A closer look and comparison of cross-platform development environment for smartphones

Erik Johansson  
Bachelor of Computer Science  
Mälardalens University  
Ejn11015@student.mdh.se

Tobias Andersson  
Bachelor of Computer Science  
Mälardalens University  
Tan10006@student.mdh.se

Mentor: Caroline Uppsäll  
Lecturer  
Mälardalens University  
Caroline.Uppsäll@mdh.se

Examiner: Rikard Lindell  
Senior Lecturer  
Mälardalens University  
Rikard.Lindell@mdh.se

ABSTRACT

A problem with having a fast and wide production of different platforms for mobile devices is that you can’t code for one and deploy on all devices at the same time. This thesis is focused on cross-platform development environments for smartphones and mainly to see what options there are on the market. This report will investigate how well a cross-compiler solution compares to hybrid cross-platform development. To do this we took a closer look at their architecture and then compared this with the results from different tests made. All the tests were made on the same smartphone to ensure fairness between them. All the tests strive to be as equal as possible even though the languages might differ from each other. The tested frameworks were PhoneGap, Qt, Unity3D and GameMaker. The different tests were about performance, power consumption, difficulty in accessing web browsers to perform HTML parsing and lastly to see if the platforms can access different native APIs such as the camera and accelerometer. The previously mentioned topics were compared between all the frameworks. We also compared the documentation found on their webpage to figure out which is the easiest to get started on.

Keywords

Keywords: Cross-Platform Cross-Compiler, iOS, Android, Windows Phone, Mobile devices, Unity3D, C/C++, Qt, PhoneGap, Game Maker.

1. INTRODUCTION

The smartphone is used worldwide. In 2012, there were more than a billion people using a smartphone, and it is believed 2014 will bring it up to 1.75 billion [4]. The rapid growth of smartphone usage indicates that the smartphone application market is becoming increasingly important. However, the market is fragmented, which means that if you as a developer want to create an app for every mobile device you will have to learn each language that pertains to each specific phone type (Android, iPhone or Windows phone). In an article from 2011 it is explained that a developer which is experienced at Java might be less experienced at Objective-C, therefore it could be very difficult for smaller companies, with one or two developers, to reach several platforms [21]. Could a cross-platform framework solve this problem?

2. Background

To program for different devices such as iPhone, Windows phone and Android you need to have knowledge about their specific languages. To be able to program for iPhone you need to know Objective-C, for Windows phone there is C# and for Android you have Java [24].

A similarity between all three devices is that they can all use C++ as well as their well-known main languages mentioned above [7, 10, and 11]. The devices are mainly meant to use C++ for high CPU-using parts of the application, while the UI (User Interface) is built in the respective native languages (i.e. Java for Android). This means that parts of the code can be copied from one device to another, but not all [20].

Research comparing the security levels between the iOS and Android mobile devices, concludes that iOS is the safer device. One aspect that makes it safer is the vigorous checking of each and every app done by app store [14]. Li and Clark explain in their article on mobile security that both iOS and Android isolate each application from accessing unauthorized data. There is however still a problem, because these applications can make requests to access this dangerous data and the kernel as well as the user can grant these requests the necessary permission [22]. To quote Li and Clark, “users have little trouble granting dangerous permissions when installing free Android applications” [22], which explains why iOS could be considered safer.

In order to reach out to iPhone, Windows Phone and Android at once with one programming language, a developer can use a cross-platform development environment. Cross-platform development can be explained with help from Zhuang, Baldwin, Antuña, Yazir and Ganti. They explain how they can access functionalities on several different operating systems (such as Android, iOS and Windows Phone) with help from one API, PhoneGaps API (PhoneGap is the most popular cross-platform according to an article written in 2012 [8]) [6]. This lets programmers code once and then deploy on said operating systems. To quote Zhuang, Baldwin, Antuña, Yazir and Ganti, “programmers build application logic using JavaScript, and the PhoneGap API handles communication with the native operating
system” [6]. This means that one person with knowledge of the language JavaScript can program for several different operating systems. Although all cross-platforms doesn’t use JavaScript, the web based platforms do.

A few examples of programs that are not mobile applications and can reach several platforms would be Microsoft Word and iTunes, as they are available on both Windows and Mac OS X [13]. Another example of cross-platforming outside of mobile development would be the program CMake, which is an open-source cross-platform build system [19]. Wojcik and Knoll recommend CMake and believe other developers can benefit from using it in the future [16]. Researchers have made attempts at teaching students how to program for both embedded environments and PC at the same time and their results were positive as the students seemed to learn faster with a direct touch to the hardware [17].

With the growing interest in the entire idea of cross-platform programming [8], one must be mindful of the downsides to the chosen cross-platform development environment. An example would be the cross-platform tool MoSync. In 2012 MoSync did not support access to simple camera functions [5]. Due to this fact a developer might avoid this cross-platform tool. This means that if a developer wants to use cross-platform development environments to make their application, research about what they can do could prove necessary. If the camera is a necessary part of a certain application, a cross-platform that cannot access the camera would not suffice for the developer. An article from 2012 mentioned PhoneGap as the most popular cross-platform, although it also mentions that 27% of the developers using PhoneGap are planning to leave it. They claim the framework is poorly made for debugging and runtime performance, while the UI capabilities are lacking [8]. Raj and Tolety took a closer look at cross-platform development to see what benefits and drawbacks there were to the entire concept, and came to the conclusion that as long as cost and time to market is a critical factor, it is worth it [3].

Recent research compares different cross-platform mobile development environments [5, 2 and 3]. Some research compared the cross-platform development environments PhoneGap, Rhodes, DragonRad and MoSync, and concluded PhoneGap had most access to the native API [5]. A study was made in 2011 where PhoneGap only supported a handful of features [2] and another in 2012 where PhoneGap could almost reach everything a native language could reach [5]. This points to the vast development for cross-platforming tools. The only native API it couldn’t use in 2012 was the menu [5].

There has been research to compare the power, memory and CPU usage between some cross-platform development environments. A comparison was made between PhoneGap and Titanium (a similar cross-platform tool as PhoneGap [12]), as well as, PhoneGap with a few additional SDKs. The conclusion was that PhoneGap is far ahead in power, memory and CPU usage compared to Titanium. However, adding some SDKs to PhoneGap to make it easier to make the UI slowed PhoneGap down on power, memory and CPU, and on the tests with many SDKs it even fell behind Titanium on all counts [1].

A cross compiler was discussed in the article “Cross-Compiling Android Applications” [9]. This allows a programmer to reach the different operating systems in another way. The example used is XMVM [15] which compiles java into C/C# to let their code reach more devices. For instance, a program made from java could be compiled into the programming language C and then distributed on C based platforms.

2.1 MAWA AB

This thesis was presented by MAWA AB which is located in Västerås, Sweden. The company’s main business is consulting work, application development and in recent time mobile applications. Thus this thesis was assigned for us to find the best cross-platform tool for their company. The goal is to have everyone work under the same coding language and be able to deploy on all devices.

2.2 Research Objective

The objective of this research is to investigate cross-platform development environments for MAWA AB. The main part of the research will go deeper into the architecture of the different cross-platform development environments, and then make a comparison between the different platforms in regards to battery time, native accessibility, performance and so on.

The main restriction will be the programming language. There are no web developers in this company, they only have knowledge of C/C++, C#, Java and some Objective-C. This means that cross-platforms using these languages will be studied closely and then compared to see which will have the best performance for what MAWA AB intends to do with them.

The state of cross-platform research today is very much based on web based application and very little about trying to emulate a real application. So is it possible to compete with the web based solutions with any cross-platform development environment fitting the requirements laid down by MAWA AB?

3. Method

A series of tests will be run on a number of cross-platform development environments to see how well the frameworks compare. The different cross-platform environments will be Unity3D, Qt, PhoneGap and GameMaker.

All the different tests on the cross-platform environments will be done on the same smartphone to make sure the results are as fair as possible. There will be tests to see delays during runtime i.e. how long the delay is between pressing a button and when the result from pressing the button shows up, as a test of performance on the framework. To see how much battery each framework takes by itself, a test will be performed using the framework’s most standard shell and have it run for a period of time. How susceptible to HTML parsing and submitting to webpages is each platform? The documentation found on each of their webpage will be compared to see how well documented the API is for each framework. A comparison between the framework’s different ways of handling the actual cross-platforming will be performed to get a better understanding. Lastly, the results from these tests will be compared to an application made in their native language (i.e. Java for Android) to see what gains and losses there are compared to the native approach.

4. Case Study

Each test will be made on the same Android smartphone device, to make sure that all tests are fair. The API documentation part
will be somewhat subjective, as it will include the writer’s opinion of how well documented the frameworks are.

4.1 PhoneGap

Although PhoneGap uses JavaScript as their language of choice the background study shows that it is one of the favorites among developers. Another reason to look at PhoneGap is because its architecture utilizes the hybrid approach, and PhoneGap is the only one in this report that uses this approach. The hybrid approach means that the code is executed like a native application but with web technologies [3].

Tests will be made on this cross-platform development environment to get a closer look at what JavaScript might bring to the table. According to the background a cross-compiler is meant to be quicker, therefore it will be useful to find out how PhoneGap compares to the cross-compiled platforms mentioned below when it comes to performance.

4.1.1 PhoneGap architecture

PhoneGap takes advantage of the smartphone’s already built-in web browser engine. It makes a web browser window that covers the entire screen of the smartphone, and all of this is through the same web view that the operating systems use. Because the different operating systems have different web browsers and these use different view-rendering engines, it is important for the developer to account for this in the UI implementation [18].

The architecture used with PhoneGap is somewhat different from one application to another, however most of them use the same basic architecture. The way it works is that the client will communicate with a server and through that receive data. The client is the application that the user interacts with. The architecture used on the client is one where a HTML page is used to store all the logic used in the application, and this is never unloaded from the memory. Instead of communicating directly with a database, a PhoneGap application will route its communication with databases through a server. The techniques used is the same one would normally use in a desktop-browser [18].

As PhoneGap uses a hybrid approach for their cross-platforming, there are some advantages and disadvantages. The main advantage for PhoneGap is that the user interface can be reused across all the platforms. A few other advantages are that the application can use the hardware of the device, such as the camera, as well as being able to use the full computing capabilities of the device. The main disadvantage is that the performance will be worse than a native application because the execution will be done in the browser window. Another disadvantage is that it will suffer from cross-space communication vulnerabilities and threading model incompatibilities because of JavaScript [3].

4.1.2 PhoneGap API documentation

The PhoneGap documentation found on their own webpage is rather extensive. It not only shows how many of the different functions work and how they are meant to be used, but also has little guides for new users. The API shows a number of functions that lets the programmer use PhoneGap to access functionalities that would normally usage of the native language for the smartphone and these have examples to make it easier to figure out what to do. For each function shown in the API there is also a list of the platforms where the functions can be used [26].

4.1.3 PhoneGap test cases

With a simple application that had very few features to it, just a small picture shown on the phone drawn with techniques from the PhoneGap API, it was possible to test how much PhoneGap drains from the battery by default. From a series of tests of the almost empty application running for 30 minutes the average battery drain was approximately 3% of the full battery of the testing smartphone.

A simple test was created to test the performance of the frameworks. The test application contains a picture and this picture gets drawn and removed repeatedly ten million times. A series of runs of this application resulted in an average of 76.25 seconds for the test run.

With PhoneGap it is also possible to create a browser inside an application allowing the developer to extract and insert information to and from webpages [39].

4.2 Unity3D

A way for a developer to reach not only all major computer platforms but also iPhone, Android and Windows Phone with their games and other interactive media is to use Unity3D. It is a development environment that uses the languages C#, Unity Script (similar to JavaScript) and Boo [23]. All testing done with Unity3D will be made with the language C# based on the requirements for this analysis.

4.2.1 Unity3D architecture

Unity3D is a cross-compiler. The main advantages with being a cross-compiler are performance and ability to access device hardware and software. The main disadvantage for a cross-compiler is that hardware specific code, such as accessing a phone’s camera, cannot be reused. The compilation process is different depending on what kind of platform it is compiling for. On the major devices, such as PC, Mac and Android, the code will be compiled into more universally understood byte code. Then it will be using “just-in-time compilation” at runtime to execute the code. If Unity3D is used to compile for another kind of platform, it will compile the code into machine code when the application is built [23].

4.2.2 Unity3D API documentation

The Unity3D documentation appears to cover all that one needs to know. It has a large list of different functions and classes that the user can click on to find an explanation of what each part of these functions can do. There are not many examples to show what each function does, but a few can be found on some functions. Mostly the text seems to explain all the variables in the different classes available, so it does not seem to be too complicated to figure out how they are meant to be used [25].

4.2.3 Unity3D test cases

With a simple application it was possible to test and see how much Unity3D drains by default, with as little done in the application as possible to avoid complex algorithms increasing the battery drain. The application had a simple picture drawn on the screen with nothing else happening. A series of tests with the
application running for 30 minutes resulted in an average battery drain of 5.3%.

Another application was created to be a performance test. This test involved a picture being drawn and removed repeatedly ten million times to see how well each platform performs. This test, repeated multiple times, resulted in an average time of 10.62 seconds to complete the entire loop.

Using Unity3D will not be a problem if a developer needs to access a website for information or adding information to websites, as there is information and code examples found in their documentation explaining how it is done [37].

### 4.3 GameMaker

GameMaker is a development environment chiefly made for making games and it can be used to develop programs that will reach all the major platforms, such as iOS, Windows, Android and Ubuntu with help from their cross-compiler [27]. Tests will be made on the GameMaker framework with the language GML (Game Maker Language) to see if learning this new language would prove useful for normal application development.

#### 4.3.1 GameMaker architecture

GameMaker uses a cross-compiler, which means it will have the advantages and disadvantages mentioned before. More specifically, it compiles the code in two steps. First the code will be turned into C++ code, then it will go through Clang and with this get compiled to a language that the native platforms understand [28]. Clang is a frontend for the LLVM compiler, and it can handle the languages C, C++, Objective C and Objective C++, so the code compiled into C++ code can now be understood by Clang [29]. LLVM is, to quote their webpage, “a collection of modular and reusable compiler and tool chain technologies” [30], so GameMaker uses this, through Clang, to get a compilation to the languages each platform understands. This compilation technique will let Clang do some global final optimizations towards the end, but also let the compiler do a lot of first level optimization beforehand with the C++ compiler [28].

#### 4.3.2 GameMaker API documentation

Since GameMaker uses their own language, GML, their API documentation is from a very basic level where very basic functionalities are explained up to a more advanced level, such as explaining how to use physics and other features in their engine. The explanation for how these functionalities are done seems to be well documented. There are explanations as well as code examples to make a new user of GameMaker have an easy time understanding how things are done. The main issue might be the sheer size of the documentation which can make it hard to find a specific solution to a problem [31].

#### 4.3.3 GameMaker test cases

With a simple application it was possible to test and see how much battery GameMaker framework drains. The application only had one simple picture on it to see how much the framework drained by default, without any complex algorithms. In a series of tests done with the application running for 30 minutes each time, the average battery drain was 10% of the smartphones full battery.

To measure the performance of the framework, another application was created. This application contained a simple picture that was drawn and removed repeatedly ten million times, as well as a timer. Repeated runs of this application gave the average of 16.44 seconds for each time through the loop.

GameMaker is also a framework with no issues when it comes to accessing a website for information. The example code found on their webpage explains exactly how it is done [38].

### 4.4 Qt

Qt is a cross-platform framework that allows a developer to use the languages C++ and QML. QML is a language similar to JavaScript and CSS [34]. As the framework is not made specifically for making games it could prove to have very different results compared to the other development environments in this report that use something other than web-based code. Qt supports Windows, Mac OS X and Linux for desktops and for mobile phones, Android, iOS and Windows Phone are supported in Qt version 5.3 [35 and 36].

#### 4.4.1 Qt architecture

Qt compiles an understandable code for the different platforms it supports, making it a cross-compiler [33]. This means it will have all the benefits mentioned earlier in this article related to cross-compilers. However, compared to the more game-focused frameworks, Qt should perform rather differently. Since Qt is event-based and not made for making games, it will not have the same “game loop” that GameMaker and Unity3D has. This might lead to interesting differences between the frameworks.

#### 4.4.2 Qt API documentation

As seen on their own webpages documentation section [32], the list of functionalities that Qt brings to the table is rather large. A clever choice for their names makes it a simple task to see in one’s code what function is related to Qt, as all of them begin with the letter ‘Q’. Going deeper into the functions, each of them appears to have a detailed description explaining what a certain object does as well as some shorter explanation of what each variable does and can handle. All in all it seems to explain everything well enough to be understood. Included in Qt modules are also many examples made in Qt that are easy to understand.

#### 4.4.3 Qt test cases

To test the framework’s battery consumption, a simple picture was drawn on the screen with no extra algorithms or functionalities. This test was then executed for 30 minutes and an average of 2.5% battery drain was established.

Another test evaluated the performance of the framework. This test drew a simple picture on the screen and then removed and redrew it ten million times with functions built into the framework. When this test was run repeatedly, it came out with an average result of 28.79 seconds for each test run.

Accessing webpages to alter information or extract information is easy with the help from Qt JavaScript part, and Qt has a detailed description of how their functions work when it comes to accessing webpages [40].

### 5. Results

In this section the results of the tests will be presented. While the architecture section explains the differences found between the
framework’s different architectures, the API section will be moved on to the discussion as it will be more subjective.

5.1 Architecture comparison
Qt, GameMaker and Unity3D have a quite similar architecture, while PhoneGap differs as can be seen in Table 1. This means that, according to the background research, Qt, GameMaker and Unity3D should have the possibility of being quicker. As mentioned in the GameMaker architecture section, Clang is meant to do some extra optimization with the compilation thus it should increase the performance. The fact that Qt isn’t primarily made for games should also give it a different kind of result. It should be useful to find out how the different cross-compilers compare to each other and to the hybrid approach PhoneGap uses.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Architecture type</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneGap</td>
<td>Hybrid</td>
</tr>
<tr>
<td>Unity3D</td>
<td>Cross-compiler</td>
</tr>
<tr>
<td>GameMaker</td>
<td>Cross-compiler</td>
</tr>
<tr>
<td>Qt</td>
<td>Cross-compiler</td>
</tr>
</tbody>
</table>

5.2 Power consumption
With a simple program made on each platform that only showed a picture it was possible to measure how much power they drain. The results are as shown in Table 2. As both Unity3D and GameMaker are mainly for making games, it is not very surprising that they drain so much power compared to the rest. The game loop should make it rather inefficient. The other tested frameworks got almost the same result as the application made in its native language (In this case Java for Android). So if battery efficiency is important in an application, Qt and PhoneGap should be considered rather than Unity3D and GameMaker.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Percentage of full battery drained</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneGap</td>
<td>3%</td>
</tr>
<tr>
<td>Unity3D</td>
<td>5.3%</td>
</tr>
<tr>
<td>GameMaker</td>
<td>10%</td>
</tr>
<tr>
<td>Qt</td>
<td>2.5%</td>
</tr>
<tr>
<td>Native</td>
<td>3%</td>
</tr>
</tbody>
</table>

5.3 Performance
A simple application made for each framework, all having the same picture being shown and then removed repeatedly ten million times, lead to the results shown in Table 3. For each framework the test was made with equal conditions, no changes were made to the picture drawn and functionalities from the frameworks were used to draw and remove the picture in the loop. So if an application will have many pictures moving around, it is obvious that the frameworks aimed towards games have the advantage.

<table>
<thead>
<tr>
<th>Framework</th>
<th>Time for 10 million loops in seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneGap</td>
<td>76.25s</td>
</tr>
<tr>
<td>Unity3D</td>
<td>10.62s</td>
</tr>
<tr>
<td>GameMaker</td>
<td>16.44s</td>
</tr>
<tr>
<td>Qt</td>
<td>28.79s</td>
</tr>
<tr>
<td>Native</td>
<td>66.33s</td>
</tr>
</tbody>
</table>

5.4 Internet Accessibility
Each of the cross-platform development environments that have been looked into have some kind of function that will let them access and add information from webpages. The information required for this section was all found on their own webpages’ documentation section.

6. Discussion
This section will be more subjective and include the writers’ own thoughts on some of the subjects. The section about API will be purely subjective as it will simply include what the writers think of the documentation of the different frameworks and how well they helped the writers with the tests.

6.1 API documentation
With a quick look at the API section for each framework it would seem like they were all good however this was not always the case. To help the writers with making the tests for the frameworks, the API documentation was used. Through this use of the documentation it was possible to get a better understanding of each framework and documentation. PhoneGap’s API documentation turned out to be less good than what it first seemed to be, as it was quite difficult to navigate through the structure to find the right functions. Unity3D was rather easy to navigate through, and if anything lacked in their explanations of the functions it was a simple task to find more information about it on their forums. Qt had a documentation so well structured that it did not require any time spent on the site to find the correct functions and to figure out how they were meant to be used. Qt also came with a lot of examples and tutorials which made it very easy to find code examples of what was needed to test i.e. HTML parsing. GameMaker had a lot of tutorials to explain how things are made, making it a simple task to make the required tests, however the API documentation on their webpage was somewhat messy and lacking.

6.2 Power consumption
As expected from the background research where PhoneGap did well compared to Titanium, PhoneGap did well here too. It could however come as a surprise that Qt outperforms not only PhoneGap for power consumption but also the natively programmed version. Unity3D and GameMaker both fell far
behind in the subject of power consumption, but it should not come as a surprise as they are both specifically made for games. It is however good to see there is still an option with C/C++ for cross-platforming in Qt.

6.3 Performance

As claimed by R. Raj C.P and S. B. Tolety [3], that a cross-compiled application should have a higher performance while a Hybrid should not, it appears to be correct. Both Unity3D and GameMaker are far ahead of PhoneGap, the Hybrid, when it came to this performance test. This could however be related to Unity3D and GameMaker mainly being made for games, and the performance test involved something graphical. The perhaps more interesting comparison would be between Qt and PhoneGap and it would appear the cross-compiled Qt is quicker here too, more than twice as fast as both the native and PhoneGap version. As mentioned in Cross-Platform Developer Tools 2012, PhoneGap really does lack in performance [8]. The article also mentions 27% of the current PhoneGap users are leaving PhoneGap for something else, with performance being one of the main reasons. This test truly shows how far behind PhoneGap is in this regard.

6.4 Internet Accessibility

With the information found from their own documentation, some were easier to understand than the others. Unity3D has great examples in their documentation on how to get information from a webpage, while Qt lacked when it comes to examples but had a rather decent amount of explanatory text and its application can partially be built with JavaScript. GameMaker is a bit more difficult to understand than the previous two. For PhoneGap it is quite easy to access webpages and perform html parsing due to its web programming nature and it utilizes JavaScript as its main language although it is quite hard to find information and help.

7. Conclusion

For the frameworks GameMaker, PhoneGap, Qt and Unity3D we compared power consumption, API documentation and architecture. In short it would appear that Qt is far superior when it comes to power consumption, draining less than half of GameMaker and Unity3D and a little bit less than PhoneGap. Unity3D and Qt appear to be the easiest frameworks to get started with, as Qt has a very well written documentation site and Unity3Ds forum provided a lot of useful information. Unity3D once more produced a better result than GameMaker, making it a great choice unless power consumption is an issue. With Qt running the test at less than the time of Native and PhoneGap, it is the obvious choice whenever power consumption is a consideration. On the topic of network accessibility, all of the frameworks had at least one working function for accessing information from webpages. The only development environment standing out among the others would be PhoneGap, as it needed to create a browser window to gather information. So it would appear that Qt is ahead of PhoneGap at everything tested, while Unity3D appears to be the choice for an application where performance is the only focus.

To summarize the entire work, you can see the combined results in figure 1. It has the information that a developer might find useful if they chose to take on a new cross-platform development environment.

![Figure 1: Visualization of the results](image)

For the future it could be useful to build an application for Android and port it to iPhone to see how much of the code needs to be changed for each platform. Another topic to analyze is to see how well compilation on the cross-platform development environments compares to natively programmed versions on different platforms. This could be useful to see if they are equally optimized for different platforms, i.e. Qt for Android vs native Android and Qt for iPhone vs native iPhone.

8. REFERENCES


