UNIVERSITY OF GOTHENBURG

Comparison of Native, Cross-Platform and Hyper Mobile Development Tools Approaches for iOS and Android Mobile Applications

Bachelor of Science Thesis in the Software Engineering and Management

SHAN JIANG

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Comparison of Native, Cross-Platform and Hyper Mobile Development Tools Approaches for iOS and Android Mobile Applications

Shan Jiang

Examiner: Boban Vesin

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2016

Comparison of Native, Cross-Platform and Hyper Mobile Development Tools Approaches for iOS and Android Mobile Applications

Shan Jiang

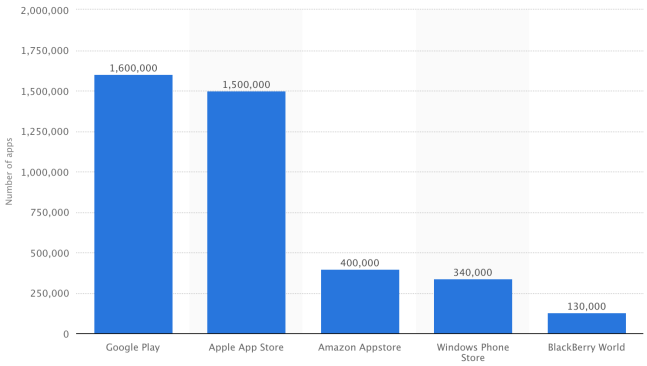
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gusjiash@gmail.com

Abstract: To take a position in the competitive mobile application market, most of the Information Technology (IT) companies have to develop the same application for various mobile operating systems. To solve this issue, Cybercom provided a thesis topic and a project which decreased development time by sharing the same core functions between Android and iOS applications. To share more code between platforms and take advantage of cross-platform tools and hybrid tools, the author has conducted a case study which includes experimenting on Xcode, Android Studio, Xamarin and Cordova to produce certain type of mobile applications and collect data during implementation processes and from generated applications. This paper not only compares the performance of individual process and application but analyses the data from various mobile development tools. As a result, it provides reference of choosing the optimal mobile application development tool for future researchers and developers.

1. Introduction

As the functionality and popularity of electronics grow, the demands of related applications are also increasing. Nowadays, these applications are directly influencing our daily lives from different perspectives. Android and IOS, the leaders of smart phone markets, are providing millions of applications till the middle of 2015 (Figure 1) . As J.Perchat [14] mentioned, to achieve the demands of these growing markets, developers need to master various programming languages and own more than one devices in order to create different applications. The ultimate goal is to lighten the load for developers and provide suggestions for choosing the most suitable tool for different circumstances.

Figure 1: Number of Apps Available in July 2015[1]



Cybercom has realized and experienced this conflict during several projects, therefore they decided to solve this issue by providing a thesis topic to the author. The author, who is a third year student of Software Engineer and Management bachelor program and thesis worker in Cybercom, conducts a case study on comparison of three categories development tools, which are Single Platform Development Tool (**SPDT**) , Cross-Platform Development Tool (**CPDT**) and Hybrid Mobile Application Development Tool (**HMDT**) from development process and quality of the result point of views. This case study contains four implementation processes and all the experiments are conducted by author, the only developer for this study. Due to the unique relationship between Cybercom and author, no other developers can join this study.

With the combination of industry practice and literature review, this work makes several contributions. First of all, it provides readers a comprehensive overview of these three

categories of IDE, for example the licence, API, support mobile platform, availability, etc. Secondly, the author conducted four experiments on the outstanding tools from different categories. The process, number of blockers and so on will be recorded for evaluation and reference for other researchers. Thirdly, this paper presents a detailed analysis and evaluation of the products that are delivered from various tools. These approaches can provide advices for the industry partner Cybercom Group as well as researchers in the same domain in future tool selection.

There are three major stages in this research: preparation of implementation and case study from Section 4, conduction of case study on four develop tools from Section 5 and solve research question from Section 6. Preparation section states which tools are used in this study, the reason for choosing them, and functional requirements for final products. In the second stage, data is collected from two aspects: implementation process and applications produced from them. In order to collect both quantitative and qualitative data objectively, certain tools and methods are selected to assisted this step. After gathered enough data, the last stage started. To solve research question, author has separated data into two concerns: Process and Products. This stage states analysis results from different aspects. Beside these, Section 7 illustrates threats to validity. Finally discussion and conclusion is presenting in section 8.

2. Background

This section starts with describing the three categories of platforms more in depth and providing a general mobile development knowledge to the reader. Then it comes up with description of previous work and how this paper differs from them.

2.1 Three categories of development platforms

As mentioned before, there are three major categories of mobile development tools. Xcode and Android Studio are the typical representatives of the first kind – SPDT, which can only be used to develop single platform applications for native environment. This single platform development tools all have fully integrated functionalities such as user interface builder, debugger, support external libraries, accessibility to hardware. Besides all of these benefits, the limitations include that they can only build application for single environment with different programming language and some of them have restricted installation environment, for instance Xcode can only used in Mac OS X with its unique development tool Xcode to develop iOS application, and developer can only make use for Android application development with Android Studio.

On the other hand, development tools from the CPDT category usually support developers to build mobile applications for more than two platforms and support installation on various environment. One of the features from them is that only one programming language is required, and the tool will help the programmer to synchronize and generate platform specific code automatically in order to create more than one mobile applications suitable for various platforms.

In HMDT category, for example Cordova and Titanium, also can generate mobile application suitable for different environment due to its specificity of a web based application. These applications are built with HTML and JavaScript as a website but can access more hardware features than a web browser.

This paper focuses on providing more information of developing both iOS and Android applications to junior developers. Therefore, the author as the only developer has conducted four developments by using development tools selected from these three categories to create three pairs of iOS and Android applications.

2.2 Related Work

Most of the papers in this field have separate concerns. Boushehrinejadmoradi [4], Heitkötter [8], and Palmieri [13] compared different development frameworks by building a prototype testing tool to examine the behaviour consistency between them. Appiah [3], Thakare [17], Heitkötter et al.[8], Dalmasso et al.[5] on the other hand, established a few sets of criteria to estimate different development tools. Xanthopoulos et al. [18] focused on overall acknowledgement of single category of CPDT with case study of development.

The majority of existing literature focused on one or two categories of mobile development tools with emphasis on either criteria analysis or prototype testing such as Angulo and Ferre [2], Paulo R. M. [6] and J.Masner [10]. This research, however, is going to focus on a few outstanding tools from all three categories of mobile development tools with a pragmatic comparison. Also an evaluation of their performances will be conducted by building a similar iOS and Android application with these candidates.

3. Research Question

After observed an uneven usage of SPDT, CPDT and HMDT in the mobile development fields, a survey has been conducted in middle March, 2016 to discover "IOS and Android Mobile Application Development Tool Decision [9] ". The result shows that 15% of junior developers from the third year of Gothenburg University Software Engineering and Management Program were willing to use CPDT and HMDT for their future project. This number grew to 68%

[9] after these junior developers read the basic information of CPDT and HMDT. This phenomenon raise up few questions. Why junior developers are afraid to choose CPDT and HMDT? Can CPDT and HMDT help junior developers to increase the overall performance? Does the process of developing both Android and IOS applications differ from SPDT to CPDT and HMDT? What are the deployment result among these three categories?

This research is going to provide the readers a better understanding of the overall development process from the three categories of mobile development tools when creating both Android and IOS application. A deeper comprehending of the relationship between development tools and the quality of their products can also be achieved. Therefore the research question of this study is:

Is there any difference in performance between SPDT, CPDT and HMDT when junior programmer building mobile applications for both Android and iOS platform?

On behalf of the overall qualitative and quantitative data from development processes and final products, their performance is represented by the value of each project. The purpose of measure performance in this study is to evaluate the combination of inputs and outcomes. Many index such as development time, lines of code, memory usage, are selected to represent performance in order for the author to analyse the variation among different tools. Explanation of these index and how to analyse them are stated in research method section.

The answer of this research question can help the industry partner Cybercome to select the right tool in the future mobile application projects. A correct decision can bring out maximum output with the best quality also it can inspire employees for future work.

Sub Research Questions:

1. What are the performance differences regarding the implementation processes?

2. What are the performance differences regarding the final products?

4 Preparation

This section explain the plan for the whole study, which includes how to choose development tools and decide functional requirements for the applications.

4.1 Selection of tools

As mentioned before, Xcode and Android Studio are the only two officially authorised development tools available, which have indisputable positions in SPDT area. Therefore they are the representations for SPDT category.

Combined the knowledge of previous studies, official documentations and functional requirements of target mobile applications, these assessment criteria are set to help the author to make the right selection of CPDT and HMDT.

- **Licence**: It can have multiple licence types, but on the purpose of research study, there must be at least one type for free.

- **API**: It should be easy to learn and use for junior developers.

- **Support Mobile Platform**: It must support at least Android and iOS.

- **Availability**: It is able to run in Mac OS X, since the case study is only conducting on this environment.

- **Programming Language**: To achieve an impartial result, case study object should have knowledge of related programming languages

- **Feasibility**: Code generated by these tools should be able to run in related SPDT.

- **Expandability**: It should support the Google Authentication API, encrypted storage and secure channel communication with server by add external modules/libraries.

- **UI experience**: Unify code should be able to generate native alike user interface.

- **Simulator**: Must support native simulator for the purpose of development and testing.

At the beginning of this study, more than ten options were found from CPDT and HMDT categories. Most of them are eliminated due to the fact that they can not fulfil the criteria of this study. However Xamarian and Cordova not only satisfied all the requirements but also provided well-defined documentations and examples. Hence these two tools joined the evaluation of case study.

As result, Xcode, Android Studio, Xamarian, and Cordova are chosen for this study. Detailed information of these tools are presenting in table 1 from all the selection criteria.

4.2 Functional Requirements

Functional requirements play a major role in this study. They determine the plan for the whole project, selection of development tools, and what kind of data that can be collected later. All the mobile applications of this study should apply these functional requirements:

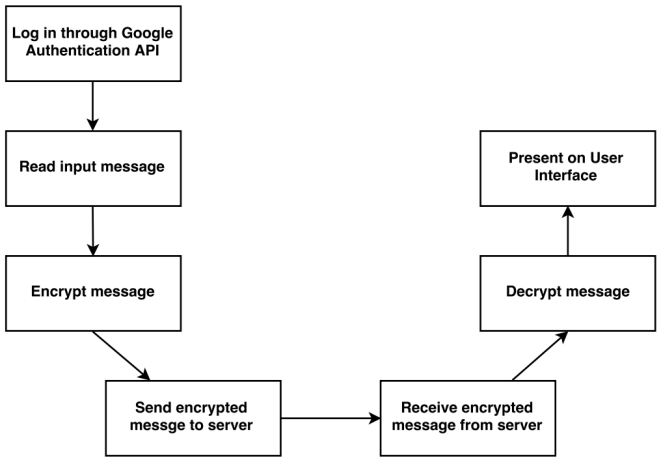
Table 1: Comparison of the representatives from CPDT, HMDT and SPDT.

	CPDT Xamarin	**HMDT** Cordova	**SPDT** Xcode	**SPDT** Android Studio
Licence	Community:Free Professional:$45/month$ $Enterprise$:250/month Integrated with Xamarin MIT licence (opem sauce)	Free, Open source	Free for MAC user	Free, Developed based on IntelliJ (open source)
API	Xamarin.Forms Api Xamarin.iOS Xamarin.Android	Battery, Camera Console, Contact .... Network information, File Transfer, etc.	Native support	Native support
Support Mobile Platform	Android IOS Windows	Android, iOS, Windows, blackberry	IOS	Android
Availability	Mac OS X ,Windows	Mac OS X Windows	Mac OS X	Windows, Linux, Mac OS X
Programming skills	C# XAML for mark up	HTML5, CSS3, JavaScript	C, C++, Objective-C/C++, Swift, AppleScript, Python, etc.	Java, XML
Feasibility	1. Provide test on real device in cloud 2. Support multiple threads debug at once and inspect state across many threads 3. Hardware debug 4. Monitor application performance, such as the usage of CPU, GPU and memory usage, UI responsiveness and network utilization 5. UI debug for XAML 6. Updating with mobile platform	1. Work with Phone Gap, Ionic ,Monaca , App Builder ... 2. Do not provide native debug options. Have to use Xcode or Android Studio	1. Interface Builder 2. Version Editor to compare difference between versions 3. Test Navigator 4. Customized environments 5. Preconfigured code 6. Customized Schemes (builds and runs) 7. Zombie Detection can trap hard-to.find crashes and errors	1. Translation Editor helps to update string resources 2. Integrated sample project 3. Integrated with Google App Engine 4. Separate test model
Expandability	Support library in: Objective-C, Java, C/C++	Has 1199 plug-ins to support different platform	Native libraries, External library with objective-c, Swift and C++	Choose from "Android Support Library"
UI experience	1. Support native UI. 2. Xamarin.Form can transfer the base code into system specific UI. 3. And user can add platform native code for different UI.	1. With the support of all kinds of additional framework: Jquery Mobile, ionic, Ratchet... 2. Different UI will be created for various target platform	1. Build-in Interface Builder help user to create UI by drag-and-drop. 2. Assistant provide link between interface component with source code. 3. Auto Layout.	1. Layout Editor enable user to drag and drop element to create UI. 2. Theme Editor create UI for future usage. 3. Use existed image and generated customized icon
Distribution	Not included	Not included	Support to App store	Market place and users
Simulator	1. Native support iOS and Android simulator. 2. Remotely support run project on device	1. Preview in browser and native simulators. 2. Support running on devices.	All type of IPhone, IPad simulator. Remotely support all iOS devices.	All type of Android simulator with remotely Android devices
Share code	up to 90%	None	None	

1. It must allow the user to login with a Gmail account.
2. It must be able to encrypt and decrypt messages locally.
3. It must be able to read and write encrypted messages to the server.

For the purpose of secure communication, development tools must be able to integrate with the Google authentication API. Furthermore, messages should be changed to encrypt content before they are transferred to the server, a local decryption is ran once the user received it. The step by step work flow of these applications are presented in Figure 2 .

Figure 2: Work flow of final product



5. Research Methodology

A case study usually focus on the contemporary phenomena and generate deeper understanding in its context [16] . Deploying this methodology is beneficial to collect plenty of detailed information during deployment process and to established criteria to compare the results generated from various procedures. In this paper, author has conducted four experiments on selected development tools. The result of these experiments is used to evaluate the performance among three categories of tools, and to introduce an alternative way for develop both iOS and android application together to junior developers. The aim of this case study is to decrease redundant development time and increase productivity for junior developers.

5.1 Data Collection

To comprehensively and objectively gather data, data collection is emphasize on two sides: implementation processes and finalized mobile applications.

Table 2: Observed Parameters

Development Process	• Implementation hours • Installation time • Easy to Use (Duration time from start project to finish the first task) • Productivity • Integrated Simulator • Powder consumption of development • Memory Usage of development • User interface builder
Product	• Execution time for every functions • Security from three aspects: 1. Authentication 2. Encryption & Decryption 3. Communication with Server • Maintainability 1. Size of project 2. Lines of code 3. Complexity of code (Cyclomatic complexity) 4. Coupling 5. Separation of concern: Cohesion (LCOM Lack of Cohesion of Methods)

5.1.1 Implementation process

Author has conducted three implementation processes in this study. The first one was a native implementation, where officially development tool Xcode and Android Studio were used with native programming language Objective-C and Java to create one iOS application and one Android application. The second project was a Cross-platform development. With its distinguishing features, both IOS and Android applications were generate by using C# in Xamarin. Last one was a Hybrid development, which

developed both IOS and Android applications based on web-development technique in Cordova. All the deployments were running on the same environment by the same programmer to avoid unnecessary influence.

To recording more accurate and consistent data, all the process were sharing a uniform project plan. Here are the data information related every aspect:

- **Usability**: Is it easy to use? The term "easy to use" summarize the effort for setting up a fully usable development environment for a framework and desired platform [8]. How long time will a junior developer spend before families with it?
- **Accuracy**: How many tasks need to extend the time schedule? How many tasks have been finished before/within the schedule? How many hours does the project needs?
- **Expandability**: Which plug-in/module have been involved for this study? How many plug-in/module have been tried for this project?
- **Integration**: Are the provided functionalities completed?
- **Power Assumption**: How much battery does the development consumed per hour?
- **Memory Usage**: How much memory does the application consumed per hour?

5.1.2 Finalized applications

After the development stage finished, every tool has generated two mobile applications as showing in table 3. More data were collected from these applications in these respects:

Table 3: Relationship of development tools and applications

Category	Tool	Application
SPDT	Xcode	iOS application
SPDT	Android	Android application
CPDT	Xamarin	iOS application
CPDT	Xamarin	Android application
HMDT	Cordova	iOS application
HMDT	Cordova	Android application

- **Performance**: How long time did the applications need to be fully function-able since simulator started? How long time did the applications needed to execute every function, which included send messages to server, read messages from server, login, encryption and decryption?

  1. **Start up time** were measured by calculating the time difference on simulators log.

  2. **Execution time for each function** can be measured by using the build-in "Print" function. This function can print out current time in console with accuracy to millisecond. Therefore execution time for the target function can be calculated by the time difference of function start up time and function finish time.

- **Security**: Authentication, secure storage, communication with server are the major object for this aspect. Data were collected from their performance and development times.

- **Maintainability**: Both quantitative and qualitative data were collected from this field. Lines of code, complexity and coupling represent the quantitative side, and cohesion represent qualitative side. These data were measured by following metrics:

  1. **Size** as one of the most important attribute of software project [11], it not only indicates the consumption of cost and time but also represents the quality and maintainability of the project. To measure the size of each application in this study, Source Lines of Code (SLOC or LOC) will be calculated by Count Lines of Code (CLOC) .

  2. **Complexity** as another essential index for comparing efficiency beside LOC, it represents the degree of maintainability for a project. A clean written code, on the other side can avoid redundant functions. Cyclomatic complexity known as "a measure of the maximum number of linearly independent circuits in a program control graph" [7] is widely used in industry and academia. Cyclomatic complexity number (CCN) as the result of this metric has a stander range from 1 to 10 as presenting desirable complexity, which will be used as an important index in Result section. The higher the CCN is, the more complexity the source code is.

Figure 3: Complexity Metric

```
Cyclomatic complexity = E - N + P
where,
  E = number of edges in the flow graph.
  N = number of nodes in the flow graph.
  P = number of nodes that have exit points
```

3. **Coupling** as "one key to several quality factors of software is the way components are connected" [12] , it can be used to measure numbers of quality factors, such as reliability, maintainability and complexity. Tightly coupled classes are highly depend on each others by sharing types, variables and states. On the opposite side, loosely coupled classes are much more independent. A high coupling value usually refers to lower readability, reuseability and maintainability. *Software engineering: a practitioner's approach* [15] introduced a simple yet efficient way to calculate coupling as figure 4 showing below:

Figure 4: Coupling metric

For data and control flow coupling:

- d_i: number of input data parameters
- c_i: number of input control parameters
- d_o: number of output data parameters
- c_o: number of output control parameters

For global coupling:

- g_d: number of global variables used as data
- g_c: number of global variables used as control

For environmental coupling:

- **w**: number of modules called (fan-out)
- **r**: number of modules calling the module under consideration (fan-in)

$$\text{Coupling}(C) = 1 - \frac{1}{d_i + 2 \times c_i + d_o + 2 \times c_o + g_d + 2 \times g_c + w + r}$$

This metric brings out a result in the range of 0.67 to 1.0. A higher number represent a highly coupled system.

4. The relationship among methods within the same class can be measured by **cohesion** metric: Lack of Cohesion of Methods (LCOM) , which also means the degree of separation of concern. A highly cohesion system usually featured with reliability, understandability and reusability. LCOM4 helps users to indicate how many sets of related methods in a class. When every method in a class has built link with another other, the metric will brings out a value of one, which means a high cohesive class. As the level of cohesion gets lower, the value of LCOM growth higher. Expect the result of zero indicated there are no method existed in the class.

- **Consumption**: How much battery and memory will the application consummated from beginning to finish testing all the functions?

In conclusion, this study has gathered both qualitative and quantitative data from two sides as table 2 presents. Process side includes four executions cycles and product side targeted on the six mobile applications.

5.2 Data Analysis

Comparative analysis was used to evaluated data collected from previous section. This method targeted on a set of characteristics of aforementioned developing processes and finalized applications to discover the gap among SPDT, CPDT and HMDT.

There are two degrees of comparative analysis in this study: horizontal comparison and vertical comparison. Data collected from the same aspect were stored in the same row (see figure 6) in order to compare the value of the target attribute. Results were presented by different background colours: green for the best performance and red for the worst performance.

After gained result from horizontal comparison, vertical analysis started. In this degree, author targeted on the accumulated result of each column (see figure 11). With the evaluation results from developing process till finalized product, ultimate results and conclusions are provided in Section 6.

6 Result

This study was conducted in the month of May 2016 to discover the relationship between development tools and performance of their usage and products. It involved four individual projects as the representatives of native tools, cross-platform tools and hybrid tools. Six individual Messages Transfer Mobile Applications were completed during these projects. With the help of research methodology, this section presents the result of evaluation that adhere to the principle of consistency, objectivity and all-round way.

6.1 Implementation Result

Due to the fact that Cordova and Xamarin both brought out two mobile applications while Xcode and Android stu-

Figure 5: Assessment of Development Process(hour) Part 1

Category		SPDT	CPDT	HMDT
Step	Task Name	Xcode & Android Studio	Xamarin	Cordova
1	Install application	1	1	1
2	Running simple test to check the installation	5	0,5	2
	Configuration of multi-platform development environment(only for CPDT and HMDT tools)	0	4	4
3	Integrate Google Authentication API to project	4	2	4
4	Add Log-in functionality (hour) and related UI component	3	3	2
5	Create private view and related functions	1	1	0,5
6	Add UI component to private view	1	0,5	0,5
7	Read input message and add storage function to save this mesaage	1,5	1,5	0,5
8	Add encrypt message function	3,5	0,5	1
9	Establish Communication and send message to server	9,5	2	5
10	Connect UI component "Send button" with the functions created in step 8 and 9	1,5	0,5	0,5
11	Establish another communication with server to read message and store this message locally	2,5	2	2
12	Add decrypt message function	3	0,5	1
13	Connect UI component "Read button" with the functions created in step 11	1	0,5	0,5
14	Connect UI component "Text View"with result from step 12 to shows decrypted message	1,5	0,5	0,5
Total Hour		39	20	25
Best Result	Worst Result			

dio only produced one. Therefore, to achieve equitable result, data collected from Xcode and Android Studio were joined together to represent native tool and compared with other two categories.

6.1.1 Implementation Hours

To ensure the objectiveness, all four implementations have followed the same project plan and accomplished the function requirements from Section 4. The number of hours spend on each task corresponding to the development tool is presenting in the figure 5.

From the table we could observe that the total implementation hours shows significant difference. To produced both iOS and Android applications, Xamarin provide the best efficiency which is 19 hours less then SPDT and Cordova achieved 15 hours faster then HMDT.

6.1.2 Installation

Following the development process, the first task is installation which presents in figure 6. Android studio and Xcode both requires half hour to download and install in the OS X EI Capitan system and other two applications require the same amount time as the sum of two native tools.

6.1.3 Easy To Use

Beside basic task (ID 2 in figure 5) , the hours spend on successfully apply the log-in function (ID 3 and ID 4) in figure 5 are also included in this category. As showing in second row of figure 6. Xamarin shows the best performance of 9,5 hours, which is half hour faster then Cordova and 2,5 hours faster then native tools.

Figure 6: Assessment of Development Process Part 2

	SPDT	CPDT	HMDT
	Android Studio & Xcode	Xamarin	Cordova
	Android IOS	Android&iOS	Android&iOS
Installation (hour)	1	1	1h
Easy to use (hour)	12	9.5	10
Implementation hours	39	20	25
Integrated Simulator	Provide native simulator	Support native simulator	Support native simulator
Powder consumption of development	99.43	48	34,9
Memory Usage of development(mb)	1818	901	840
User interface builder	Support	Support	Not Support
Best Result	Worst Result		

Figure 7: Assessment of Produced Application

	Android			IOS		
Category	SPDT	CPDT	HMDT	SPDT	CPDT	HMDT
Development Tool	Android Studio	Cordova	Xamarin	Xcode	Cordova	Xamarin
Start up time (min)	0,64	0,59	0,955	0,04	0,15	0,19
Log-in function execution time(s)	0,34	0.78	0,85	0,0031	0,0049	0,0036
Encrypt function execution time(s)	0,001	0,001	0,001	0,001	0,001	0,001
Send function execution(s)	0,088	0,084	0,084	0,0757	0,0847	0,076
Decrypt function execution time(s)	0,076	0,08	0,063	0,001	0,0005	0,0005
Receive function execution time(s)	0,10	0,072	0,078	0,0667	0,0653	0,066
Size of Project	64,6MB	104,1MB	147,9MB	33,2MB	104,1MB	147,9MB
Lines of code	185 + interface builder	388	473 + Interface Builder	260 + interface builder	388	473 + Interface Builder
Complexity	3	5	4	3	5	4
Coupling	0,8	0,94	0,87	0,9	0,94	0,87
Cohesion	1	4	4	2	4	4
Best Result	Worst Result					

6.1.4 Productivity

Each category of development tool has brought out an even number of products with same functionalities. Therefore the productivity can be measured from the overall time spend on each process. As figure 6 shows, native tools spent almost double amount of time then cross-platform tool to achieve the same result. Hence Xamarin shows the best productivity in this case study.

6.1.5 Integrated Simulator

Both figure 6 and table 1 show that every development tool from three categories supports simulator, the only difference is that both cross-platform tool and hybrid tool are depending on the simulator provided by sing-platform tools. From the integration point of view, Xcode and Android Studio have a better control on the simulator then others.

6.1.6 Memory and Powder Consumption

Memory usage and powder consumption are the two index which can directly influence the usability of the tools. The evaluation results of powder are almost the same for every project. However when combined the value of Xcode and Android Studio, it is not surprised that SPDT consumed almost doubled amount of memory and battery then CPDT and HMDT candidates.

6.1.7 User Interface Builder

Because the nature of hybrid tools are web-based development, it uses html and css files to construct the user interface. Therefore similar to native and corss-platform tools, Cordova does not provide a user friendly interface builder. However, as shows in figure fig:Schedule, Cordova project did not spend extra time on interface related task then other projects which have user interface builder.

6.2 Produced Applications

To answer the research question about the performance of final applications. This section shows the the analyse result of six products from different angles.

6.2.1 Execution Time

In general, speed means the running time of application in devices or simulator. This section gives speed a more intensive evaluation from six aspects: launch application, successfully authorize, encryption, send messages to server, decryption, and receive messages from server. They are presenting in figure 7.

For android applications, Cordova shows the best results in starting time, send messages and receive messages, which are 38%, 4,5% and 28% respectively higher then the worst value. Applications from Android Studio and Xamarin both have two worst results in different categories. However Xamarin bring out two top score in send messages and decryption, while Android studio only gain one top score in the third category. Accordingly, the android application from Cordova project has the best performance and all three applications show diversely result in every category.

For iOS supplications, it is more difficult to evaluate which application performs the best. Disregard the encryption time, where all the applications have the similar results, the applications generated from Xcode and Cordova have covered all the best and worst records. From figure 7 we can see that single platform application are 73%, 24% and 11% faster then CPDT application in launch, login and send messages categories. On the other side, Cordova has the best performance in decryption and receiving messages where it spent 50% and 2.1% less time then Xcode. HMDT application, on the other hand keeps average speed for most field. Thus the application from Xcode project perform the best in the speed aspect.

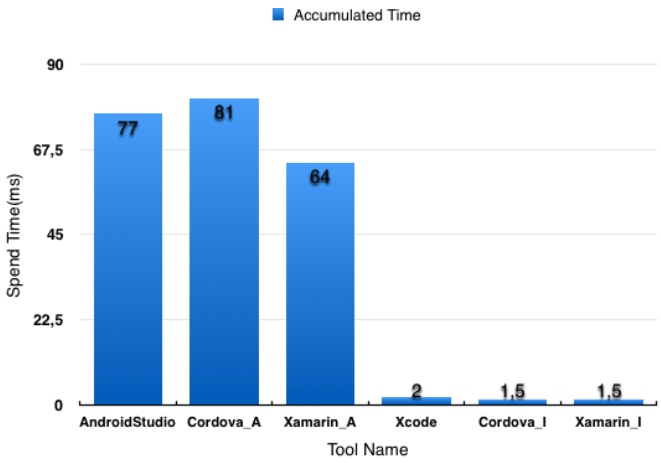
6.2.2 Security

That the major functionality of these applications is to transfer messages between local devices and on-line server. Therefore to compare the performance among three categories of development tools, the result of functionalities on behalf of security level must be taken more concern. The results of security can be analysed into three parts: login, encryption storage and secure channel communication.

With the assist of Google authentication API, every mobile application from this study can login with a Gmail account through the same channel support by Google. Thus, all the result remains at the same level of functionality for this part. However applications produced from SPDT shows the the fastest execution time.

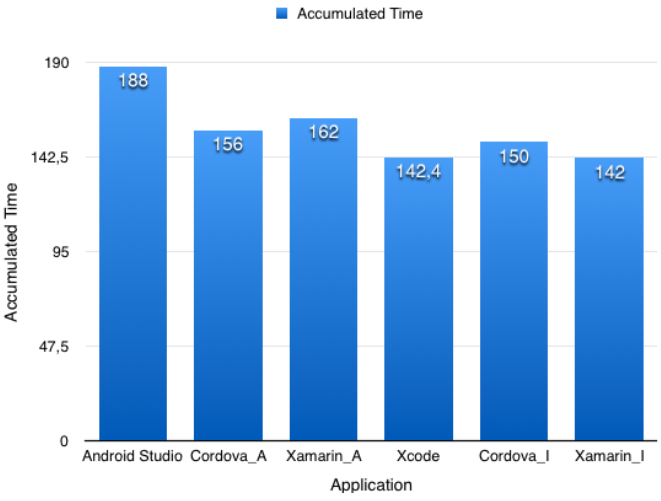
Base on source code and content from figure 6, applications have conducted different ways of encryption and decryption, For example, native android application used Crypto, native iOS application applyed AES256, Cordova project used CryptoJS and CryptoStream is used for Xamarin project. These four methods all belongs to symmetric key algorithms and followed the same principles. A pre-setted key is required, and it is a necessary component for both encryption and decryption to finish the process. Analyse

Figure 8: Encryption and Decryption Functions Speed



the principles and process behind this methods are beyond the scope of this study. A more objective way is compare the speed of these functions to evaluate their performance. Figure 8 presenting the accumulated time spend on encryption and decryption for each application. As we can see, there is a significant difference between Android and iOS applications. Nevertheless, this phenomenon can be ignore due to the fact that the values present by second and fifth bar are sharing the same source files, same as third bar with sixth bar from Xamarin. The fastest android application comes from Xamarin, which is 9% higher then average speed. All the iOS applications shows an even performance, the one comes from Xcode project is slightly slower then other two. Further more, this figure presents that there is a big difference among the applications generated from various tools. HMDT spent 0,066 minute to finish encryption and decryption, while SPDT took 0,079 minute and CPDT 0,083 minute.

Figure 9: Connection Speed

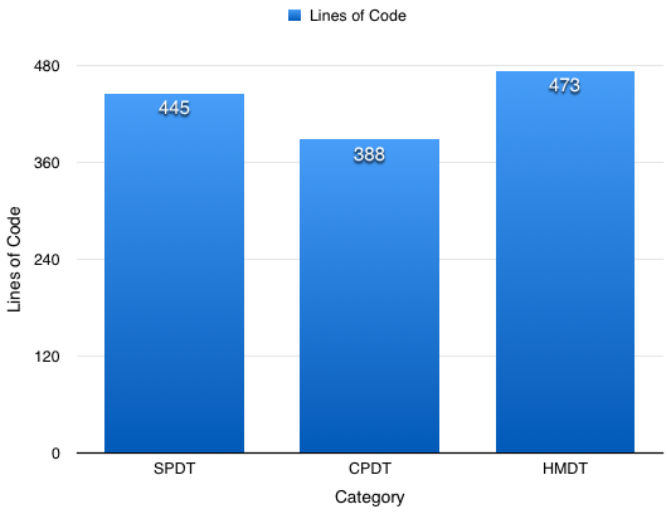


The last part of security is the communication between local device and server. As figure 9 shows iOS applications have an average faster speed then Android applications. Android application from Cordova project and iOS application from Xcode took the shortest time for both way connection. And the accumulated results for SPDT, CPDT and HMDT are: 330,4ms, 306ms, 304ms. In conclusion, finalized results from CPDT and HMDT are better then SPDT.

6.2.3 Maintainability

As explained in Section 5, the maintainability of mobile application come from four parts: size, complexity, coupling and cohesion. It is another index to indicate the performance of development tools. These results with corresponding targets are presenting in figure 10. In this figure,

Figure 10: Lines of Code



CPDT remains the lowest value of CLOC, which is 18% lower then HMDT and 13% lower then SPDT. However this metric only calculated the lines of code which are directly programmed or used in case study. None of the applications can be built with the support of only these lines. Therefore, size of the project (figure 7) has added as another critical index. The summarized size of Android Studio and Xcode projects is 97.8mb which is the 6% lower then CPDT and 34% lower then HMDT. From this aspect, it is easier to maintain single function in CPDT project and easier to maintain the whole project with SPDT project. The complexity level from figure 7 shows that the individual application from Xcode and Android Studio have the lowest level. Whereas their accumulation result becomes the 6, which is the highest level among all three categories. Same situation happens on coupling level. Even though they obtained the second grades individually this time, after sum up their results, SPDT gained the last position again. Meanwhile, applications from Xamarin project achieve best result of coupling not only individually but as representative of HMDT. Cohesion level are generally even for all six applications expect Android studio, which acquired half grade then others.

6.3 Summarized Result

Based on the unilateral comparison results from implementations and produced applications, an overall outcome presents in figure 11. This figure created by given three points to every horizontal analyse result in figure 7 and figure 6, two points for middle grade and one point for the worst one. Then accumulated these values based on different categories of development tools.

Figure 11: Summarized Result

	SPDT	CPDT	HMDT
Implementation			
1. Implementation hours(points)	1	3	2
2. Installation(points)	3	3	3
3. Easy to Use(points)	1	3	2
4. Productivity(points)	1	3	2
5. Integrated Simulator(points)	3	2	2
6. Memory and Powder Consumption(points)	1	2	3
7. User Interface Builder(points)	3	1	3
Sum(points)	13	17	17
Produced Application			
1. Execution Time(Points)	3	2	1
2. Security(Points)	6	5	7
2,1 Authentication (s)	0,3431	0,785	0,854
2,2 Encryption & Decryption(s)	0,072	0,087	0,066
2,3 Communication(s)	0,311	0,306	0,304
3. Maintainability (Points)	10	10	9
3.1 Size of Project(MB)	97,8	104,1	147,9
3.2 Lines of code	455 + interface builder	388	473 + Interface Builder
3.3 Complexity(CCN index)	6	5	4
3.4 Coupling(index)	1,7	0,94	0,87
3.5 Cohesion(LCOM index)	3	4	4
Sum(points)	8	6	5
Final Result	21	23	22

Worst | Best | Middle

From figure 11 it is clear that CPDT and HMDT show

higher performance than SPDT during implementation. However the produced applications from SPDT shows the best result, which is 33% higher then the CPDT applications and 60% higher then HMDT result.

7 Threats to validity

Although author has used objective metrics to collect data from all aspects, lack of data and bias data are still the biggest threat to this research.

7.1 Lack of data

There is no doubt that Android studio and Xcode should be represent on behalf of SPDT category. However there are numbers of mobile application development tools from the other two categories but only Xamarin and Cordova were selected from them. These two tools are known for their performances and powerful functionalities, whereas other tools from CPDT and HMDT categories might have different qualities. Therefore by using the data collected from only Xamarin and Cordova to represent CPDT and HMDT categories can lead to a narrow answer. A wilder range of representatives should be choose from both CPDT and HMDT categories to enrich data to provide more accurate result. Further more, increase functionalities of application can help to collect more data. Major data source of this research come from implementation time and execution time of each function. Adding more functions can help to increase accuracy of the final result.

7.2 Data bias

There are four projects conducted in this study, all of them were following the same project plan in order to produce six identical products. Author as the only programmer has involved in these projects. As development continues, programmer usually gain more knowledge and become more familiar with each function. This phenomenon might affect the data collected during implementation process, as in this case study, the first conducted project came from SPDT, which took the longest implementation time.

On the other hand, different experiences of development tools and related programming skills can directly affect product quality and development efficiency. Specially in this study, each developments tool required different programming language. And the only participated programmer does not have the same control of related language: Java, HTML, CSS, C#, Objective-C and Java Script. Hence the quality of produced applications and development times can be affect.

These issues can be solved by adding more developers to case study, who have evenly experiences of development tools and related programming languages. And developer only need to focus on one project in order to avoid the influence from other projects.

8. Discussion & Conclusion

The results show (see figure 11) that there is not an extreme difference in terms of development processes and finalized products by SPDT, CPDT and HMDT, but candidate from CPDT still obtained better results in these respects than the others. On the whole, CPDT has provided a better performance with the value of 23 points, which is 5% higher then CPDT and 10% higher then SPDT. The difference among the results of development processes (see figure 5 and figure 6) are major cause by the functionalities and principles of the development tools. As a matter of fact, SPDT can only produce either Android or iOS application each time and tools from other categories however can produce both Android and IOS applications at same time. Therefore the implementation time of SPDT has double the value of CPDT and HMDT (see figure 6). Evaluation of finalized products (see figure 11) shows that SPDT has produced the best performance applications in general. Applications from Xcode and Android Studio provided the fastest execution time and easiest maintainability. Which means the SPDT can not only server applications with fast speeds and best functionalities but can also provide the best environment for developer to maintain and update products.

Most of the existed literature within the same domain have provided similar conclusions. When evaluating development approach, Heitkötter et al. [8] provided the result that ”the maturity of crossplatform approaches reveals that native development is not necessary when implementing mobile applications. Even if only a single platform is to be supported, a cross-platform approach may prove as the most efficient.” When evaluating CPTD and SPDT with focusing on UX and resulting applications, Angulo and Ferre stated that ”there are more possibilities of getting a better UX by maintaining the control over interaction issues that provides the development of an app with native code” [2].

As mentioned in Section 7, the limitations of this study come from two part: lack of data and data bias. Limited number of development tools and functionalities of finalized applications might restricted data source. And the number of developer participated in case study might cause data bias. However by collecting data from all aspects with objective metrics, author tried to minimized the impact of restrictions.

Due to the limitation of the experiments, the result of this study can only applies on the project which has a limited numbers of functional requirements with the range from one to five. Project for example have more then five functional requirements or includes senior developer are out of range

of this study, therefore this paper might no longer provide any reference.

This research aims at helping junior developers understand different performances of SPDT, CPDT and HMDT when developing applications for both Android and iOS systems. Also provide reference for contact company Cybercome when choosing development tools for future projects. Not same as most of the related work present in Section 2, comparisons of development tools, implementation processes and finalized applications of native, cross-platform and hybrid developments are all included in this paper. As result shows (see figure 11), utilizations of CPDT and HMDT can definitely shorten development time and decrease related cost, but the applications produced from SPDT provide the best performance. This contrast phenomenon can be analysed in the future studies from numbers of aspect. For example, enhance the accessing criteria by adding factors that impact reliability, usability, user experience etc. Another idea is to increase assessment of application such as user interface responsiveness, user-perceived performance, multi task performance and communication failure handling. Moreover, create a credit system to present the importance of each category of data collected during case study for the purpose of brings out a more reliable result.

9 Acknowledgement

This paper has been written in cooperation with Cybercom Group, I would like to thank them specially Gabriel Ibanez who provided this topic and gave us continuously help as industry partner and technical supervisor. I would like to thank all the colleges from Gothenburg University and Cybercom who provided help during implementation phases. I would like to express our heartfelt gratitude to Riccardo Scandariato for his constant help and for his guidance and patience. In the end, I would like to thank my examiner Boban Vesin who provided invaluable feedback and guidance on this paper.

References

[1] S. 2016. Number of apps available in leading app stores as of july 2015, 2015.

[2] E. Angulo and X. Ferre. A case study on cross-platform development frameworks for mobile applications and ux. In *Proceedings of the XV International Conference on Human Computer Interaction*, page 27. ACM, 2014.

[3] F. Appiah, J. Hayfron-Acquah, J. K. Panford, and F. Twum. A tool selection framework for cross platform mobile app development. *International Journal of Computer Applications*, 123(2), 2015.

[4] N. Boushehrinejadmoradi, V. Ganapathy, S. Nagarakatte, and L. Iftode. Testing cross-platform mobile app development frameworks (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 441–451. IEEE, 2015.

[5] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International*, pages 323–328. IEEE, 2013.

[6] P. R. de Andrade, A. B. Albuquerque, O. F. Frota, R. V. Silveira, and F. A. da Silva. Cross platform app: a comparative study. *arXiv preprint arXiv:1503.03511*, 2015.

[7] G. K. Gill and C. F. Kemerer. Cyclomatic complexity density and software maintenance productivity. *Software Engineering, IEEE Transactions on*, 17(12):1284–1288, 1991.

[8] H. Heitkötter, S. Hanschke, and T. A. Majchrzak. Evaluating cross-platform development approaches for mobile applications. In *Web information systems and technologies*, pages 120–138. Springer, 2012.

[9] S. Jiang. Ios and android mobile application development tool decision, 2015.

[10] J. Masner, P. Simek, J. Jarolímek, and I. Hrbek. Mobile applications for agricultural online portals-cross-platform or native development. *AGRIS on-line Papers in Economics and Informatics*, 7(2):47, 2015.

[11] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm. A sloc counting standard. In *COCOMO II Forum*, volume 2007, 2007.

[12] J. Offutt, A. Abdurazik, and S. R. Schach. Quantitatively measuring object-oriented couplings. *Software Quality Journal*, 16(4):489–512, 2008.

[13] M. Palmieri, I. Singh, and A. Cicchetti. Comparison of cross-platform mobile development tools. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, pages 179–186. IEEE, 2012.

[14] J. Perchat, M. Desertot, and S. Lecomte. Common framework: A hybrid approach to integrate cross-platform components in mobile application. *Journal of Computer Science*, 10(11):2165, 2014.

[15] R. S. Pressman. *Software engineering: a practitioner's approach*. Palgrave Macmillan, 2005.

[16] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[17] B. S. Thakare, N. Parween, and S. Parween. State of art approaches to build cross platform mobile application. *Int. J. Comput. Sci. Eng*, 107:22–23, 2014.

[18] S. Xanthopoulos and S. Xinogalos. A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 213–220. ACM, 2013.