
# Codename One and PhoneGap, a performance comparison

**Andreas Arnesson**

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

**Contact Information:**
Author:
Andreas Arnesson Andreas_A@outlook.com

University advisor:
Nina Dzamashvili Fogelström
Department of Software Engineering

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

# ABSTRACT

Creating smartphone applications for more than one operating system requires knowledge of several code languages, more code maintenance, higher development costs and longer development time. To make this easier cross-platform tools (CPTs) exist. But using a CPT can decrease performance of the application. Applications with low performance are more likely to get uninstalled and this makes developers lose income.
There are four main CPT approaches hybrid, interpreter, web and cross-compiler. Each has different disadvantages .and advantages.
This study will examine the performance difference between two CPTs, Codename One and PhoneGap. The performance measurements, CPU load, memory usage, energy consumption, time execution and application size will be made to compare the CPTs. If cross-compilers have better performance than other CPT approaches will also be investigated.
An experiment where three applications are created with native Android, Codename One and PhoneGap will be made and performance measurements will be made. A literature study with research from IEEE and Engineering village will be conducted on different CPT approaches.
PhoneGap performed best with shortest execution time, least energy consumption and least CPU usage while Codename One had smallest application size and least memory usage.
The research available on performance for CPTs is short and not well done. The difference between PhoneGap and Codename One is not big except for writing to SQLite. No basis was found for the statement that cross-compilers have better performance than other CPT approaches.

**Keywords**: cross-platform tools, approach, performance, android

# Table of Contents

# 1 Introduction

By Q4, 2014 the smartphone market can be divided into five parts, Android, iOS, Windows phone, Blackberry and other OS's [30].

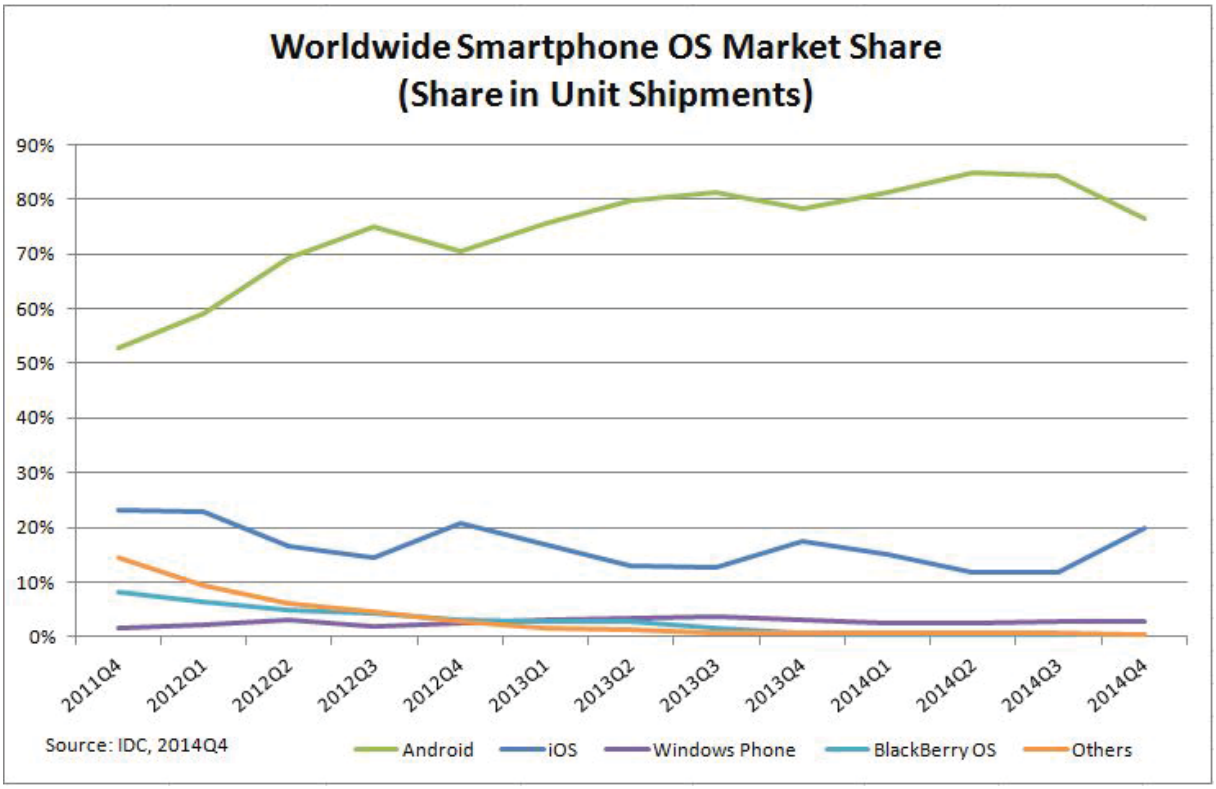



*Figure 1- Worldwide Smartphone OS Market Share.*

These four operating systems and others each use different languages to develop applications. For developers and companies who want to expand their market share beyond one OS, they need to maintain several projects instead of one. With this an increase of development time, development cost, maintenance and a bigger knowledge base is needed.

Luckily there is hope for developers because nowadays tools are available for developing applications for multiple OS's at once, so called cross-platform tools (CPTs). They have also been called Write once run everywhere. The pros of using CPTs are: [16]

- Less skill required from developer.
- Less coding needed.
- Less development time and maintenance cost.
- Less API knowledge.
- Easier to develop one app instead of one for each OS.
- More market share.

Another problem with smartphones is their limited resources, CPU, memory, battery and data.

The worst thing for a developer or company is for their application to get uninstalled from user's smartphones. There are a number of reasons for this to happen, abusing privacy, unstable behavior, advertisement, no need for it anymore, high memory allocation, poor interface, high battery usage, low performance and sluggish behavior e.tc. Some of these reasons, high memory allocation, poor

interface, high battery usage, low performance and sluggish behavior, correlate to performance, CPU usage, memory usage, execution time and battery consumption. This means that if an application has low performance that application has a much higher risk of getting uninstalled and then developers and companies loses income. This is why it's important for developers to always have in mind the resource use of their applications [31].

Using CPTs can come at a cost of resources. Using CPTs can decrease performance of application compared to native applications [21, 22]. That's why it is important to investigate the performance of CPTs compared to native, so developers can see if using a CPT will cause lower performance and in turn run a higher risk of getting their application uninstalled.

There are four main approaches to CPTs, hybrid, interpreter, cross-platform and Web. Each approach has its advantages and disadvantages. According to [21] the Cross-compiler approach has best performance but they haven't done any experiment to prove that statement nor do they reference it. [22], on the other hand say all CPT approaches have performance loss.

This paper will compare the performance of the two CPTs Codename One, a Cross-compiler, PhoneGap, a hybrid, and compare them to native Android. An experiment will be made where an Android application will be developed with Codename One, PhoneGap and native. Performance measurements will be taken while the applications are executed. The performances that will be measured are memory use, CPU use, energy consumption and execution time. The application size will also be compared.
This paper aims to display the performance differences between the CPTs and how they compare to native and show which CPT of the two has best performance. It will also show if there are any ground for [21]'s claim that Cross-compilers have better performance than other CPT approaches.
The Android OS was chosen because it was the only available smartphone and there wasn't time to learn another language.
A literature study has also been made. In the research study related work and information about the different CPT approaches has been found. I have also researched how to minimize code affecting performance.

Section 2, *Background and related work*, will introduce the concept of CPT some more, the Android platform and why I chose Codename One and PhoneGap. It will also contain relevant previous work that has been done, comparing the performance of CPTs.
Section 3, *Research questions and research design*, will present the research question for the literature study, how the study will be carried out and how the experiment will be performed.
Section 4, *Results*, presents the result of the literature study and the experiment results.
Section 5, *Analysis*, Contains an analysis of the experiment.
Section 6, *Conclusion and future work,* will present a conclusion of this paper, how this paper can be improved and what related things can be interesting to research in the future.

# 2 Background and Related work

## 2.1 Background

A CPT is a tool to help developers write applications with a consistent user interface for several Oss with one codebase. For example with PhoneGap developers create an application with HTML, JavaScript and CSS. The developer can then compile an application for each supported OS with that code.
A problem with CPTs can be the lack of support for native functionality, such as camera, GPS and accelerometer. With some CPTs, like Codename One, it is possible to write native code for each OS. For example Codename One doesn't have GPS functionality but it is possible to write native code, so if the developer wants an Android and an iOS application the developer can write native Android and iOS code to access the GPS on the phone.

### 2.1.1 Performance priority

Software optimization largely impact execution time and energy consumption, and these are the most important aspects of performance. Users want quick and responsive applications which don't drain the battery. High CPU usage means shorter execution time, which leads to lower energy consumption [36].
When deciding which CPT has the best performance, shortest execution time and lowest energy consumption are the most important aspects. If they can achieve those aspects and have lower CPU usage and memory usage it is even better. However if the low CPU and memory use comes at the cost of long execution time it is not desirable.

### 2.1.2 How my code affects performance

Most research I found was irrelevant as it was about how you affect performance on a hardware level.
Most papers I found that are not about hardware are about energy consumption.

My findings are:

1. Don't use setters or getters. [1, 5]
2. Don't use ".length" in *for loops.* [1,5]
3. Allocating more memory has little impact on energy consumption. [5]
4. To save power you should offload some of the CPUs work to the GPU. [3]
5. Over 69% of applications energy consumption happens in idle state. [4]
6. To limit the energy consumption in idle state you should optimize the color scheme. [4]
7. 85% of energy consumption that is consumed outside of idle state is consumed by API calls. [4]
8. HTTP request is the most energy consuming operation. [4]
9. The three most common bugs are:
    - Energy leak bugs – not tuning sensors or not disabling them.
    - GUI lagging.
    - Memory bloat bugs – Not reusing resources.
10. For PhoneGap
    - 12. Use "touchstart" .[33]
    - 13. Use CSS animations. [35]

- 14. Avoid using global variables. [35]
- 15. Memory management. [35]
- 16. jQuery is slow. [35]
- 17. Concatenate and minify JavaScript file.[35]
- 18. Use JSLInt. [35]
- 19. Create own Plugins. [35]

11. For Codename One [34]
    - 20. Create own ListModel and ListRenderer.
    - 21. Don't use String width function.
    - 22. Don't use Bitmap fonts.
    - 23. Don't draw small images.
    - 24. Done use Image.scale.
    - 25. Don't draw round rectangular borders.
    - 26. Don't use translucency on Symbian.

I will not utilize all of my findings. I will explain which in the experiment design.

**Research**

Rodrigues Tonini, A. et al. [1] evaluated some of the best practices given by the Android developer site. [2] concluded that code without getters/setter and *For loops* without ".length" are faster and more energy efficient.

D. Li and W. G. Halfond [5] test common tips used for better performance. They found that allocating more memory has little impact on power consumption. Developers shouldn't be afraid of allocating memory although it is always good not to allocate unnecessary memory.
*For loops* which don't access the arrays length in the body has less energy usage. Accessing fields directly instead of using setters and getters also gives less energy usage. Using static invocations instead of virtual also gives better performance.

In the research by Hung-Ching Chang et al. [3] they reach the conclusion that developers should offload some of the CPUs work to the GPU and schedule the work to one of the CPUs, letting the other CPUs save power.

Ding LI et al. [4] conducted an extensive study on energy consumptions. According to their research over 69% of applications power consumption happens in idle state and to diminish this, developers should optimize their color scheme. They also report that 85% of the energy consumed by applications when not in idle state is consumed by calls to the Android API. 13% is consumed by the developer code. Therefore developers should optimize their API usage to reduce energy consumption. HTTP request is the most energy consuming operation. They also show that using milliseconds to measure energy consumption can lead to faulty results. They propose using nanoseconds instead.

This means that the way I write my code has the least impact on the energy consumption if my application does API calls and goes into idle state.

In the paper written by Yepang Liu et al. [6] they discuss and give solutions to performance and energy bugs. The bugs are energy leak bugs, such as tuning sensors frequency or not disabling them temporarily, GUI-lagging and memory bloat bugs such as not reusing resources.

For PhoneGap:

- Applications you should use “touchstart” instead of “click”. “click” adds an 300 milliseconds delay before it handles the event. [33]
- Use CSS animation instead of using JavaScript to manipulate DOM.
- Avoid using global variables.
- Help JavaScript with memory management by using delete or assigning functions to null when they are of no use anymore.
- jQuery is a popular framework  to use but it can slowdown the application.
- Concatenate and minify the JavaScript file.
- Use JSLint to find syntax wrongs and optimization tips.
- Using plugins for CPU intensive activities is great to split up the work. [35]

For Codename One there are several tips on their homepage. [34]

- Create your own Listmodel, because the default one creates an event for all transaction in the list, i.e. add/remove item from list. This slows down performance.  The same goes for the ListRenderer.
- Don’t use String width as it is very slow on some OSs.
- Don’t use Bitmap fonts, they are slow.
- It is faster, CPU wise, to draw larger images than small images.
- Using Image.scale slows down performance because normally a picture takes up the size of the compressed file but if it is scaled it takes up size of width*height*4.Instead use drawImage.
- Don’t draw round rectangular borders. Use nine-piece images.
- If you develop for a Symbian device don’t use translucency.

### 2.1.3 Different CPT approaches

The different CPT approaches utilize different technologies to create applications, from one codebase, that can be executed on multiple OS’s.

There are four main CPT approaches:

- **Web**
  - Creates applications that executes in the web browser. [21]
- **Hybrid**
  - Creates applications that are executed locally in the web browser and uses JavaScript plugins as a bridge to native features. [22]
- **Interpreter**
  - Creates applications that are interpreted and executed during runtime.[21]
- **Cross-compiled**
  - Compiles applications to native binaries for each supported OS. [21]

### 2.1.4 Why I chose these CPTs

#### 2.1.4.1 PhoneGap

PhoneGap is one of the most popular tools on the market [15, 14] and I have found the most research about it [12, 14, 15, 16, 17, 20]. I will use PhoneGap, because it is popular, and see how the other Codename One compares to what is considered the best one [14].

PhoneGap was released in 2008 by the company Nitobi but was acquired by Adobe in 2011 and was contributed to the Apache Software foundation in 2012. It uses a hybrid approach. PhoneGap can create applications for iOS, Android, Blackberry, Windows Phone, Ubuntu and Firefox OS. [12]

PhoneGap applications are written in HTML, JavaScript and CSS. A lot of community created plugins are available to use. It uses JavaScript to bridge to native code.

#### 2.1.4.2 Codename One

Codename One calls themselves “write once, run everywhere”. It’s a cross-compiler. Applications are written in Java, and then it builds a native application out of Java byte code.
Codename One works on the OS’s Android, iOS, BlackBerry, Windows Phone, and J2ME. [23]

I chose Codename One because I have found no research papers about it and, according to themselves, applications that have been made with Codename One has been installed over 75,000,000 times and there are over 15,000 developers using Codename One. [24]

### 2.1.5 Android

Android is an open source OS for smartphones, tablets, TVs and wearable devices.
Android is led by Google Inc.
The platform is based on a Linux kernel with C and C++ libraries. Applications are executed in the Dalvik Virtual Machine, DVM.

In figure 2.1 we can see a picture of the Android software stack.

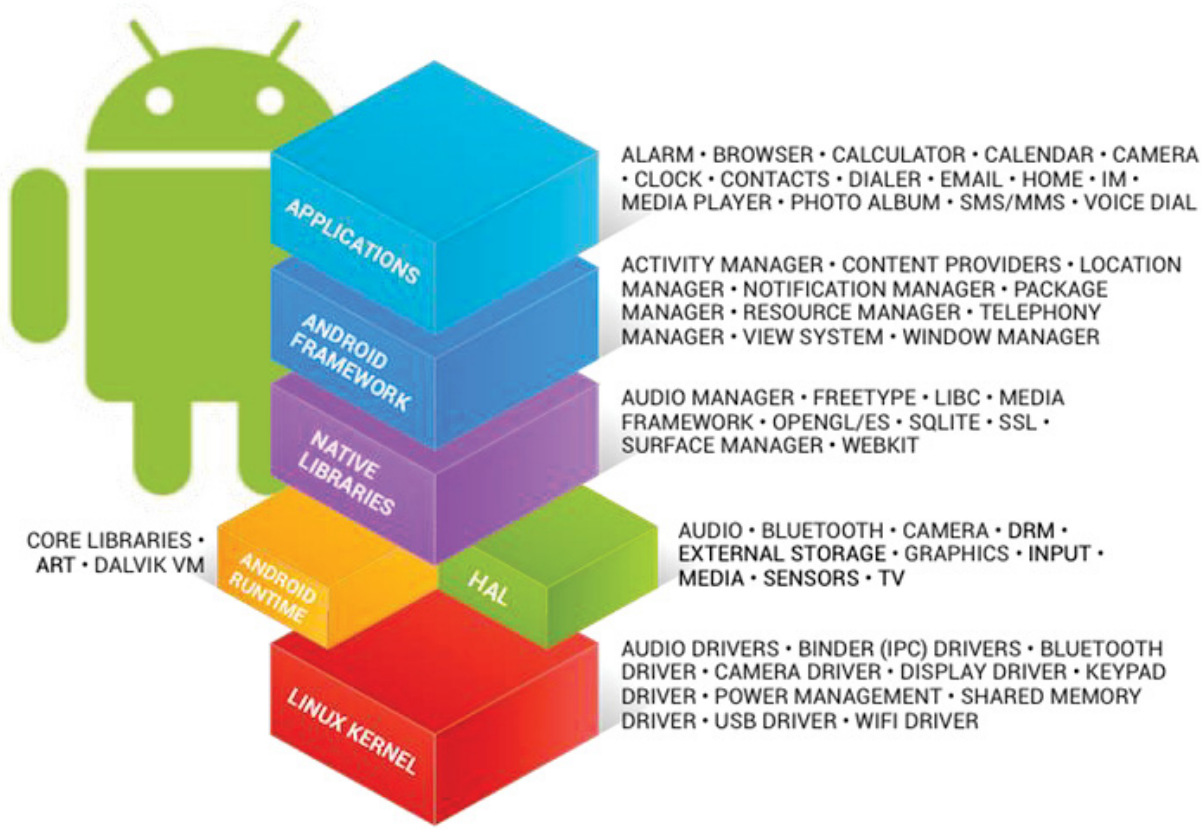



*Figure 2.1 - Android software stack.*

Android was released in September 2008 and since its release several new versions has been released. [32] The major releases are:

| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0.3% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 5.7% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 5.3% |
| 4.1.x | Jelly Bean | 16 | 15.6% |
| 4.2.x | | 17 | 18.1% |
| 4.3 | | 18 | 5.5% |
| 4.4 | KitKat | 19 | 39.8% |
| 5.0 | Lollipop | 21 | 9.0% |
| 5.1 | | 22 | 0.7% |

*Figure 2.2 – Usage distribution for Android versions June 1, 2015 [37].*

## 2.2 Related work

Using CPTs benefit both developers and users [11, 13, 16], but it can be hard for developers to know which CPT they should choose.

As we can see in the table 1.1, PhoneGap is the most popular CPT to research.

| CPT used in paper | Number of papers | Papers |
|---|---|---|
| PhoneGap | 7 | 11,14,15,16,17,19,20 |
| Titanium | 5 | 13,14,17,19,20 |
| Rhodes | 4 | 13,15,16,20 |
| MoSyc | 3 | 13,16,20 |
| DragonRad | 2 | 15,164 |
| Mobl | 1 | 15 |
| Mdsl | 1 | 15 |

*Table 2.1 – CPT distribution in references.*

In the table 1.2 we can see that the most popular ways to compare CPTs are via functionality.

| How they compare | Number of papers | Papers |
|---|---|---|
| Usability | 2 | 11,17 |
| Performance | 4 | 13*,17**,19***,20**** |
| Functionality | 5 | 14,15,16,19,20 |

*Table 2.2 – Ways references compared CPTs*
**They don't say what or how they test performance.*
*** Only compares memory usage of two CPT's*
**** Only compares energy consumption.*
***** Very small applications tested on, no real use cases for the tests.*

**Research**

Mesfin, G. et al. [11], researched the usability of the CPT PhoneGap [12].They created one application for Android, Windows phone and Blackberry with PhoneGap. They let nine people test the apps and answer some questions. They reached the conclusion that using CPTs benefit both developers and end users from a usability perspective. Developers can write less code and still develop usable applications for the end user.

CPTs are a good cost-efficient substitute to native development, but still there are some problems. It can be hard to choose which CPT to use. When choosing a CPT it is important to know that it supports all the features and functionality the developer needs and that can be hard to find out [13]. [13] Used three CPTs, Titanium, Rhodes and MoSync to create three applications with every CPT. They used those applications to perform an evaluation study where they tested to see if the users noticed any difference in the applications. Nine people tested their applications. The users were to answer which application they would prefer to use judged on interaction-response time and satisfaction with the applications.
They also talk about performance but they don't say how they tested it or what kind of performance they tested, only that MoSync performed worst in one of their applications.

Li Tian et al. [14] Explains how PhoneGaps underlying framework works. How it uses HTML, JavaScript and CSS to create an application in the smartphones browser and uses JavaScript to call the native API.

[15] Compares several CPTs, Rhodes, PhoneGap, DragonRad, Titanium, mobl and mdsl. They compare how the underlying techniques used by each CPT works, which platforms are supported, what development environments are available, features of the CPTs and access to device capabilities. PhoneGap is a "Web code wrapper", which spits out a hybrid application where the layout is in the browser and the JavaScript is used to link the web code to native. That is how PhoneGap gets access to native functionality.
PhoneGap has access to Accelerometer, Camera, Contacts, Media and GPS.
PhoneGaps main focus is web developers whom want to create native applications.

[16] Compare the CPTs Rhodes, PhoneGap, DragonRad and MoSync. They just as [15] compare platform support, languages, ides, licenses and access to device capabilities.

[17] Performed a performance evaluation between PhoneGap and Titanium. They also used PhoneGap with Sencha 2.0 and JQuery mobile to enhance PhoneGaps UI development. It shows the different advantages of native, Mobile web and cross platform.
They compare the performances Power consumption, CPU usage and Memory usage.
They test their applications by doing requests, AJAX, REST and SOAP, to web services and displaying the response in the formats Text, XML and JSON.
They achieved better memory usage with PhoneGap than with Titanium [18] but PhoneGaps UI wasn't as good as Titanium. The user experience wasn't as good as with native.
Their result is inconclusive as they didn't use Titanium in the comparison of CPU and energy usage. It was only concluded that PhoneGap uses less memory than Titanium. When they measure CPU and energy consumption they only do that on PhoneGap with different frameworks to enhance the PhoneGap applications UI.

Ciman and Ombretta [19] studied energy consumption of common sensors used in applications created with Titanium, PhoneGap and native Android. They created apps that used the device sensors. Their result shows that native has least energy consumption and Titanium performed better than PhoneGap overall, except for the accelerometer where PhoneGap both performed better and had more configuration options.

Ohrt and Turau [20] conducted a study on nine CPTs and native android. They compared functionality, features and performance of the CPTs. When comparing performance, required ram, apk size and launch time, they created a small application which only contained an icon and a text label. The results they produced with their experiment are very limited and don't necessarily represent the performance of a complete application created with the CPTs. Their result only shows the performance overhead each CPT has for displaying a text label. They don't state which CPT had the best performance, but after looking at their charts it looks like Illumination has the best performance in their experiment. I can't find which CPT approach Illumination uses. The CPT that has best performance after Illumination is MoSync. MoSync consists of several CPT approaches, runtime interpreter, source code translator and hybrid [13].
According to [20] the CPT that comes after MoSync is LiveCode, which is an interpreter, and after that comes PhoneGap. The cross-compiler that is highest on their list is Marmalade and it is two

spots below PhoneGap.
-

**Analysis and Summary**
There are very little and lacking research on performance. Only [17] did a bigger and more thorough performance experiment, but they only compared memory usage between CPT's. [13] doesn't state what performance or how they measure it, [19] only measures energy consumption and [20] only creates small "hello world" applications. This leads to a lacking coverage of CPTs performance research.

[17] is the paper that comes closes to this paper, but they are focused on sending data with different protocols via HTTP to web service. With my experiment I am trying to achieve a much broader coverage of the things you can do with an application created with a CPT.

[19]'s result shows what [17] failed to show. Titanium's energy consumption compared to PhoneGap, maybe only memory usage is what PhoneGap has better than Titanium.

When comparing [21]'s statement, about cross-compilers having best performance, with the results from the papers in Related work it looks like [21] doesn't have any basis for their statement. [13] states that native has best performance, still we don't know what kind of performance. [17] concludes that PhoneGap, hybrid, has less memory usage than Titanium, cross-compiler. According to [19] Titanium has less energy consumption than PhoneGap. Since [20] doesn't state which they believe have best performance I had to make assumptions based on their chart. After examining the chart, the cross-compilers didn't perform better than the other approaches.

Both 11 and 16 concludes the reasons and advantages of using CPTs, e.g. less code, less time and less costs.

# 3 Research design

## 3.1 Literature design

Before starting the experiment I embarked on a literature study where I wanted to find out:

- **RQ1:** What is the difference between different cross-platform approaches?

I want to know what different techniques there are for CPTs, how they work and if I can expect a certain result depending on which approach my CPTs are using.

To answer the question I searched the databases IEEE and Engineering village with the search strings:

- **(mobile OR android) AND ((cross platform tool) OR (cross-platform tool))**

I haven't searched for cross-platform approach because that isn't always the used word, instead I searched for CPT because it is broader and all papers about cross-platform approaches should contain CPT. It is easier to filter out irrelevant research than to find all the different names used for cross-platform approaches.

## 3.2 Experiment design

With this experiment I want to find out:

- **RQ2:** Which CPT had best performance, PhoneGap or Codename One.
- **RQ3:** Is my result in line with cross-compilers having better performance than other approaches.

[21] Made the statement that cross-compiler has better performance than other approaches, but they didn't have any references for it nor did they perform any experiment themselves.

Three applications will be written, one in native Android, one with PhoneGap and one with Codename One.

I will measure:

- CPU load
- Memory use in KB
- Application size in MB
- Energy consumption in Joule
- Execution time in MS

The applications will be divided in activities, during every activity CPU, memory use and execution time will be measured. Energy consumption will be measured from when I start the application until I close it.

I will use two Smartphones to conduct my experiment on. I chose those two phones because that is what I had available to use. I chose to use two instead of one because I wanted a bigger spread on the result, I didn't want to limit the experiment to just show what the result is on one specific phone. Instead the experiment will give a more general result and show that performance can differ between devices. I will use:

- Samsung Galaxy trend plus – 2013, runs Android 4.1 (Jelly Bean), CPU-dual core 2.3 GHz, RAM- 766MB
- Sony V – 2012, runs Android 4.3 (Jelly Bean), CPU-dual core 1,5GHz, RAM-1024MB

### 3.2.1 How to test my applications:

To find what kind of tests are commonly used when testing mobile applications I did some research to find how other researchers have tested theirs and I have also been in contact with a local company which develops Android and iOS applications, both native and with PhoneGap. I use their suggestions because it is interesting to know what a real company find interesting when they want to see performance tests.

When deciding which tests to do I compiled what had been done previously and what the local company found interesting to test. I chose the tests I found the most interesting and the tests that test different operations.

- For integer calculation test I will find prime numbers between 2 and 30000 using Sieve of Eratosthenes and randomize 100000 integers.
- To test memory access I will do a Bubble sort of 5000 integers and also sort 2000 Strings with each tools own String array sort.
- To test read and write "Big data" I will write and read 1000 integers to/from SQLite.
- To test loading graphical components and String manipulation I will create and display a list of 2000 String.
- To test interaction to the native API I will retrieve my position with GPS. The SQLite test also tests interaction with the native API.

Bubble sort is a sorting algorithm where you repeatedly goes through a list and compares adjacent items and exchange them if they are in the wrong order. The code for it can be found in appendix A. How many elements that were used in the different tests is a compromise between getting close to out of memory and that the execution time isn't longer than three minutes. For the bubble sort and the write to SQLite I wrote a simple program in Java where I used to built-in randomizer to generate integers. To generate the string for the list phases I used [38].

I will prepare my devices for the tests by:

1. The battery is fully charged
2. The device has been rebooted
3. All optional background processes have been terminated. Only the programs necessary for the operating system to be functional remain active
4. Screen timeout has been deactivated
5. Synchronization has been turned off

I plan on arrange my applications according to the picture.

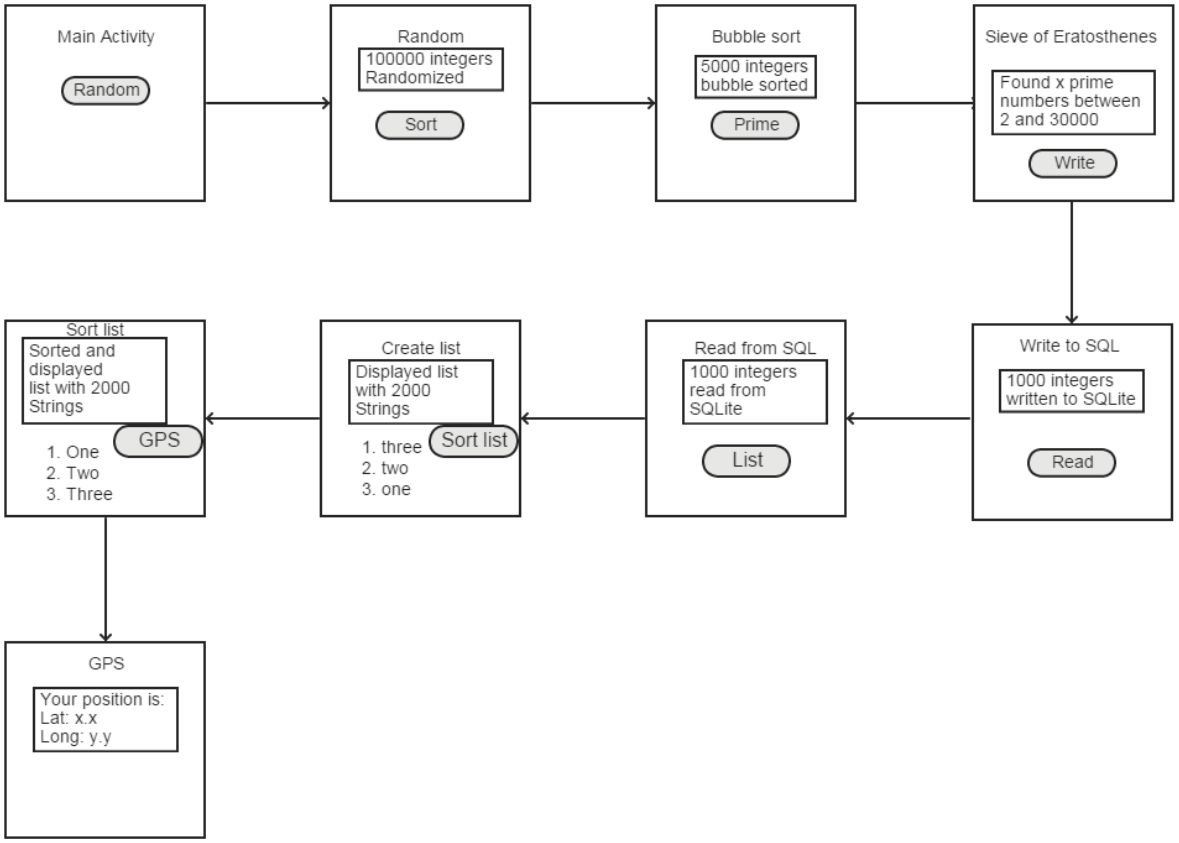



*Figure 3.1 – Application layout.*

**Research**

The local company had input on what is interesting to test. According to them it is interesting if the GUI lags, to read in "big" data, sort it, get position and load graphical components. They also felt that 3D and image processing is irrelevant because that is only important when creating games.

Cheng-Min et al. [7] test their applications by doing Numeric calculation with recursion, Library facilities, User made data structure, Polymorphism, Nested loops, Random number generation, Sieve of Eratosthenes and String operations.

Jae Kyu Lee and Jong Yeol Lee [8] use JNI delay, Integer & Floating-point calculation, Memory access, String Processing to test their applications.

Lewerentz and Linvall [10] researched how performance differs between native C and Java code. They test their applications with Integer calculations, Floating-point calculations, Memory access operations and recursive operations. They do those tests with the algorithms sieve of Eratosthenes, trigonometric functions, bubble sort and quicksort.
Before every experiment they prepare their smartphones by:
1. The battery is fully charged
2. The SIM card has been removed

3. The device has been rebooted
4. All optional background processes have been terminated. Only the programs necessary for the operating system to be functional remain active
5. WiFi, Bluetooth and GPS have all been deactivated
6. Screen timeout has been deactivated
7. Synchronization has been turned of

### 3.2.2 Validity threats

**My code**

There is always the possibility that my code contains a bug or that it is not optimized enough.

**Different devices**

The devices used in the experiment have different hardware and this can affect the results from the experiment.
Having different versions of the OS is a risk because there might be something different about them that make the result differ, but this isn't necessarily a risk. Instead it can be interesting to see how the performance can differ between versions of the OS. The problem is that I won't know if this will be a reason for different results.
Using two devices might not be enough to generalize the result. Also the devices have similar versions of Android. It would be more interesting to see an experiment where the versions were more different.

**Measurements**

The energy consumption measured with PowerTutor might not be correct because it was created for HTC G1, HTC G2 and Nexus one. Therefore the results are a rough estimate.

**Android prioritizing**

The Android system chooses how to prioritize CPU use for applications and system processes, if a system process needs the CPU while a test is running it can affect the result negatively. I minimize this by preparing the devices so that there are fewer processes that might need the CPU while I run my tests.

**Different OSs**

In this experiment Android is the only OS that will be tested. Therefor this experiment can't guarantee that the result would be the same for other OSs. One CPT might be better with Android but not with iOS.

**Java garbage collector**

The garbage collector is self-regulated which means I can't control when it runs. This can affect the outcome of the memory usage measurements.

### 3.2.3 Performance tips from research

In 2.1.2 I found practices about writing code that doesn't affect performance negatively. I will use the following findings:

- *Finding 1*, don't use setters or getter. I will utilize as much as possible, but as my applications aren't very advanced or complex I will not have the need for getters or setters in my code.
- *Finding 2,* Don't use ".length" in *for loops*. I will take advantage of this completely.

- *Finding 7*, minimize API calls. I will not use API calls in my code, that isn't necessary for the experiment, but the CPTs might do API calls in the background. If one of the CPTs do a lot of API calls it could be a reason for why one application has more energy consumption than the other. I won't know this because I will not look into how or why the application drains energy.
- *Finding 9*, common bugs. I will only use GPS as a sensor and it will be the last thing I do in my experiment so I don't need to think about tuning it or disable it. GUI lagging will probably not be a big concern for me as my GUI will be very simple and I will reuse as much resources as I can.
- *Finding 12*, use "toutchstart". I will use this for pressing buttons in the PhoneGap application.
- *Finding 14*, don't use global variables. I am going to try my best to avoid global variables.
- *Finding 17, c*oncatenate and minify JavaScript file. I am going to utilize this.
- *Finding 18*, Use JSLint. I am going to utilize this.
- *Finding 20*, create own ListModel and ListRenderer. This I am going to do since I will use a big list.
- *Finding 22*, don't use Bitmap fonts. I am going to utilize this.
- *Finding 25*, don't draw round rectangle borders. I am going to utilize this

### 3.2.4 Measuring tools

I will use two tools for the measurements.
I will use PowerTutor [26] to measure energy consumption. I chose Power tutor because I want to measure how much energy an application consumes and according to [37] Power tutor is the best choice. It measures energy consumption in Joules on a hardware level for Android smartphones. It can measure use from individual applications and the Android system. It was developed for HTC G1, HTC G2 and Nexus one so the measurements can be imprecise. Therefor the measurements should not be taken too seriously, but instead used as an indication on how the applications differ in energy consumption.
The second tool I will use is Trepn Profiler 5.1 [27]. I chose Trepn because it is developed by Qualcomm, a company which creates hardware for smartphones. I was originally going to use this tool for energy measurements as well but the tool wasn't able to measure energy consumption on the Samsung phone. It is a performance profiling application which can measure applications GPU/CPU frequency and load, memory usage, energy consumption for certain phones, Wi-Fi and cellular data and more. It collects data every 100 millisecond.
Memory use is calculated with "(number of pages the application has in RAM) * (page size) / (total available RAM)".
I chose to measure CPU load because it's the only one that can be measured for individual applications.
CPU load is calculated with the data from "/proc/[pid]/stat".

# 4 Results

## 4.1 Literature review

When I searched for papers about performance i got 1956 hits.

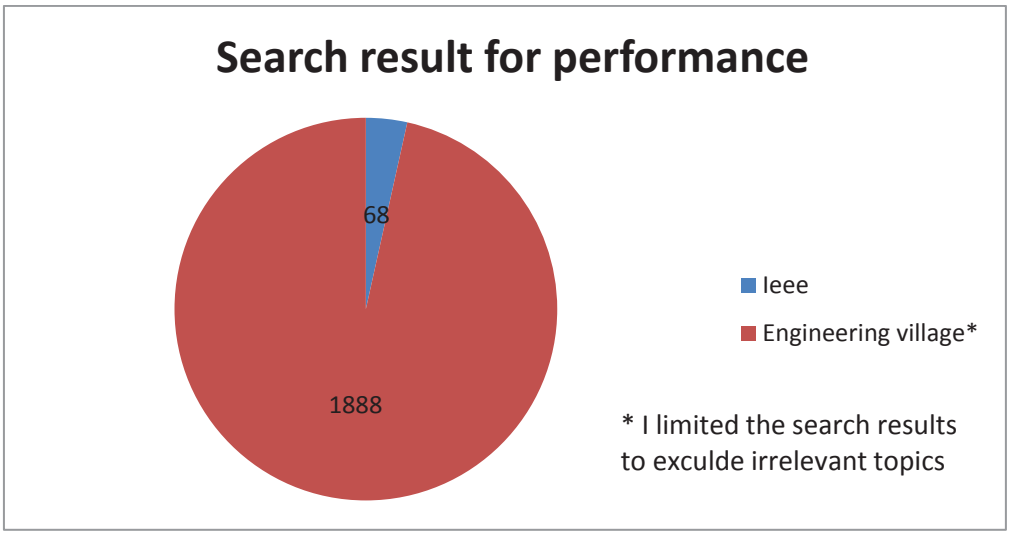



*Figure 4.1 – Search result for performance.*

And out of those 1956 I found 30 to be relevant when I read their abstract. After reading them completely I only found 9 that I used when writing about performance. Two of those 30 I also used for how to benchmark test my applications.

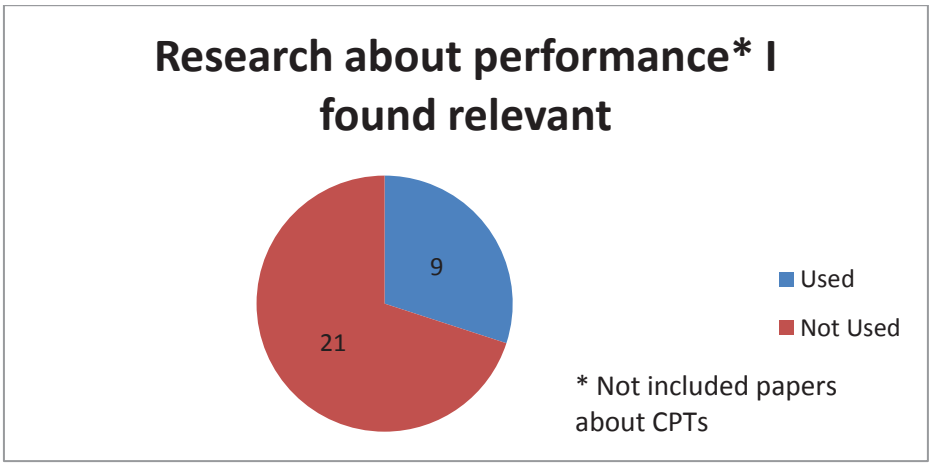



*Figure 4.2 – Relevant research about performance.*

When I searched for papers about Cross-platform tools i got 189 hits.

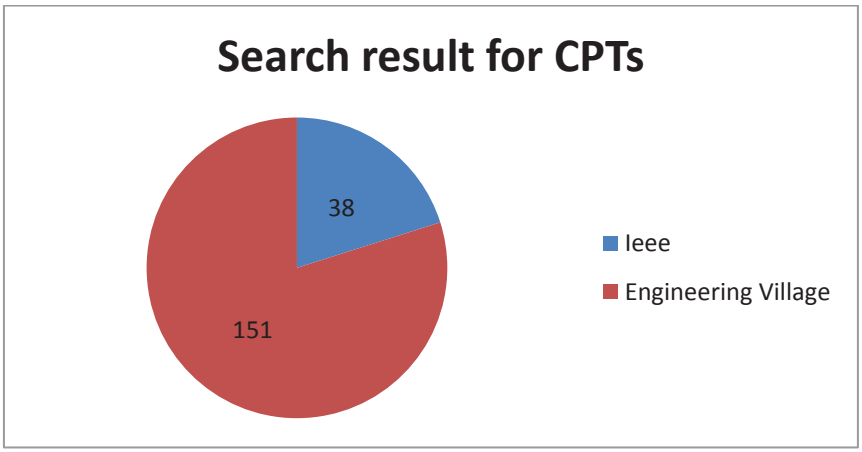



*Figure 4.3 – Search result for CPTs.*

Out of those 189 hits I found 26 to be relevant after reading their abstracts. I used 12 of the 26 papers.

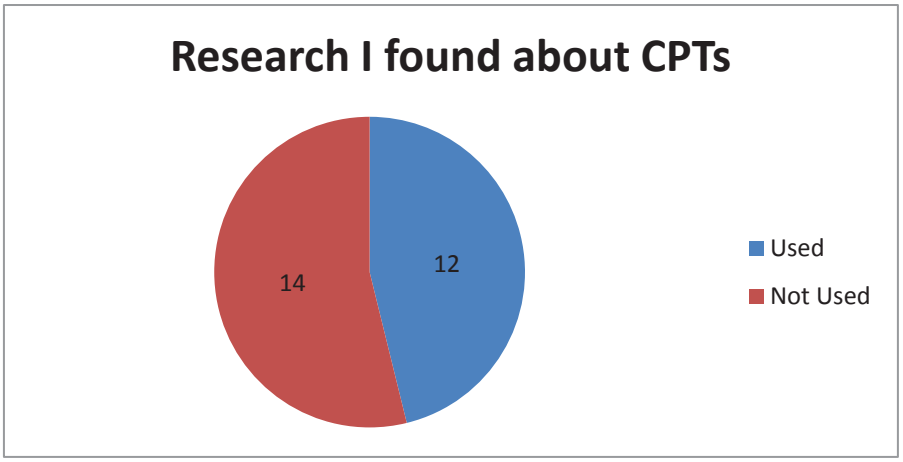



*Figure 4.4 – Relevant research about CPTs.*

### 4.1.1 Answer to literature question

There are four main approaches to cross-platform, there are *Web, hybrid, Interpreted* and *Cross compiled*. Each approach has its purpose, disadvantages and advantages.

| Advantages | Web | Hybrid | Interpreted | Cross compiled |
|---|---|---|---|---|
| Doesn't require installation | x | | | |
| Distributed with application stores | | x | x | x |
| Interface reused across platforms | x | x | | |
| Can use native capabilities, i.e. sensors | | x | x | x |
| Native look and feel | | | x | x |

*Table 4.1 – Advantages of CPT approaches.*

[21] Say that cross-compilers have the best performance compared to other CPT approaches, but they don't have any evidence to back-up their statement. They haven't done any experiments themselves and neither has their sources nor have their sources made that statement.
According to [22] hybrid is the best option when creating cross platform applications, but no matter what CPT method is used there will be performance reduction compared to if native had been used.

In my experiment a hybrid CPT (PhoneGap) and a Cross-compiler CPT (Codename One) will be used. It will be interesting to see how my results correlate to [21]'s statement. If Codename One proves to be better than PhoneGap that could be used to back-up their statement, but maybe the performance depends more on the implementation of the CPT. A much bigger study would need to be carried out, where at least two CPTs of every method will be used, to make any real conclusions.

**Research**

***Web approach***

CPTs which use the Webb approach create an application that is executed in the smartphones web browser, is platform independent and server driven. [21] Its main disadvantage is that it can't use the native graphical capabilities and functionality. It also requires a network connection. [22]

***Hybrid approach***

It is a hybrid between web and native. It's a web application that is run locally and can use native features. It uses the web render engine to display HTML and CSS and a JavaScript to access hardware components. [22] It gives the advantages of the browser and the native capabilities. It uses an abstraction layer to access native capabilities. Hybrid applications have inferior performance to native because it is executed in the browser. [21]

***Interpreted***

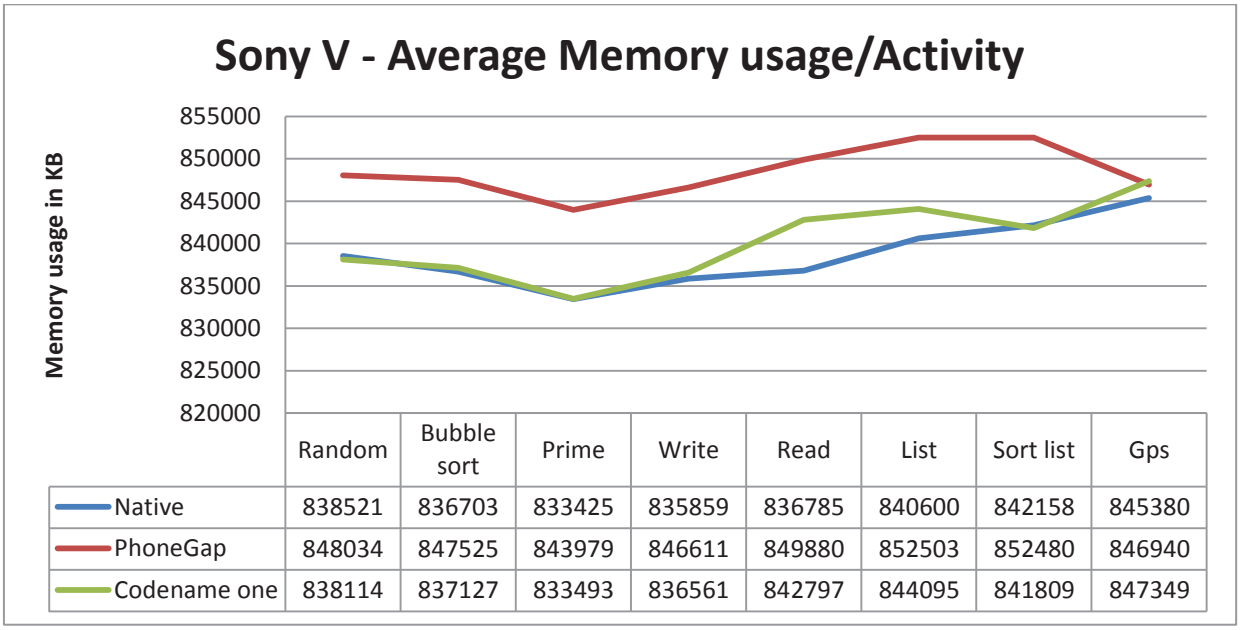
Applications are written in a scripting language and are interpreted and executed at runtime. The application uses an abstraction layer to get access to the native Capabilities. Applications can have inferior performance because of the runtime interpretation. [21]

***Cross-compiled***

Applications source code gets compiled to native binaries. The application is developed with one programming language and at compile time the code gets compiled to the native code. Has best performance out of the different cross platform approaches. [21]

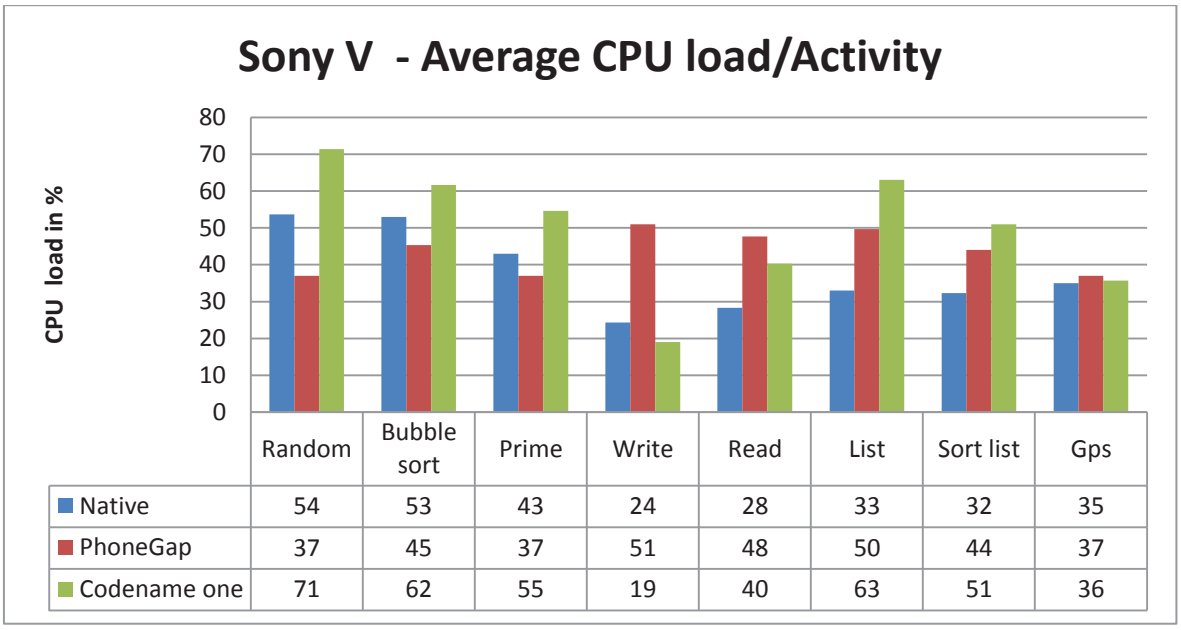
## 4.2 Experiment

### 4.2.1 Result from Sony V




| | Random | Bubble sort | Prime | Write | Read | List | Sort list | Gps |
|---|---|---|---|---|---|---|---|---|
| Native | 838521 | 836703 | 833425 | 835859 | 836785 | 840600 | 842158 | 845380 |
| PhoneGap | 848034 | 847525 | 843979 | 846611 | 849880 | 852503 | 852480 | 846940 |
| Codename one | 838114 | 837127 | 833493 | 836561 | 842797 | 844095 | 841809 | 847349 |

*Figure 4.5 –Sony V, Average memory usage/Activity.*




| | Random | Bubble sort | Prime | Write | Read | List | Sort list | Gps |
|---|---|---|---|---|---|---|---|---|
| Native | 54 | 53 | 43 | 24 | 28 | 33 | 32 | 35 |
| PhoneGap | 37 | 45 | 37 | 51 | 48 | 50 | 44 | 37 |
| Codename one | 71 | 62 | 55 | 19 | 40 | 63 | 51 | 36 |

*Figure 4.6 – Sony V, Average CPU load/activity.*

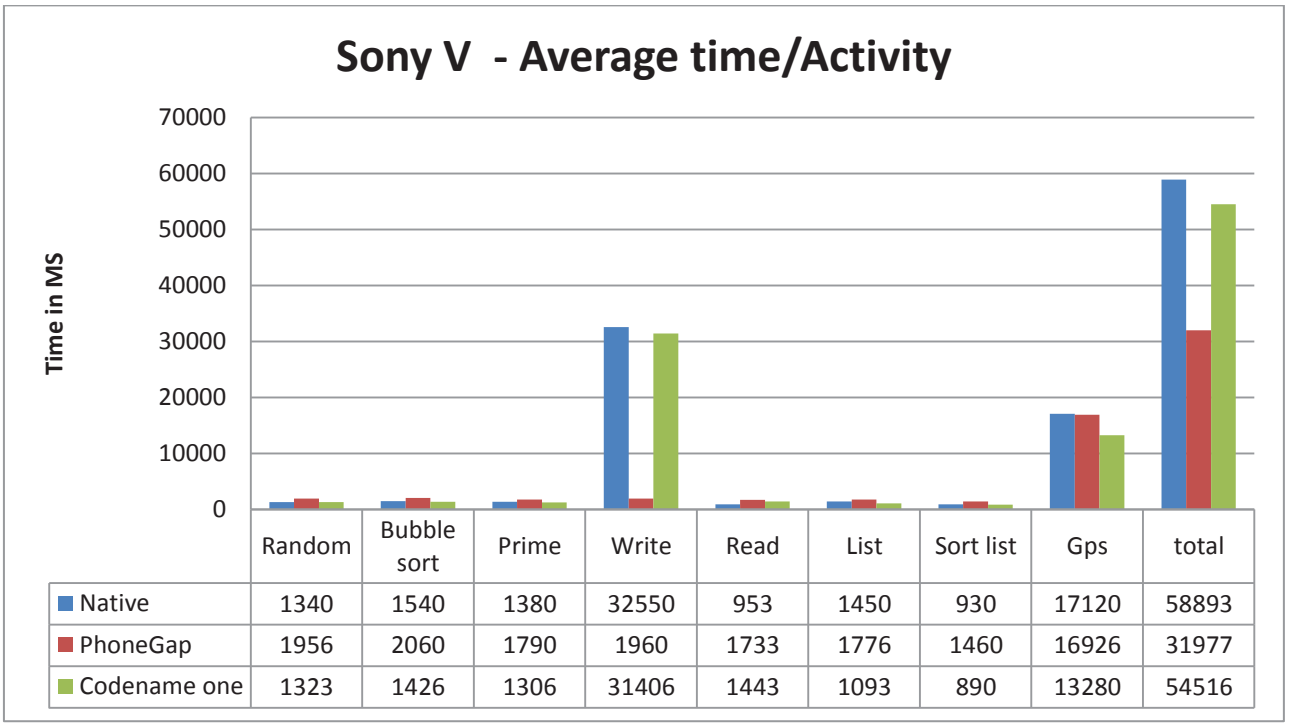



*Figure 4.7 – Sony V, Average execution time/Activity.*

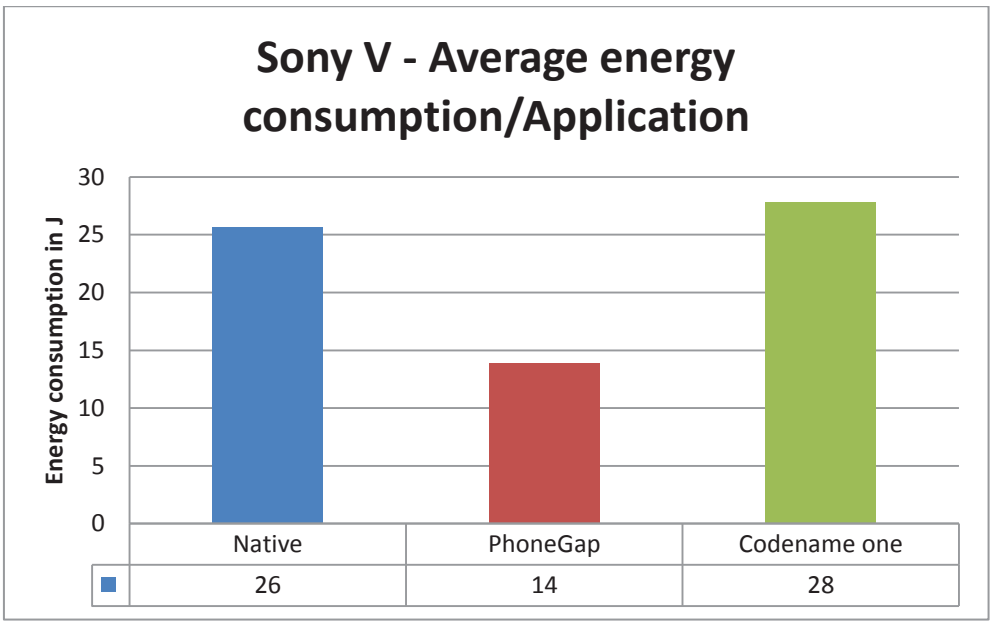



*Figure 4.8 – Sony V, Average energy consumption/Application.*

The low energy consumption for PhoneGap is a consequence from it having much shorter execution time than the other two during the Write to SQL phase. This is a consequence of PhoneGap using over double the CPU during that phase.

The amount of CPU used correlates to the execution time, more CPU equals less execution time. This is shown in the CPU and time chart forPhoneGap.

For the GPS phase memory usage, CPU usage and the execution time are very similar for all three applications. This is because only an API call to the same thing is made and then does the same thing for all three.

For the rest of the activities Codename One is using a little more CPU than native and it also has shorter execution time for the activities. PhoneGap uses second most CPU but at the same time it has a little longer execution time for its activities.

Codename One had higher energy consumption than native even though native had longer execution time. This shows that not only execution time impact energy consumption. One possibility is that the higher memory usage for Codename One is making it drain more energy.

The three of them show very little performance difference for making API calls. PhoneGap seem to have a memory overhead compared to native and Codename One.

PhoneGap clearly performed worst in memory use, which gives the impression that PhoneGap isn't good for memory intensive tasks. Native and Codename One got very similar numbers except during the read and display list phase.
Since Codename One compiles its code to native, unless if it isn't optimized, it is expected they have similar performance numbers.
The reason for PhoneGaps high memory usage can be because the application is executed in the browser and the browser adds the extra memory usage compared to the other two.
The three mostly follow the same pattern, their memory usage increases and decreases at the same places, but the amount of memory used is only similar during the GPS phase.

The CPU chart, figure 4.6, shows that Codename One uses most CPU. PhoneGaps CPU use doesn't fluctuate as much as for native and Codename One's CPU use. When they performed the Write to SQL PhoneGap used double as much CPU as the other two.

For the activities the execution time doesn't vary much between the applications except for the Write to SQL phase where native and Codename One are a lot slower than PhoneGap. This leads to PhoneGap having shorter execution time and less energy consumption.

The energy consumption chart shows, figure 4.8, PhoneGap used the least energy during its execution and Codename One had highest. The biggest reason for this is that it had a much faster execution time. It would have been interesting to see the energy consumption if they had similar execution time. Then the execution time wouldn't have such big impact on the result and we would be able to see how they used memory and CPU impact the energy consumption.

**Summary**

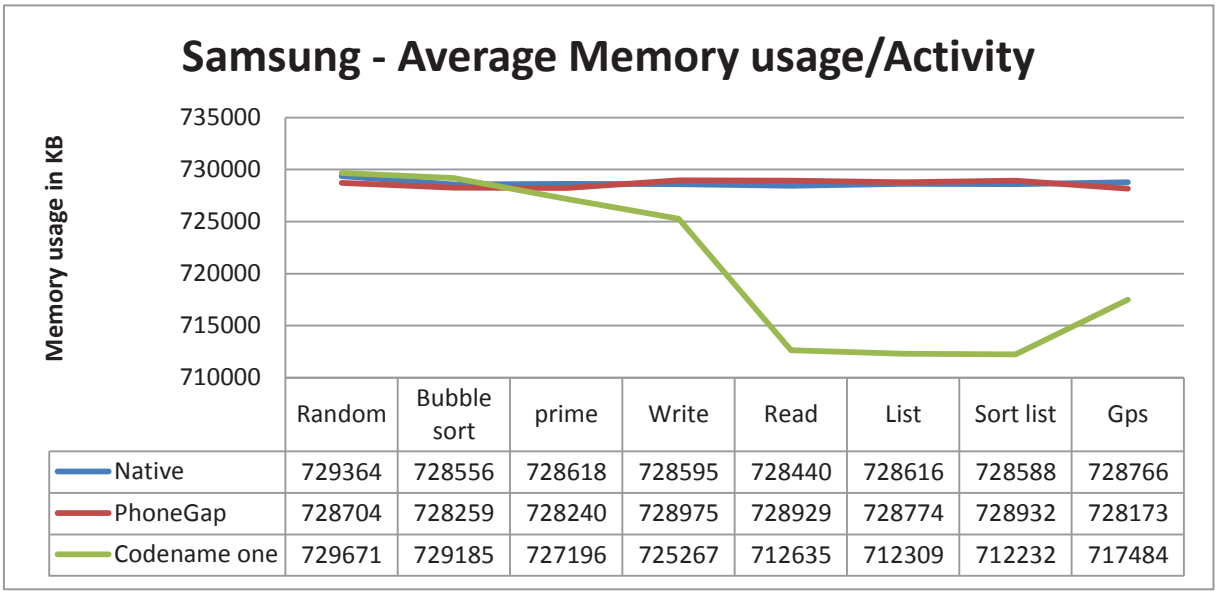
On this phone PhoneGap outperformed the other two. It has much lower execution time and energy consumption. But it used more memory. The Write to SQL phase caused PhoneGap to have less execution time and energy consumption.
Codename One didn't differ much from native's performance. This is expected because Codename One creates a native application. It had shorter execution time but more energy consumption.

It would be interesting to see what the outcome would have been if PhoneGap hadn't outperformed the other two in the Write to SQL phase.
All three applications had similar performances for the GPS phase because all they do is an API call.

**Findings:**

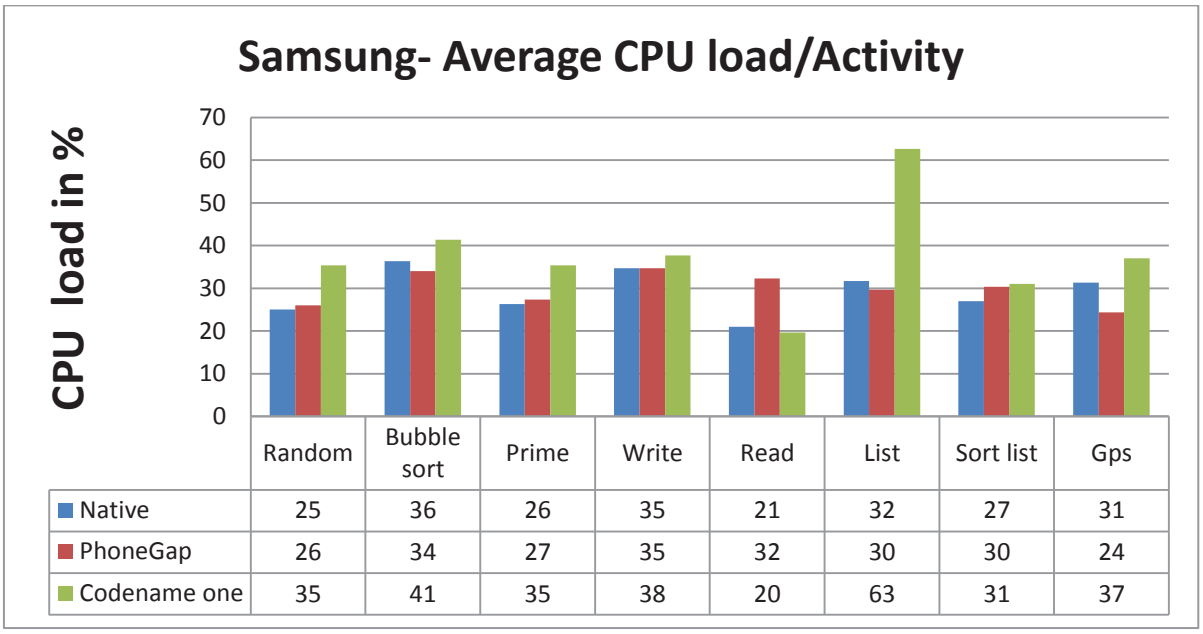
- PhoneGaps low energy consumption and execution time is a consequence of the high CPU usage in “write to SQL phase”.
- CPU usage has big impact on execution time and energy consumption.
- All three applications showed similar performance during the “GPS” phase because only an API call was made.
- Not only execution time impacts energy consumption, this is shown between native and Codename One, as native had longer execution time but less energy consumption.
- Codename One had very similar results as native. this is expected if Codename One is optimize.

### 4.2.2 Samsung Galaxy Trend plus



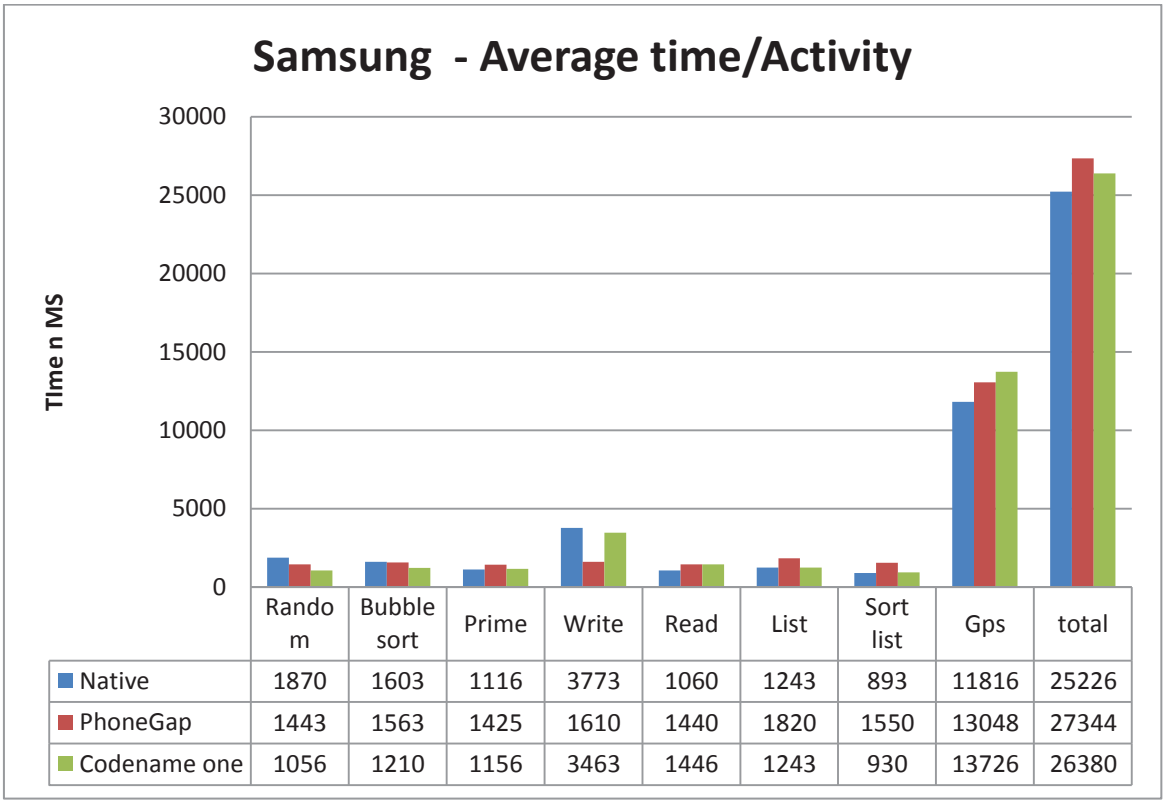

| | Random | Bubble sort | prime | Write | Read | List | Sort list | Gps |
|---|---|---|---|---|---|---|---|---|
| Native | 729364 | 728556 | 728618 | 728595 | 728440 | 728616 | 728588 | 728766 |
| PhoneGap | 728704 | 728259 | 728240 | 728975 | 728929 | 728774 | 728932 | 728173 |
| Codename one | 729671 | 729185 | 727196 | 725267 | 712635 | 712309 | 712232 | 717484 |

*Figure 4.9 – Samsung, Average memory usage/activity.*




| | Random | Bubble sort | Prime | Write | Read | List | Sort list | Gps |
|---|---|---|---|---|---|---|---|---|
| Native | 25 | 36 | 26 | 35 | 21 | 32 | 27 | 31 |
| PhoneGap | 26 | 34 | 27 | 35 | 32 | 30 | 30 | 24 |
| Codename one | 35 | 41 | 35 | 38 | 20 | 63 | 31 | 37 |

*Figure 4.10 – Samsung, Average CPU load/activity.*




| | Random | Bubble sort | Prime | Write | Read | List | Sort list | Gps | total |
|---|---|---|---|---|---|---|---|---|---|
| Native | 1870 | 1603 | 1116 | 3773 | 1060 | 1243 | 893 | 11816 | 25226 |
| PhoneGap | 1443 | 1563 | 1425 | 1610 | 1440 | 1820 | 1550 | 13048 | 27344 |
| Codename one | 1056 | 1210 | 1156 | 3463 | 1446 | 1243 | 930 | 13726 | 26380 |

*Figure 4.11 – Samsung, Average execution time/Activity.*

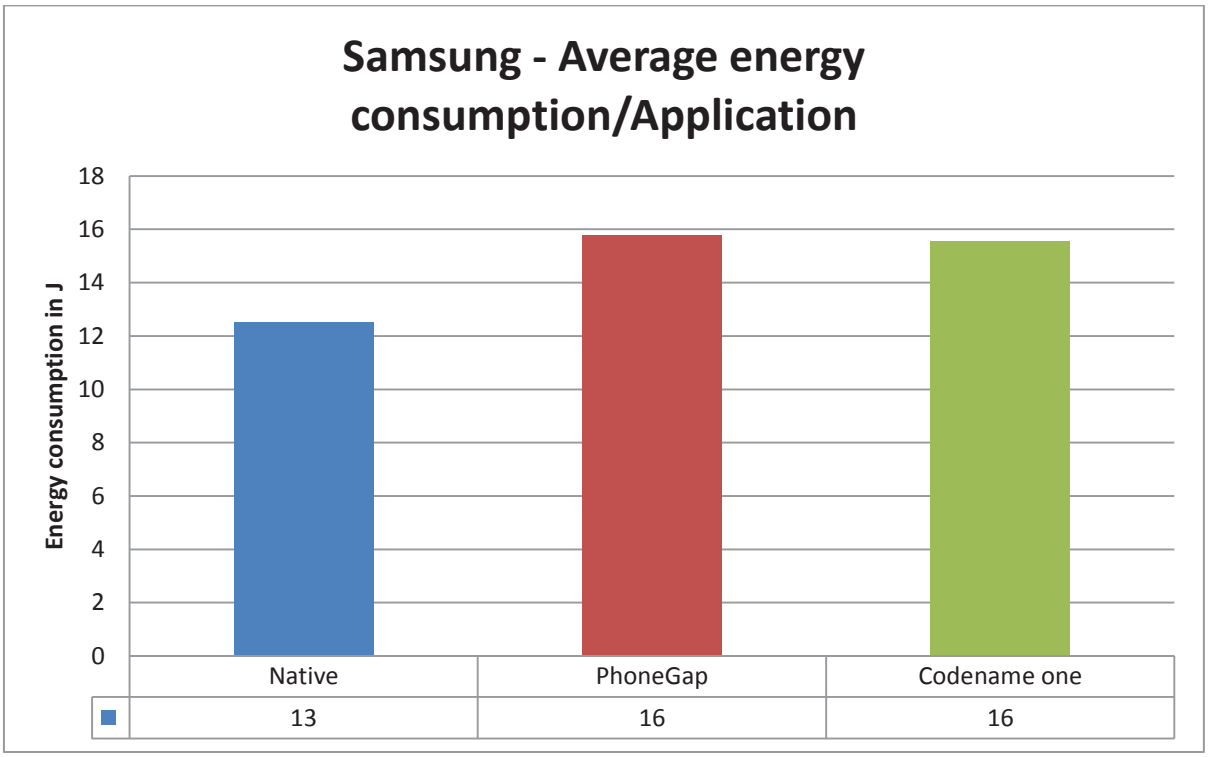



*Figure 4.12 – Samsung, Average energy consumption/Application.*

Native came in close third for memory usage, used least CPU, had shortest execution time and consumed least energy. Native performed best.
Codename One used much less memory, used most CPU, came second in execution time and had as much energy consumption as PhoneGap.
PhoneGap came second in memory usage, second in CPU usage, had longest execution time and equal energy consumption as Codename One.
The Samsung phone shows completely different result than Sony V. Here Codename One uses less memory than the others and PhoneGap as marginally better than native. This could be because of the garbage collector.

In the CPU, figure 4.10, chart we see a more even result than in the Sony V chart. There isn't much difference between the three except during create list phase where Codename One shoots up a bit. And the end result is that overall Codename One uses more CPU than the other two. Codename One seems to need more CPU to create a graphical list as fast as native. In the sort list phase the same graphical list is used and there it has similar CPU use as the other. Creating and initializing a list takes a lot more CPU than just filling it with ne elements.

Native had the shortest execution time of the three with Codename One as close second.

In the energy consumption chart, figure 4.12, native outperformed the other two. Native and Codename One seem to have the same ratio as on Sony V. But here PhoneGap didn't perform much better than the other two but instead is equal to Codename One.

Once again PhoneGap has shortest execution time during the Write to SQL phase. Since the difference isn't as big this time and PhoneGap having a little longer execution time in the other phases, PhoneGap gets longest total execution time. PhoneGap also came in a close second place for

the CPU usage. It has low CPU usage and long execution time. This leads to believe it uses less CPU and therefor gets longer execution time.
Native used least CPU and still had shortest execution time, this leads to native probably being optimized best.
Codename One had highest CPU usage while it had second shortest execution time.
Codename One sacrifices CPU for shorter execution time while PhoneGap saves CPU and therefor gets longer execution time. In the end this results in equal energy consumption for the two while native consumes least.

**Summary**

The results on the Samsung don't look anything like the results from Sony V.

On this phone native had the best performance, least execution time and energy consumption. Here it looks like native is more optimized because it achieved this even though it had the least CPU usage. The numbers are more evenly aligned for this phone, the amounts that were used are much closer each other between the phones. PhoneGap is still better at writing to SQLite.
Out of the CPTs I would say Codename One performed a little better than PhoneGap because it had shorter execution time but it had the same energy consumption as PhoneGap.

**Findings:**

- Very different performance results depending on what device is used.
- PhoneGap is best for writing to SQLite.
- Codename One used least memory.
- Codename One needed much more CPU for the "Create list phase".
- PhoneGap and Codename One consumed the same amount of energy.

### 4.2.3 Application size

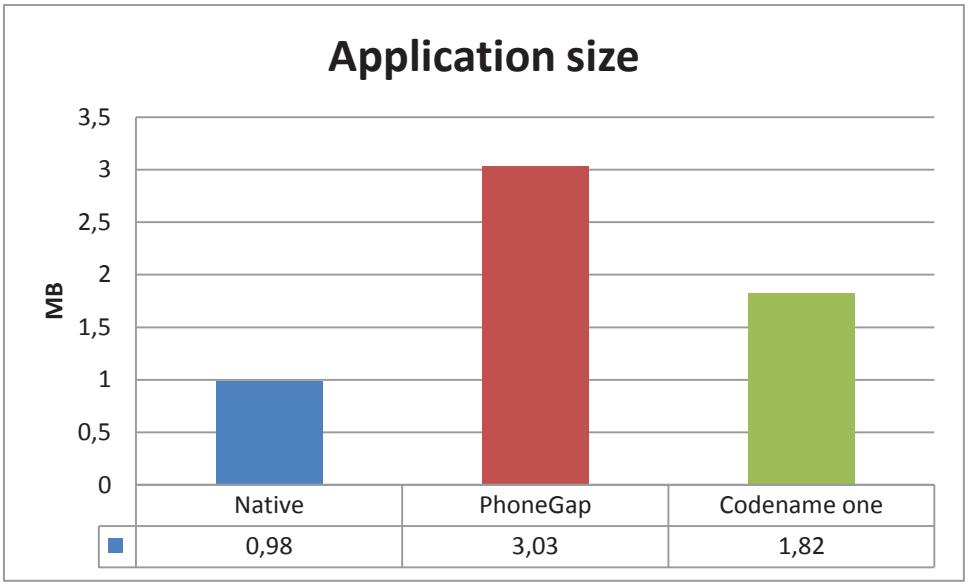



*Figure 4.13 – Application size.*

Here we can see that the PhoneGap application is bigger than the rest and native is smallest.

# 5 Analysis

Here I will do an analysis of the experiment result. In the charts in this section the numbers from both phones have been added together then that has been divided by two. This was done to get an average between the two phones.

## 5.1 Memory usage

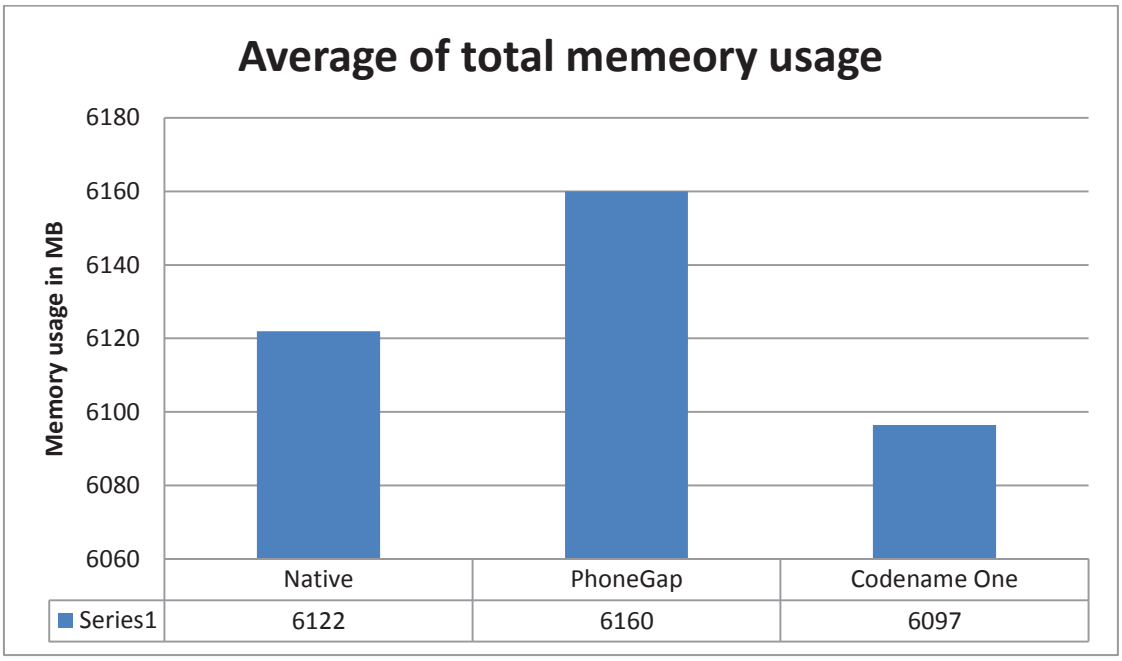



*Figure 5.1 – Average of total memory usage*

The difference is very small in memory usage. The charts show bigger difference because they are zoomed in, but if we look on the numbers we see that difference is very small.

The applications had very varied behavior on the two phones. On Sony V native had least memory usage while on Samsung Codename One had the least.
Native goes from having the least, figure 4.5 use to getting a shared last place with PhoneGap, figure 4.9. Codename One performed better than PhoneGap on both Samsung and Sony V.
Using a PhoneGap or Codename One doesn't have a big impact on memory usage although Codename one a little less memory usage.

**Related work**

In paper [17] it was concluded that Titanium (a cross-compiler like Codename One) uses more memory than PhoneGap. The result contradicts the claim by [21] that cross-compilers have better performance than other CPT approaches. In regards to memory usage my result agrees with the claim [21] makes. We don't know how [21] define performance since they only say that cross-compiler has better performance and their references don't make the claim either.

## 5.2 CPU load

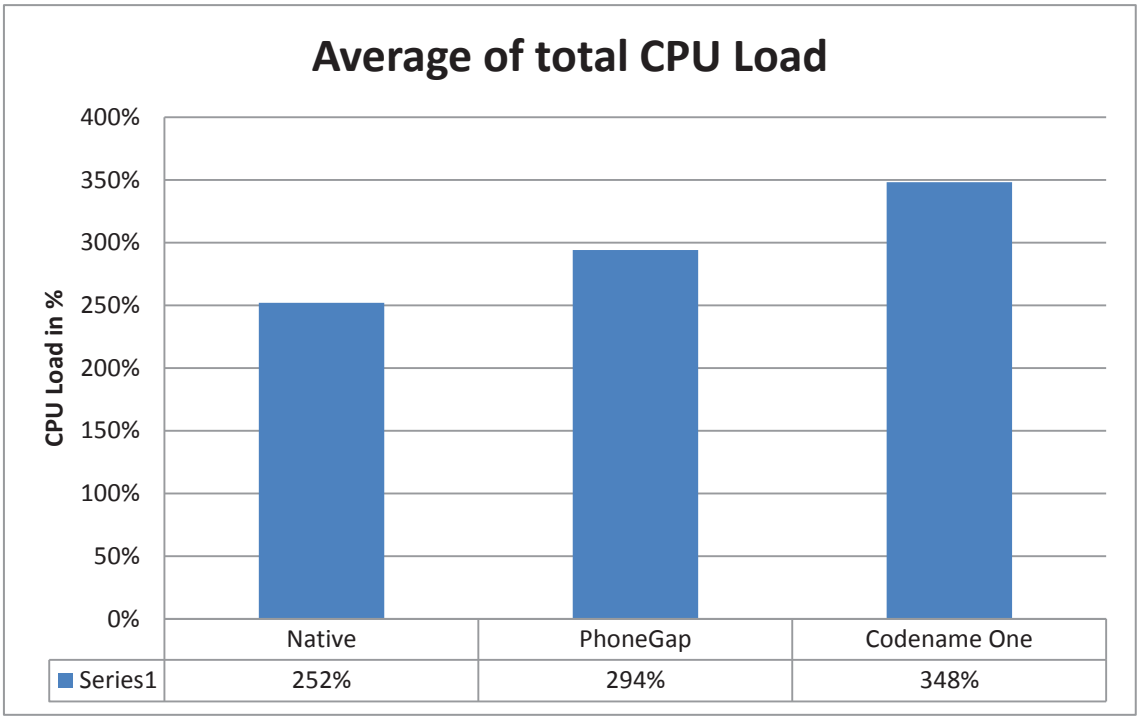



*Figure 5.2 – Average of total CPU Load.*

PhoneGap performed best of the two CPTs but native used less than both. Codename One used most. PhoneGap used less than Codename One. The Samsung phone, figure 4.10, has a much faster CPU than the Sony V, figure 4.6, this explains why the applications were faster on the Samsung.

When measuring CPU usage it is very important to also measure execution time so you can put those in comparison to each other. Otherwise the result can be interpreted in different ways.

## 5.3 Execution time

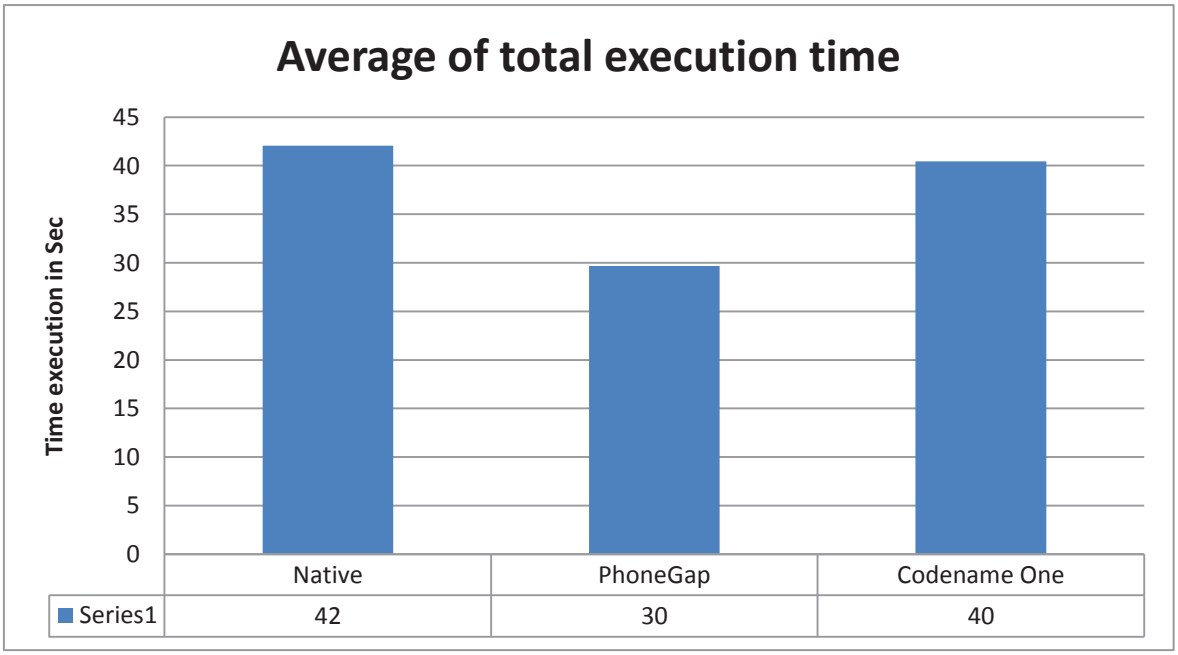



*Figure 5.3 –Average of total execution time.*

I got very different execution time result depending on the phone. PhoneGap went from shortest, figure 4.7, to longest, figure 4.11, by a small margin but we can see that on average PhoneGap had the shortest execution time. This mostly has to do with the time difference in the Write to SQL phase. PhoneGap has less CPU usage and shorter execution time than Codename One, PhoneGap gets more done with the CPU.

## 5.4 Energy consumption

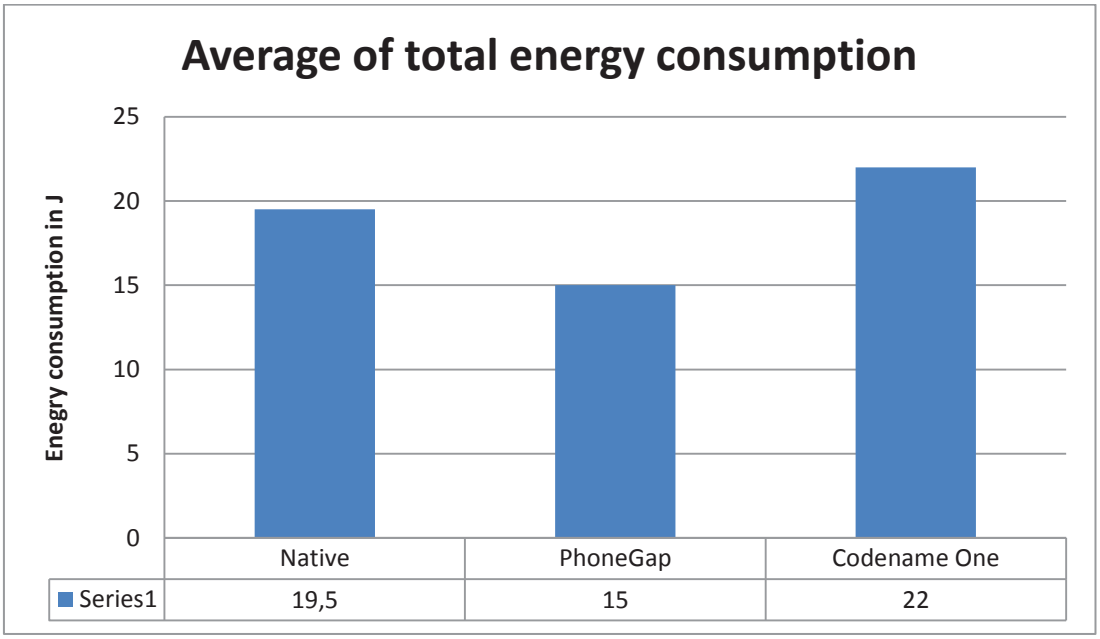



*Figure 5.4 – Average of total energy consumption.*

Energy consumption is affected by execution time and execution time is affected by the CPU usage. The longer execution time the more energy is consumed.
PhoneGap had least energy consumption. The small energy consumption by PhoneGap correlates with the high CPU use it had when writing to SQL which made it have much shorter execution time and thus the energy consumption is less than the other two.
If we look instead on the Samsung results, where the CPU load during write to SQL was more even between the three applications, the result is more even and there we can see that native had least energy consumption.

**Related work**
This result both does and doesn't show the same as [19]. They conclude that native has the least energy consumption and that PhoneGap is worse than their cross-compiler, Titanium. My result shows that PhoneGap is better, figure 4.8 and 5.4, than or as good as, figure 4.12, a cross-compiler.

After doing my experiment I realized it is important to also look at the execution time when measuring energy consumption as longer execution time equals more energy consumption. Only showing energy consumption and not the execution time can show a skewed result.

## 5.5 Application size

The application size was smallest for native and Codename One was smaller than PhoneGap. The reason for PhoneGap being bigger in application size might be because it needs many JavaScript libraries for doing native API calls.

**Relate work**
This result shows the same as [20] does, native has smaller application size than PhoneGap.

## 5.6 Validity threats

**Different devices**
Because I used two phones instead of one the result show that depending on which phone the application is run on the performance differ. This threat happened but at the same time it was a good threat. It shows that performance isn't black and white.

**Measurements**
Since energy consumption measurements are just a rough estimate they can't be trusted blindly. But we can say that PhoneGap and Codename One had very similar energy consumption on one phone and on the other PhoneGap used less.

**Android prioritizing**
This can be the reason for why PhoneGap had much higher CPU usage during the "Write to SQL phase" but I don't think this is the reason. Because it was the same every time I ran the tests, also for during the test runs, and it is very unlikely that some other process and needed the CPU every time I ran native and Codename One and not when I ran PhoneGap.

**Different OSs**
I don't believe this is what caused the big difference in performance between the devices, it was more likely the hardware and the fact that the Sony V was used daily while the Samsung was new.

**Java garbage collector**
It is very possible that this affected the memory usage results but I can't control that.

## 5.7 Summary

For **RQ2** PhoneGap performed best it had shortest execution time, least energy consumption, used less CPU than Codename One but used most memory. The results were very different between the two phones. It could be a result of the Sony V being older and well used while the Samsung is new and has never been used before. It can also be because the Sony V runs Android 4.3 and Samsung runs 4.2. Although I found no changes in the change log [28] that I believe would cause the difference.

The big performance difference for the Write to SQL phase, figure 4.6, caused PhoneGap to have much better execution time and energy consumption on the Sony V. If that is ignored native had best performance overall but the difference between the applications wasn't very big. PhoneGap used less CPU and Codename One used less memory. The results for energy consumption varied. If we look at the results from the Samsung, where the results were more stable, the energy consumption was equal between the two and on Sony V PhoneGap used a lot less than Codename One. Codename One had a smaller application size than PhoneGap.

On the Samsung phone all three applications stay relatively the same on CPU, figure 4.10, usage during the Write to SQL phase and all the other phases. This leads to them having about the same execution time and energy consumption, but this isn't always the case, as we can see with native on

the Samsung where native used least CPU and still had the shortest execution time and the least energy consumption. Native gets the most done with less CPU while Codename One has to use more CPU to get done, and the increased CPU usage consumes more energy. PhoneGap has less CPU use and longer execution time than Codename One, but the same energy consumption.

For **RQ3** [21] Makes the claim that cross-compliers have better performance than other CPT methods. They didn't have any experiments or results to prove their theory, their references don't make that claim nor do they have any experiments to prove it. [21] doesn't tell how they define performance. Let us assume they include the things I have been testing and see how my result relates to their claim.

- **PhoneGap:** shortest execution time, least energy consumption and least CPU usage.
- **Codename One:** Smallest application size and least memory usage.

Than they are wrong in their statement as PhoneGap, the hybrid, performed best of the two.
Based on related work there isn't any basis for [21] claim that cross-compilers have better performance and my result doesn't back it up either.
Perhaps the performance of each CPT depends on the implementation of the tool. One CPT approach might have a better foundation for better performance. PhoneGap might be much more optimized and therefore not losing much performance. According to [29] PhoneGap is much optimized and should have close to native performance.

The related work I have found is too lacking for me to be able to compare it to my result.

When doing the experiment I realized that it was important to measure the execution time because if you don't it can display a skewed result. E.g. Low execution time is sought but for that we need high CPU use or a much optimized CPU which can get more done. If we only show CPU use we don't know if it is good that the CPU use is low or high. We want low CPU use with low execution time but high CPU use is better if it means that we get lower execution time. The same goes for energy consumption, it is normal for execution time to be higher if it's longer execution time but if we don't show execution time we don't know it that is the case.

| **Performance** | **Memory usage** | **CPU usage** | **Execution time** | **Energy consumption** | **Application size** |
|---|---|---|---|---|---|
| **Cross-platform tool** | Codename One | PhoneGap | PhoneGap | PhoneGap | Codename One |

*Figure 5.5- Summary of the Performance results.*

# 6 Conclusion and future work

## 6.1 Conclusion

This thesis has research which CPT of PhoneGap and Codename one has best performance, native android was also measured to use as a reference. Execution time, energy consumption, CPU load, memory usage and application size was measured as performance. Execution time and energy consumption are the most important aspect, they will be used to judge which CPT was best. The other aspects are used to explain the result and measured because it is interesting.

It is important to measure execution time to hinder showing a skewed performance result when measuring CPU usage and energy consumption. Those twos result need execution time to show the whole picture, they can't be judged correctly without execution time.

**RQ1: What is the difference between different cross-platform approaches?**
There are four main approaches, hybrid, interpreter, web and cross-compiler, each one has its disadvantages and advantages and should be used depending on what kind of application will be created.

**RQ2: Which CPT had best performance, PhoneGap or Codename One.**
There isn't a very big performance difference between the two CPTs but PhoneGap was a best. PhoneGap had shortest execution time, least energy consumption and least CPU usage while Codename One had smallest application size and least memory usage. Which device is used has a big impact on the result. PhoneGap is the best for applications working with SQLite. For applications using GPS it doesn't matter as much which CPT you use.

**RQ3: Is my result in line with cross-compilers having better performance than other approaches..**
Nor my experiment or the related work I found supports for the statement that cross-compilers have better performance than other approaches.

The related work on performance I found was lacking in substance, either they only measured one resource or they only used a "hello world" for the application. Neither did they state which CPT was the best.

## 6.2 Future work

Include a more graphical test, using pictures or just more graphical elements.

Another experiment where the tests are done with two hybrid CPTS and two Cross-Compiler CPTS would be interesting. Then we could see if the results are consistent between hybrid and Cross-Compiler or if these results only apply to PhoneGap and Codename One.

To improve this paper the experiment could be done on more devices to establish the consistency in the results and to see if the result from the devices used in this paper were an anomaly.
Since this experiment only used Android phones it would be good to see this experiment on other OS's as well to see if the results are consistent through different OS's.

# References

[1] Rodrigues Tonini, A., Matthis Fischer, L., Balzano de Mattos, J.C., Brisolara de Brisolara, L., “Analysis and Evaluation of the Android Best Practices Impact on the Efficiency of Mobile Applications”, III Brazilian Symposium on Computing Systems Engineering (SBESC 2013), p 157-8, 2013.

[2] Android developer, “Performance tips”, Available: http://developer.android.com/training/articles/perf-tips.html .

[3] Hung-Ching Chang , Agrawal, A.R., Cameron, K.W. , “Energy-aware computing for Android platforms”,2011 International Conference on Energy Aware Computing (ICEAC 2011), p 1-4, 2011.

[4] Ding Li, Shuai Hao, Jiaping Gui, Halfond, W.G.J., “An Empirical Study of the Energy Consumption of Android Applications”, 30th International Conference on Software Maintenance and Evolution (ICSME 2014), p 121-30, 2014.

[5] D. Li and W. G. Halfond, “An investigation into energy-saving programming practices for android smartphone app development.” in Proceedings of the 3rd International Workshop on Green and Sustainable Software (GREENS), 2014.

[6] Yepang Liu, Chang Xu, Shing-Chi Cheung, “Diagnosing Energy
Efficiency and Performance for Mobile Internetware Applications”, IEEE Software, v 32, n 1, p 67-75, Jan.-Feb. 2015al invocations gives lees energy usage.

[7] Cheng-Min, Jyh-Horng Lin, Chyi-Ren Dow, Chang-Ming Wen, “Benchmark Dalvik and Native Code for Android System”, of the 2011 2nd International Conference on Innovations in Bio-Inspired Computing and Applications (IBICA 2011), p 320-3, 2011.

[8] Jae Kyu Lee, Jong Yeol Lee, “Android programming techniques for improving performance”, Proceedings of the 2011 3rd International Conference on Awareness Science and Technology (iCAST), p 386-9, 2011.

[10] Andreaz Lewerentz, Jonathan Lindvall , “Performance and Energy Optimization for the Android Platform”, pp. 28. COM/School of Computing, 2012.

[11] Mesfin, G. , Ghinea, G., Midekso, D., Grønli, T.-M. , “Evaluating Usability of Cross-Platform Smartphone Applications”, Mobile Web Information Systems. 11th International Conference (MobiWIS 2014). Proceedings: LNCS 8640, p 248-60, 2014.

[12] PhoneGap, Available: http://phonegap.com/.

[13] Humayoun, S.R, Ehrhart, S., Ebert, A, “Developing mobile apps using cross-platform frameworks: a case study”, Human-Computer Interaction. Human-Centred Design Approaches, Methods, Tools, and Environments. 15th International Conference, HCI International 2013. Proceedings. LNCS 8004, p 371-80, 2013.

[14] Li Tian, Huaichang Du, Long Tang, Ye Xu, “The discussion of cross-platform mobile application based on Phonegap”, 2013 IEEE 4th International Conference on Software Engineering and Service Science (ICSESS), p 652-5, 2013.

[15] Ribeiro, A., da Silva, A.R., “Survey on cross-platforms and languages for mobile Apps “,2012 Eighth International Conference on the Quality of Information and Communications Technology (QUATIC 2012), p 255-60, 2012.

[16] Palmieri, M, Singh, I., Cicchetti, A., “Comparison of cross-platform mobile development tools”, 2012 16th International Conference on Intelligence in Next Generation Networks (ICIN 2012): Realising the Power of the Network, p 179-86, 2012.

[17] Dalmasso, I., Datta, S.K., Bonnet, C., Nikaein, N., “ Survey, comparison and evaluation of cross platform mobile application development tools”, 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), p 323-8, July 2013.

[18] Appcelerator, Titanium, Available: http://www.appcelerator.com/.

[19] Ciman, Matteo, and Ombretta Gaggi. ”Evaluating impact of cross-platform frameworks in energy consumption of mobile applications.".

[20] Ohrt, J., Turau, V., “Cross-platform Development Tools for Smartphone Applications”, Computer, v 45, n 9, p 72-9, Sept. 2012.

[21] Raj, R, Tolety, S.B., “A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach”, 2012 Annual IEEE India Conference (INDICON 2012), p 625-9, 2012.

[22] Charkaoui, S, Adraoui, Z., Benlahmar, E.H., “Cross-platform mobile development approaches”, 2014 Third IEEE International Colloquium in Information Science and Technology (CIST). Proceedings, p 188-91, 2014.

[23] Open source Java iOS tools compared, 2013, Available: http://www.javaworld.com/article/2078740/mobile-java/java-ios-developer-open-source-java-ios-tools-compared.html .

[24] Codename One, Available: http://www.codenameone.com/index.html.

[25] Is Appcelerator Titanium Mobile really a Cross-Compiler? , Available: http://developer.appcelerator.com/question/45001/is-appcelerator-titanium-mobile-really-a-cross-compiler

[26] PowerTutor, http://ziyang.eecs.umich.edu/projects/powertutor/

[27] Trepn Profiler, https://developer.qualcomm.com/mobile-development/increase-app-performance/trepn-profiler

[28] Android Jelly Bean, http://developer.android.com/about/versions/jelly-bean.html

[29] PhoneGap Experts, is PhoneGap slow or is it poor development? , http://phonegapexperts.com/is-phonegap-slow-or-it-is-poor-development/

[30]”Smartphone OS Market Share, Q4 2014”, http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[31] “Constraints of designing for mobile devices”, http://docs.blackberry.com/en/developers/deliverables/17965/Constraints_of_designing_for_mobile_devices_1017065_11.jsp

[32] “Frequently Asked Questions”, http://source.android.com/source/faqs.html#why-is-google-in-charge-of-android

[33] “Fast Touch Event Handling: Eliminate Click Delay”, PhoneGap-tips.com, http://phonegap-tips.com/articles/fast-touch-event-handling-eliminate-click-delay.html

[34] “How do I – improve application performance or tack down performance issues”, http://www.codenameone.com/how-do-i---improve-application-performance-or-track-down-performance-issues.html

[35] “developing better PhoneGap apps”, http://floatlearning.com/2011/03/developing-better-phonegap-apps/

[36]Shengkui Gao, “A Survey of Latest Performance, Development and Measurement Issues of Smart Phones Design”, 2011.

[37]Platform versions, Available 2015-06-04, http://developer.android.com/about/dashboards/index.html

[37] Alexander Bakker, "Comparing Energy Profilers for Android." (2014).

[38] ”Random string generator”, https://www.random.org/strings/

## Appendix A – Bubble sort

Android source code

```
1. int [] arr= new int[]{n};
2. int n = arr.length;
3. int temp = 0;
4. for(int i=0; i < n; i++){
5.         for(int j=1; j < (n-i); j++){
6.                 if(arr[j-1] > arr[j]){
7.                 //swap the elements!
8.                 temp = arr[j-1];
9.                 arr[j-1] = arr[j];
10.                arr[j] = temp;
11.                }
12.        }
13. }
```