts were not fruitful.

## 5 Challenges

Developing in Swift was quite a challenge due to the recentness of the language, scarcity of documentation, and the cryptic nature of the errors. Xamarin had its own share of C# quirks which took some time getting used to. Also, the Xamarin UI builder was quite a pain to work with. Also, we had to familiarize ourselves with the iOS app components and system and UI libraries to make sure the apps were equivalent in terms of code structure and flow, while being relatively new to the frameworks ourselves.

## 6 Deliverables

All code has been uploaded on GitHub and can be accessed at <https://github.com/pratnala/swift-reaction-to-xamarin>. It is organized into the equivalent Swift and Xamarin app code by subdirectory structure.

## 7 Final Thoughts

Working on this project has been a huge learning experience for the two of us. It was our first foray into the world of mobile app development and we gained a lot of insight on the intricacies involved, coming from a web app and data engineering background. We had also initially planned to build a similar app in React Native, as mentioned earlier, but its steep learning curve coupled with an unforeseen increase in time spent debugging the Swift app forced a change in plan. Given our results through the endeavors of this project, we are strongly convinced that Xamarin is poised to take off as the future of cross-platform framework for mobile app development, and the recent Microsoft acquisition now seems like a really smart, well-planned move.