Letterkenny Institute of Technology

A thesis submitted in partial fulfilment of

the requirements for the Master of Science in Computing in

Enterprise Applications Development Letterkenny Institute of Technology

Comparing the Performance of Enterprise Applications on Limited Operating Systems

Author:

Linda McGrath B.Sc.

Supervisor:

Mrs Ruth Lennon M.Sc., B.Sc.

Submitted to the Higher Education and Training Awards Council (HETAC)

September 2012

Declaration

I hereby certify that the material, which I now submit for assessment on the programmes of study leading to the award of Master of Science in Computing in Enterprise Applications Development, is entirely my own work and has not been taken from the work of others except to the extent that such work has been cited and acknowledged within the text of my own work. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of Candidate ____________________ Date _____________

I hereby certify that all the unreferenced work described in this thesis and submitted for the award of Master of Science in Computing in Enterprise Applications Development, is entirely the work of Linda McGrath. No portion of the work contained in this thesis has been submitted in support of an application for another degree or qualification to this or any other institution.

Signature of Supervisor ____________________ Date _____________

Acknowledgements

This thesis would not have been possible without the support of many people. The author is especially grateful and would like to sincerely thank her supervisor and course creator Mrs Ruth Lennon for her continued guidance, support, and understanding. The author would also like to extend her gratitude to the Head of Computing Department Mr Thomas Dowling, and all the staff in Letterkenny Institute of Technology for all their help over the years. Finally the author would like to express her thanks to her family and friends for their support throughout writing this thesis.

Abstract

These days a Smartphone is just as important as a computer. It is essential that users select a device that has the best performance possible. This thesis used simulation to compare the performance of an application on two of the most popular mobile operating systems, Android and Windows Phone 7. An application was developed using the Android SDK/Java and Silverlight's C#/XAML respectively. This application tested the performance of threads, file I/O, database loads, buffering, and caching on both operating systems. Both simulation devices were set up with the exact same specifications. The performance was measured using response times, CPU, and memory usage. This work shows that there were major differences in the performance results. With the response times Windows Phone 7 out performed Android in every scenario except buffering. Windows Phone 7 also used the least memory in every situation except loading when a web page without caching. However when it came to CPU usage Android used half of what Windows Phone 7 did.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Purpose

This document is the final dissertation for M.Sc in Enterprise Application Development. It gives the Software Requirements Specification for a thesis on *Comparing the Performance of Enterprise Applications on Limited Operating Systems*.

1.2 Background

Throughout the last number of years the popularity of Smartphones and devices, such as tablets, has soared. With such a wide variety of operating systems available for these devices, it is important to know how each operating system performs, especially as these devices are becoming an integral part of business as well as leisure. Many limited operating systems, such as Android and Windows Phone 7, are not limited to one particular brand or type of device. The Android operating system can be found running on a Sony Ericsson device as well as a Samsung device. This makes it difficult to determine the performance of the operating system itself. In order to get a clear idea of the performance of these operating systems, the PerformanceApp software artefact was developed.

The PerformanceApp is a piece of software that was created to run on an Android and Windows Phone 7 device or emulator. For the purpose of this thesis, the PerformanceApp application was run through the Android and Windows Phone 7 emulators. The emulators were set up with matching specifications. The PerformanceApp application recorded several measurements: the response times taken to complete specific tasks, the CPU usage and the memory usage. The measurements were selected to provide a broad view of the operating system's performance, and were used to help determine which operating system performed best in these specific areas. These measurement tasks are discussed in detail in Chapter 3. This thesis contains a literature review which discusses the background of mobile devices, the operating systems they run, performance metrics, and existing research on the performance of limited operating systems. The thesis also discusses the requirements for the PerformanceApp software, how the software was tested, the results obtained and the conclusions drawn based on the results.

1.3 Outline of Report

Chapter 2 reviews the background of Smartphones, operating systems, and performance.

Chapter 3 examines design diagrams for the software including UML diagrams.

Chapter 4 tests the performance application.

Chapter 5 analyses the results of the testing and discusses its findings.

Chapter 6 closes the thesis and suggests further research.

2 Literature Review

2.1 Introduction

This chapter discusses the technologies and methodologies that were used in the development of this thesis. Areas covered include: Smartphone devices and their operating systems, the various areas of a systems performance, and existing research on the performance of limited operating systems.

2.2 Limited Operating Systems and Mobile Devices

A survey taken in 2008 suggested that of the 154.4 million people in employment in the United States, 72.2% used a mobile device or Smartphone at some point to carry out their job on the move. This percentage is expected to rise to 75.5% by 2013, which will be approximately 119.7 million people using a Smartphone for their work. These figures show how vital Smartphones and mobile devices have become. Due to wide availability of Smartphones, the market has a higher expectation for better performance. The performance of Smartphones should be analysed to determine which device provides the best performance Riedy et al. (2011). Due to the demand for Smartphones, their applications must be reliable, have low power consumption, and should also have a high performance level.

There is a wide array of mobile operating systems available. For the scope of this thesis, two of these operating systems were selected: Android and Windows Phone 7. The selection of these two operating systems was based on their popularity and availability across a large array of devices. The operating system acts as an interface between the user and the hardware components of a device, whether it be for Android or Windows Phone 7. User preference is often based on the operating system rather than the device itself. Two Smartphones with the same hardware specifications running different operating systems can produce different performance results. This thesis ran matching applications on two device emulators with the same hardware specifications using the two different operating systems. The purpose of this was to determine which handled specific tasks best. Knowing which operating system performs specific tasks best will aid users in selecting their Smartphones.

For this thesis, the application that ran on the emulators was a piece of software called PerformanceApp. The software was developed for both the Android and Windows Phone 7 operating systems and tested on their respective device emulators. The PerformanceApp application was designed and developed purposely to test specific performance aspects of both operating systems that are commonly used. The PerformanceApp has the ability to:

- spawn threads
- read and write text to an internally stored file
- run loads against a database
- buffer and play a MP3 file from an on-line source
- load a web page with the option of enabling or disabling caching

Mobile applications tend to be developed for multiple operating systems Kim et al. (2009). These applications may be written in different programming languages but their functionality is the same. Windows Phone 7 applications are developed using Silverlight or XNA, and Android applications are developed using Java or, in some cases, C or C++. For this thesis the Android PerformanceApp was developed using the Android SDK/Java in Eclipse, while the Windows Phone 7 version of the PerformanceApp was developed using Silverlight's C#/XAML in Visual Studio 2010 Express for Windows Phone.

Java and Silverlight's C#/XAML were chosen for developing the two versions of the PerformanceApp software as they are the most popular languages for their respective application development. The two versions of the PerformanceApp software mirror each other in both design and functionality. The reason for this was so the performance of the two could be compared. It is important to note that ideally a version of the PerformanceApp software would have been developed for all the major operating systems and tested on each operating system's device. However for the scope of this thesis that was not possible, it does provide an area for further research.

2.2.1 Android

Developed by Open Handset Alliance as part of Google, the Android operating system is one of the main competitors against Apple's iPhone and BlackBerry. Unlike Apple's iOS and BlackBerry OS, Android is available on a wide array of devices regardless of the brand. The current version at the time of this thesis was Android version 4.0 Ice Cream Sandwich. Version 4.0 uses the Android API level 14, which provides a unified framework for developing applications for Android phones as well as Android tablets Android (2012c).

Android provides a SDK and emulator for developers to write their own applications and gives them the opportunity publish them on Android Market. These applications can then be downloaded for free or purchased. Android applications are generally written in Java, however Android NDK allows developers to use other languages such as C and C++. As previously stated, for the purpose of this thesis the Android SDK/Java was used as it is the most commonly used language for developing Android applications Android (2012b).

The PerformanceApp software used a database to run loads against the system to determine its performance level. For database storage, Android cannot use the full version of SQL, however it does have access to SQLite. SQLite is a light weight version of SQL which is perfectly suited for a mobile device. It is one of the most popular relational databases available for mobile devices. SQLite only requires a small amount of disk space and memory to run, and is a persistent database. The popularity coupled with its ability were the reasons SQLite was used for the load task for the Android version of the PerformanceApp software.

Android has been developed on a Linux kernel, however, due to architecture changes made by Google, Android is a completely different operating system, and as such porting Linux applications to Android can be difficult. A Linux kernel was chosen as the systems hardware abstraction layer because it has been 'tried and tested' over the years. As part of the Android Runtime, there is the Dalvik Virtual Machine. This uses .dex files to convert .class and .jar files at build time to run more efficiently on a device with limited processing power such as a Smartphone Android (2012a). Android applications are allocated a space in internal storage. These applications are limited and each application is 'sand-boxed' to its own allocated storage space. Applications can only access their own storage. This is for security purposes as it stops applications reading information from other applications. The allocated memory is attached to the application and once that application is uninstalled any data belonging to it is also destroyed.

2.2.2 Windows Phone 7

Windows Phone 7 is the successor to Windows Mobile, with the latter finishing off at version 6.5.5. Although Windows Phone 7 replaced Windows Mobile, the two platforms are not backward compatible Ziegler (2010). The current version at the time of this thesis was 7.5 Mango. Windows Phone 7 was selected as one of the two operating systems that were compared, as it has gained a rapid increase in popularity, as well as being available on a wide range of devices.

The Windows Phone 7 architecture contains the Hardware Foundation layer. This contains a set list of the minimum hardware requirements that a Smartphone running Windows Phone 7 must conform to. The specifications required by Microsoft ensure that all devices using the Windows Phone 7 operating system are of a high standard. This list includes: ARM7 CPU, a GPU that can use DirectX, a camera, and a multi-touch capacitive display. This is to create a sense of consistency across all devices running the operating system for both users and developers. It is also to ensure the operating system performs smoothly on all devices that it runs on. Another hardware feature to encourage consistency are the three buttons found on the front of each device. These buttons, back, start, and search, provide quick navigation throughout the phone. The drivers for the Windows Phone 7 kernel are pre-written by Microsoft which provides a uniform experience for developers Interoperability (2011).

Like Android, each application is isolated into its own 'sandbox' in the device's internal memory for security purposes. This memory space is known as isolated memory, which is attached to its respective application. If an application is uninstalled, its isolated memory is cleared. The Applications Framework, which is part of the Windows Phone 7 architecture, allows an application to be coded in both Silverlight and XNA. XNA is a framework primarily used for developing games. Silverlight is an event driven framework which makes it easy and fast to build rich UI's and both it and XNA use Microsoft's CLR Interoperability (2011). For the purpose and the nature of the software that was developed for this thesis, Silverlight using C#/XAML was used.

2.2.3 Conclusion

The following table provides a comparison of the Android and Windows Phone 7 operating systems.

	Android	**Windows Phone 7**
Version Used:	4.0 Ice-cream Sandwich	7.5 Mango
Development Language:	Android SDK/Java (OOP)	Silverlight's C#/XAML (OOP)
IDE:	Eclipse	Visual Studio 2010 Express for Windows Phone
Database:	SQLite	Linq-to-SQL
App storage location:	Internal (sandbox)	Internal(sandbox)
Architecture:	Built on a Linux based kernel (drivers)	Almost all drivers built by Microsoft
OS Availability:	Any device	Any device (must meet standards)

Table 2.1: Android vs Windows Phone 7

Smartphones and mobile devices are used everywhere for both business and leisure. The performance of an application on a device is vital. The different architectures of operating systems can result in certain areas of an application performing better on one operating system than on another. There are many mobile operating systems available, however Android and Windows Phone 7 were selected for the purpose of this thesis. Table 2.1 shows a comparison of both operating systems. As the table showed the two operating systems chosen have several features in common. Both operating systems are not tied to a particular brand of phone or device, they are among the most popular mobile operating systems available, and both are developed using an Object Oriented Programming language. They both use the same 'sandbox' method of storing and protecting applications.

The table also shows that both operating systems are capable of the same functionality despite using different methods. It would have been interesting to see which single form of database connectivity would perform best between the two devices. However as Linq-to-SQL is only available on Windows Phone 7 devices this comparison was not possible. SQLite and Linq-to-SQL were chosen as they are the two most optimised forms of database connection available for each respective operating system. The PerformanceApp software was designed and developed to test areas of performance on the two operating systems. While the two versions of the software were written in different programming languages, both versions perform the exact same functionality. The functionality of the PerformanceApp software is discussed in detail in Chapter 3.

2.3 Performance

Performance was the key area of this thesis. Performance Metrics, Smartphone Performance, Performance Analysis and Benchmarking are covered in this section.

Stantchev and Malek (2008) tell us that performance can be defined as:

> "the system output that represents the number of successfully served requests from a total of input requests during a period of time."

2.3.1 Performance Metrics

Performance metrics are measurements used to determine the performance of a system. There are many metrics that can be used to measure a systems performance. The following are some of the most common: CPU usage, disk I/O rate, response time, throughput, scalability, path length and load Parekh et al. (2006).

Chiew (2009) say performance is the measure of how fast a system request can be completed, however a systems performance also takes in how efficiently the system runs. Performance can also be classed as how fast the system starts up and how the system runs under varying workloads. Unfortunately what the developer may believe to be good system performance may be different to what a customer perceives as being good performance. Joines et al. (2003) state that it is important to evaluate a systems performance, and while doing so the following should be watched: Load, Throughput, Response Time, Path Length, Bottlenecks and Scalability. They also state that these six can be further broken down into two separate groups, measurement and optimisation.
The **Measurement** can be broken down into:

- Load: the measure of the amount of work, or data, that the system performs. For this thesis, the load was measured by gathering the CPU usage throughout the tests.

- Throughput: the amount of successful data requests the system performs over time. The throughput was not recorded for this thesis as the scale of the thesis did not allow for sufficient results.

- Response time: the time it takes between a request being given and the request being completed. Joines et al. (2003) say response time can be defined as:

  $$RT = T_2 - T_1.T_1 \tag{2.1}$$

  Where T1 is time measured when the request is sent and T2 is time measured after the request is complete. This equation was used throughout the thesis to determine the response time for each task performed.
  The response time was recorded for each task carried out for this thesis using this equation.

The **Optimization** can be broken down into:

- Path length: the number of steps needed to complete a request.

- Bottlenecks: when a system receives more requests than it can handle and the data flow comes to a stand-still. Parekh et al. (2006) says putting an application through trails of various workloads allows you to identify bottlenecks.

- Scalability: this is adding resources, usually hardware, to the system to improve its performance. However, simply adding resources can change the dynamics of an application resulting in the application requiring tuning Joines et al. (2003). There are several ways a system can be up-scaled, the most common are:
  - Vertical Scaling: also known as scaling up, this usually involves adding hardware components like memory or processors to a single machine to improve its performance.
  - Horizontal Scaling: can also be called scaling out. Horizontal scaling is when more machines are added to a system to improve performance, e.g. going from one web server to two or three.

  - Linear Scaling: this is almost like a blend of vertical and horizontal scaling where a systems resources double. Linear scaling rarely happens in the real world Joines et al. (2003).

For the scale and nature of this thesis, scalability was not carried out. Further research could be carried out in this area in order to determine how various hardware configuration affect the performance of the operating systems.

2.3.2 Performance Analysis

Lilja (2000) says Performance Analysis has three fundamental techniques:

- Analytical Modelling: these are mathematical models that have a closed form solution. They are equations written to describe the performance behaviour of a system.

- Simulation: this is also a type of model, however they are tailored to simulate a particular system. Simulations are often used in place of analytical models when the systems is very complex. Results are not always accurate as no simulation can create a real life scenario.

- Measurements of Existing Systems: monitoring or using benchmarks from an existing system. These results tend to be a little more accurate but it is difficult to change any of the parameters.

For this thesis, Equation 2.1 was used to determine the response times for each of the tasks carried out throughout the PerformanceApp software. Ideally the PerformanceApp software would have been run and tested in a real life environment such as an Android or Windows Phone 7 device. However, due to a Smartphone or device being unavailable, it was unavoidable to use methods such as simulation. While this was undesirable, it was adequate for the scope of this thesis. Android and Windows

Phone 7 both provide an emulator for such circumstances. If both an Android and a Windows Phone 7 device were available, it would be an area for further research to measure the performance of both the real life devices against the emulators to see how they compare.

Kotsis (2006) gives five steps that were very useful for the performance analysis process. These steps provided a great starting point and guideline for the performance analysis stage of the PerformanceApp software. The steps followed were:

i) decide on the metrics to be used for testing the system

ii) set how much of a workload is to run on the system

iii) chose how and what tools are going to be used to test the system

iv) record all the results gathered

v) make adjustments and improve the system based on the information gathered

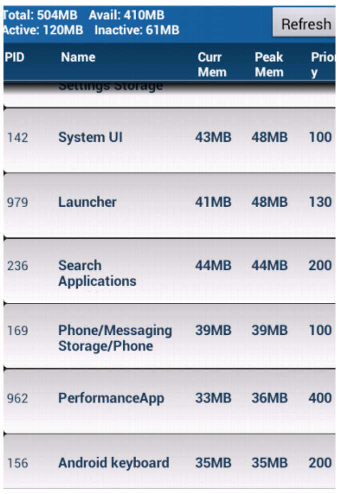
These steps were followed throughout the process of this thesis and the development of the Android and Windows Phone 7 versions of the PerformanceApp software. As the previous steps mentioned, the performance tools for the PerformanceApp software could not be chosen until the metrics were decided. The five steps described by Kotsis (2006) were invaluable to this thesis. Following the first step, the CPU usage, memory usage load, and response time were chosen as the metrics that were to be recorded. The seconds step varied from task to task. The workload set on each task had to be enough so that it would show any affect there was on the performance, but not so much that the entire application would come to a standstill. For step three there was a great variety of performance analysis tools available for both Java and Silverlight/C# and XAML. Choosing which one to use depended on what it was needed for and which

IDE was being used.

For this thesis a combination of coding and application tools were used to gather the results. For Android, code was used to obtain the CPU usage, and an application called *Memory Usage* was used to obtain the memory usage. For Windows Phone 7 the CPU usage was gathered by the Windows Phone Performance Analysis Tool. This tool also gathered the memory usage, however it showed this information in graphical form and did not give any numerical detail. To obtain the memory usage in detail, code was written to gather the results. Steps four and five used the results gathered from step three to determine which of the two operating systems performed specific task best.

In order to gather the data for the PerformanceApp software, tools were required to gather individual information. This information was used to compare the performance of the two versions of the PerformanceApp software which helped determine which performed specific tasks best. For both the Android and Windows Phone 7 versions of the application, a combination of coding and software tools were used to gather the CPU and memory usage. For Android, an application called *Memory Usage* was used to gather information on the amount of memory the application used, and code was used to gather the CPU usage information. For Windows Phone 7, code was written to obtain the memory usage, and the Windows Phone Performance Analysis Tool was used to determine the CPU usage.

The *Memory Usage* application was developed by TwistByte LLC, and shows the current memory usage, peak memory, priority, and process identification of each application or service running on the device. Fig 2.1 provides a snapshot of the readings achieved while measuring the file task on Android. This Fig shows the amount of memory that was available to the device and how much of the device's resources the PerformanceApp software required for the file task.


otal: 504MB Avail: 410MB
ctive: 120MB Inactive: 61MB
Refresh

PID	Name	Curr Mem	Peak Mem	Prio y
	Settings Storage			
142	System UI	43MB	48MB	100
979	Launcher	41MB	48MB	130
236	Search Applications	44MB	44MB	200
169	Phone/Messaging Storage/Phone	39MB	39MB	100
962	PerformanceApp	33MB	36MB	400
156	Android keyboard	35MB	35MB	200

Figure 2.1: *Memory Usage* Screen from File Task

Microsoft have provided the *Windows Phone Performance Analysis Tool* for profiling Windows Phone 7 applications. The Performance Analysis Tool runs in the background gathering data as functions in applications are used. The data is then presented in a graphical overview to show details such as Frame rate, CPU usage and memory usage. These results can be expanded and can give suggestions for improvements Microsoft (2012). Despite gathering memory usage information, the Performance Analysis Tool was not used for this purpose in this thesis. Code was written in order to obtain the memory usage of the PerformanceApp software. Fig 2.2 shows a snapshot of the Windows Phone Performance Analysis Tool in use for the thread task.

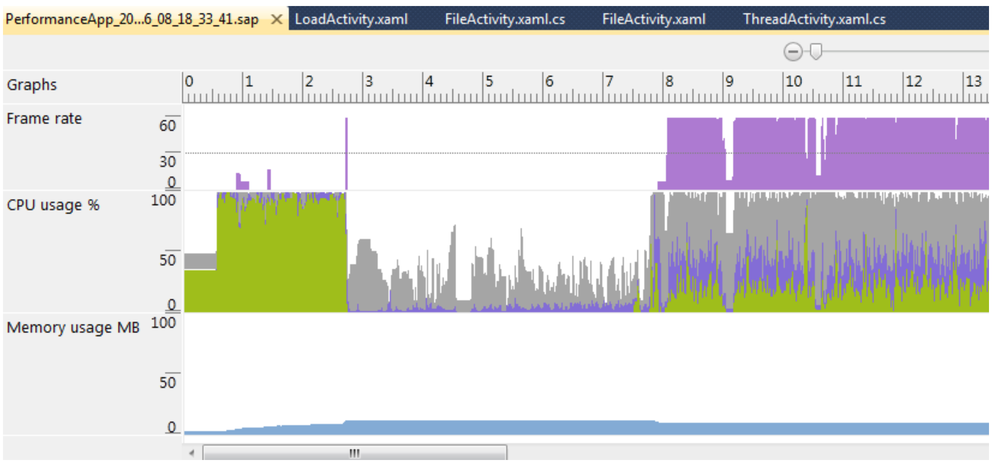


Figure 2.2: *Windows Phone Performance Analysis* Graph from Thread Task

2.3.3 Performance In Smartphones

Smartphones are infinitely more advanced than they were several years ago. Most now rival, if not match, the performance of some tablet computers. Smartphones have their limitations, having a limited power supply, processing power and memory. With these devices running off battery power it is important find ways of conserving energy. Lee and Lee (2011) tells us that writing more efficient code can enhance an application's performance as well as improving an application's energy consumption levels.

Lee and Lee (2011) have performed research in an area similar to this project. This thesis compared the performance of two matching applications on Android (written in Java) and Windows Phone 7 (written using Silverlight's C#/XAML). The idea of this was to compare which of the operating systems performed specific tasks best. Lee and Lee (2011) have done a project where they wrote two matching applications. Both applications were written for Android, one in Java and one in Native C using the Android NDK. They tested five areas: JNI delay, Integer parsing, Floating point, memory access and String processing. Section 2.2.1 explained how Android runs on the

Dalvik Virtual Machine. The Dalvik Virtual Machine has been specifically designed for devices with slow CPUs and restricted memory Lee and Lee (2011). For the purpose of measuring time in the project, Lee and Lee (2011) used a method similar to what was used in this project. Lee and Lee (2011) retrieved the current time in nano seconds. For this thesis the current time was recorded in milli seconds, this was the case for both versions of the PerformanceApp software. The time was recorded twice per task, at the beginning and at the end. Equation 2.1 shows the start time was then subtracted from the end time in order to determine the length of time taken for a task to be completed. The research carried out by Lee and Lee (2011) showed that in three of the five tests carried out, using Native C produced the best results having been faster with the Integer parsing, Floating point and Memory Access tests. Java won out with the other two tests which were JNI Delay and String processing. In fact they have suggested using Java for Android if the application is to involve many String processes.

Having performed research involving comparing the performance of an application running on Android, using the Dalvik Virtual Machine, Angstorm Virtual Machine with JIT(Just In Time Compiler) enabled and Angstorm Virtual Machine with JIT disabled, Kundu and Paul (2011) have suggested that in cases of Heap sorting and quick sorting, Android performs better using Angstorm Virtual Machine with JIT disabled. However, their research also shows that Angstorm Virtual Machine with JIT enabled provides better results than Android's Dalvik Virtual Machine. Kundu and Paul (2011) have shown that the Dalvik Virtual Machine gives 1.484 times better performance than the standard Angstorm Virtual Machine without JIT set-up, but in all of their tests, Dalvik Virtual Machine was shown to have a slower performance than Angstorm Virtual Machine with JIT enabled. Despite the slower performance Kundu and Paul (2011) state that Android's Dalvik Virtual Machine is more power efficient due being set up to optimise an applications energy consumption. Based on these findings, it would make an interesting extension to this thesis to run the Android version of the PerformanceApp software through different virtual machines and their configurations.

Kundu and Paul (2011) also discuss the potential for using parallel processors to further improve Android performance. They mention that having multiple processors can cut down the time required to run an application, and suggest having processors turned off when not in use can increase performance and an application's energy consumption. This is very fascinating and would be a great basis for further research for both Android and Windows Phone 7 performance, however for the scope of this thesis it was not included.

As previously mentioned in Section 2.2.1, SQLite was chosen as the relational database that was used for the load task in the Android version of the PerformanceApp software. Song et al. (2010) carried out research on the optimising the performance of SQLite on flash memory. Their research consisted of adding files to a SQLite database on an Android device. They used SQLite transactions and commits to ensure the data being used would not be lost in the event of a sudden loss of power. To optimise the performance of SQLite they suggested reducing the number or commits called when updating a database. They also showed that changing synchronous to off can also aid in SQLite optimisation. However, these methods can pose a security risk to the database. In the event of the device losing power, the database can become corrupt and unusable. The research Song et al. (2010) have put forward could make improvements to the response time of the load task for Android, but due to the security issues with these methods, they were not be used for this thesis.

2.3.4 Benchmarking

Performance Benchmarking is the measuring process used to compare systems. It can be used to compare the performance of hardware and/or software. Benchmark tests are written to test something specific and may not work well for testing other aspects of a system, even if they are related. Benchmarking is useful to see how an application runs on different systems Kotsis (2006). Benchmarking is a useful technique and provides an excellent bases for comparing software on various devices. As this thesis is concerned with comparing the performance of two operating systems on two different device emulators, benchmarking is not necessary or useful and therefore was not included.

2.3.5 Conclusion

For this thesis, the CPU usage, memory usage, and response times were measured for each task. Equation 2.1 was used to calculate the response times. This was the time taken to complete each individual task, such as, run threads or query a database. The response times, along with the CPU and memory usage allowed a direct comparison of the PerformanceApp software on both Android and Windows Phone 7 operating systems, and showed which operating system performed best at specific tasks. While it would have been preferable to have run the PerformanceApp software on an actual Android and Windows Phone 7 device, due to devices being unavailable, emulators were used in their place. These emulators were set up to have matching specifications to ensure results gained were accurate.

The *Memory Usage* application and *Windows Phone Performance Analysis Tool* were used in conjunction with code in order to retrieve memory and CPU usage information, and the response times for the various tasks. Lee and Lee (2011) used the same method in order to get the response time for tasks. Benchmarking can be a useful process for comparing systems, however the two versions of the PerformanceApp software were only run on their respective emulators and so was not used for this thesis.

2.4 Conclusion

The popularity of Smartphones and devices is continuing to grow every day. With this rise in popularity the performance of applications on these devices is becoming vital. Lee and Lee (2011), Kundu and Paul (2011) and Song et al. (2010) have put forward ideas which can benefit the performance of applications on Android, some of these ideas can also be applied to Windows Phone 7 performance. At present, there has been substantially more research carried out on Android performance. Windows Phone 7 is a new operating system and therefore is lacking research. Smartphones have been around for years but it is still a relatively new area and requires much further research.

The performance of the PerformanceApp software could not be measured until the metrics were decided. The metrics that were used for this project were CPU usage load, memory usage, and response time. These metrics were measured on both versions of the application using a combination of coding and software tools.

3 System Design

3.1 Introduction

This chapter includes the system, functional and non-functional requirements for the PerformanceApp software artefact that was developed for this thesis.

3.1.1 Project Scope

This thesis aimed at comparing performance aspects of limited operating systems. With a wide array of limited operating systems available today, and for the scope of this project, it was decided that two of the most popular operating systems would be chosen. Matching versions of the PerformanceApp software were designed and developed for each operating system. The operating systems selected were Android and Windows Phone 7.

Both versions of the PerformanceApp software were designed to have the same functionality with a similar GUI. As the Android version was written in Java and the Windows Phone 7 version was written in Silverlight's C#/XAML, there were slight differences in the implementation, but the overall look and feel was the same. Both versions of the PerformanceApp software ran tests on various aspects that can affect the performance of an application on a device. These included: threads, I/O rate, database loads, buffering a file, and caching a web page.

3.2 Functional Requirements

The following are the functional requirements for the PerformanceApp software artefact.

- The PerformanceApp software artefact will be developed to run on limited operating systems.
- Both the Android and Windows Phone 7 versions of the PerformanceApp software must have the same functionality.
- The PerformanceApp software will test various performance areas of limited operating systems.
  - Test areas(i.e. threads, file, load, buffer, cache) can be tested individually and in any order.
  - The response time for each task must be calculated using Equation 2.1.
  - The CPU usage and memory usage must be recorded.
  - The number the thread task counts to must be set by the user.
  - The user can set the number of threads to be run.
  - The text to be written to file must be set by the user.
  - The file cannot be read until content has been saved to it.
  - The number of records to be added to the database must be set by the user.
  - The database cannot be queried until the records have been added.
  - The MP3 must begin buffering before it can be played.
  - The server containing the web page must be running.
  - The page must be loaded with the option of enabling and disabling the cache.

3.3 Non Functional Requirements

The following are the the non functional requirements for the PerformanceApp software artefact.

- The PerformanceApp GUI should be consistent in both limited operating systems.
- The software should be robust.
- The software should be reliable.

3.4 UML Diagrams

This section contains the UML diagrams for this thesis. This includes the Use Case Diagram, Use Case Descriptions, Sequence Diagrams, System Flow Diagram, and the Class Diagrams.

3.4.1 Use Case/Sequence Diagrams and Descriptions

The Use Case diagram shows how the user interacts with the PerformanceApp software as a whole, and shows how each task can be performed independently of each other. The sequence diagrams provide a view of the user's interaction with the application for each of the separate functions, i.e. threads, file I/O, load, buffer, cache. The sequence diagrams have been split up as these functions do not have to be performed in any particular order.

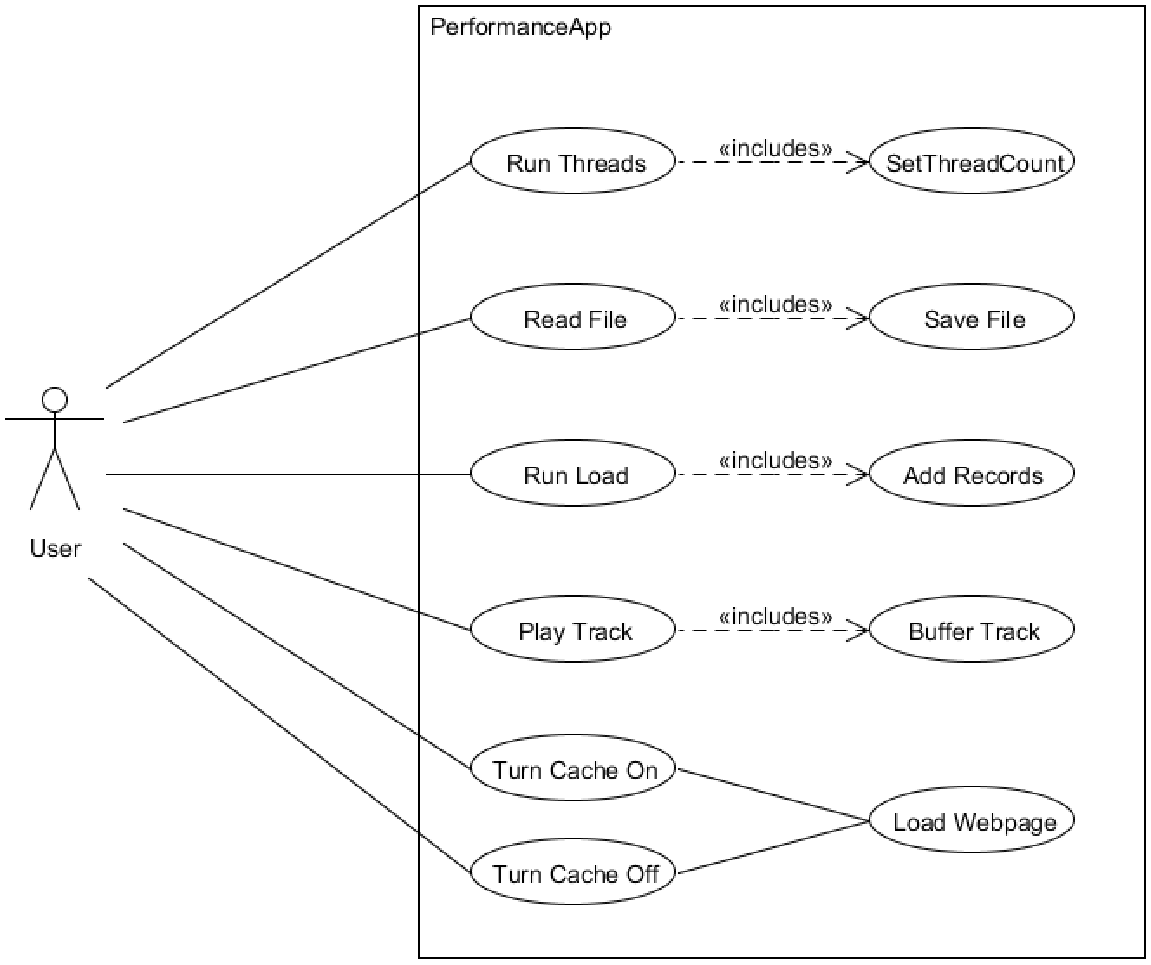


Figure 3.1: Use Case Diagram for the PerformanceApp software.

Run Threads Use Case

The purpose of the thread task was to retrieve the number of threads the user requested. The time taken for the application to count to the desired number of threads was recorded in order to determine the response time. Figure 3.2 gives a graphical representation of how the user interacted with the application to set and run the threads. Table 3.1 describes in detail what steps must be done in order to carry out the thread task.

Use Case	Run Threads
Objective	User sets the count number for the threads and runs them.
Precondition	The count number must be greater than zero.
Main Flow	1. The user enters the number desired in the text area. 2. Set count button used to set the thread count number. 3. Run threads button used to run the threads.
Alternative Flow	1. The user enters invalid data into the text area.
Post Condition	The thread runs until the count number is reached.

Table 3.1: Run Threads Use Case Description.

Figure 3.2 shows how the count number is requested from the user and set once it is input. The user then selects to run the threads. Allowing the user to set the count number made the application more dynamic as it allowed the work load to be varied. It should be noted that the Interface refers to the UI the user interacts with, which is written in XML for Android and XAML for Windows Phone 7. The term Device refers to the back end code which provides the functionality for the application, this is Java for Android and C# for Windows Phone 7. The Interface is connected to the Device providing a front end to the user.

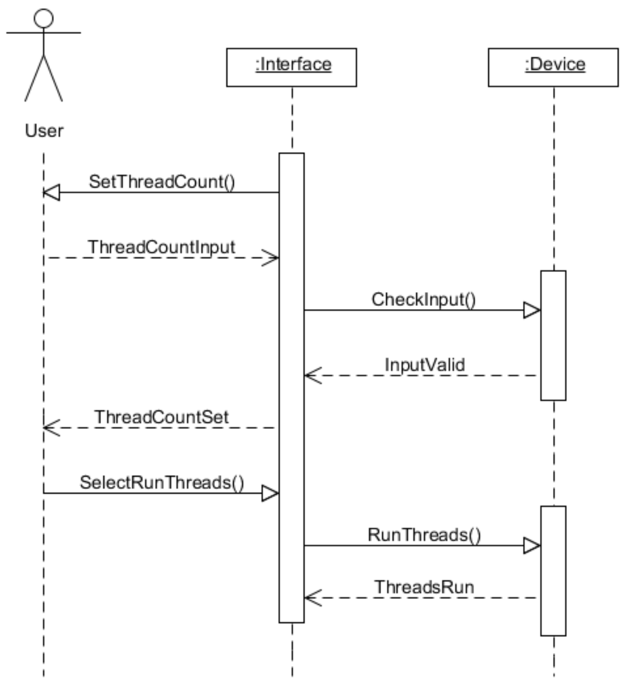


Figure 3.2: Thread Sequence Diagram

Read File Use Case

The purpose of the file task was to retrieve text from the user, and save that text to a file stored in the devices internal memory. The content of the file was then read back from internal memory and displayed on the screen. The time taken for the application to read and write to the file was recorded in order to determine the response time. The purpose of this task was to determine the performance of each operating system's disk I/O rate. Figure 3.3 gives a graphical representation of how the user interacted with the application to set the text, save it to the file, and read that file's content back from internal memory. Table 3.2 describes in detail what steps must be done in order to carry out the file task.

Use Case	Read File
Objective	User sets the text content for the file, saves it and reads it back.
Precondition	There must be text written in the text area.
Main Flow	1. The user enters the desired text in the text area. 2. The user presses the save button. 3. The text is saved to a file in internal memory. 4. The user presses the read button. 5. The file contents are viewed on screen.
Alternative Flow	1. The user enters no text in the text area.
Post Condition	A file is written to and read back.

Table 3.2: Read File Use Case Description.

Figure 3.3 shows how text for the file content is requested from the user and set once it is input. It then shows how the user can select to read the file content from internal memory and display it on the screen.

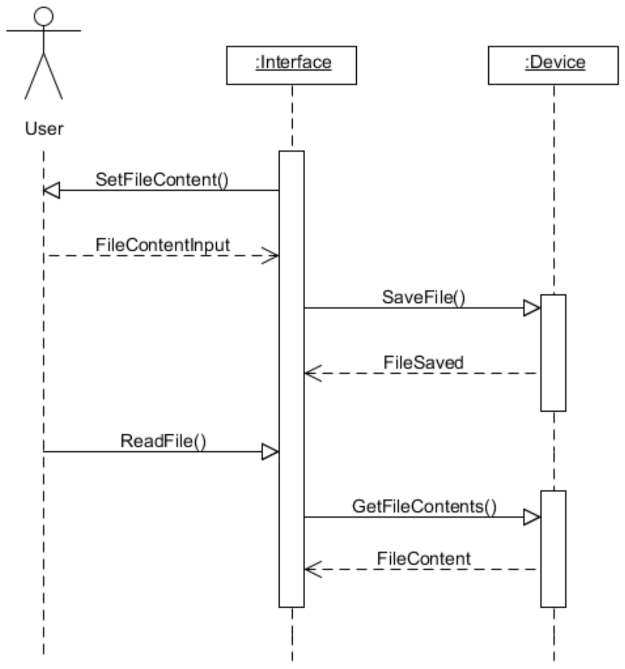


Figure 3.3: File Sequence Diagram

Run Load Use Case

The purpose of the load task was to retrieve the number of records to be added to the database from the user. After the number of records are added, the user runs a query against the database to retrieve all the records and displays them on screen. The time taken for the application to add and query the records was recorded in order to determine the response time. Figure 3.4 gives a graphical representation of how the user interacted with the application to set the number of records, add them to the database, and query the database to retrieve all the records. Table 3.3 describes in detail what steps must be done in order to carry out the load task.

Use Case	Run Load
Objective	The user runs a load on the database.
Precondition	There must be records in the database.
Main Flow	1. The user sets the number of records to add to the DB. 2. The set number of records are added to the DB. 3. The user requests the run load. 4. All records are called from database.
Alternative Flow	1. The records are not added to the database.
Post Condition	The database has been written to and queried.

Table 3.3: Run Load Use Case Description.

Figure 3.4 shows the users interaction with the application in order for the database to be created. The number of records to add to the database was then requested from the user. That number of records were then added to the database before the user can query the database to retrieve all the records.

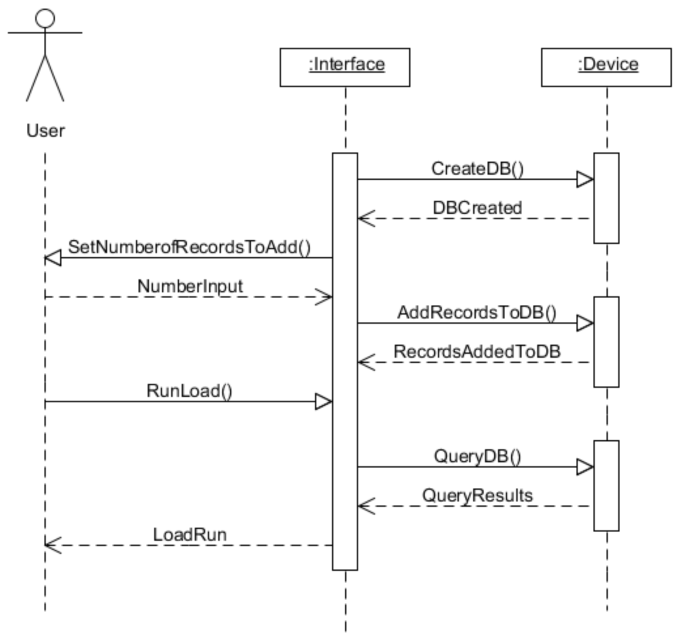


Figure 3.4: Load Sequence Diagram

Buffer Track Use Case

The purpose of the buffer task was to buffer a MP3 file from an on-line source and have it played back. The response time was record for the MP3 file to completely buffer, and for the MP3 file to play from when the user selected play, to when the file finished. Figure 3.5 gives a graphical representation of how the user interacted with the application to buffer the MP3 from an on-line source and play it back. Table 3.3 describes in detail what steps must be carried out in order for the buffer task. The user selects the buffer tab which sets the source of a MP3 from on-line. The track immediately begins buffering. The user plays the buffered track by selecting the play button.

Use Case	Buffer Track
Objective	The user plays a track from an on-line source.
Precondition	The device must be connected to the internet.
Main Flow	1. The user selects the buffer tab. 2. Track source is assigned from on-line, begins buffering. 3. The user presses the play button. 4. Track is played from on-line source to the end.
Alternative Flow	1. The device is not connected to the internet.
Post Condition	The track is buffered and played from an on-line source.

Table 3.4: Play Track Use Case Description.

Figure 3.5 shows how the data source for the MP3 file was set, which then began buffering The track began to play once the user requests it. The Interface and Device refer to the same as the tread (Fig 3.2), file (Fig 3.3) and load (Fig 3.4) sequence diagrams. However in this diagram there is also Server, which refers to the on-line source for the MP3 being used.

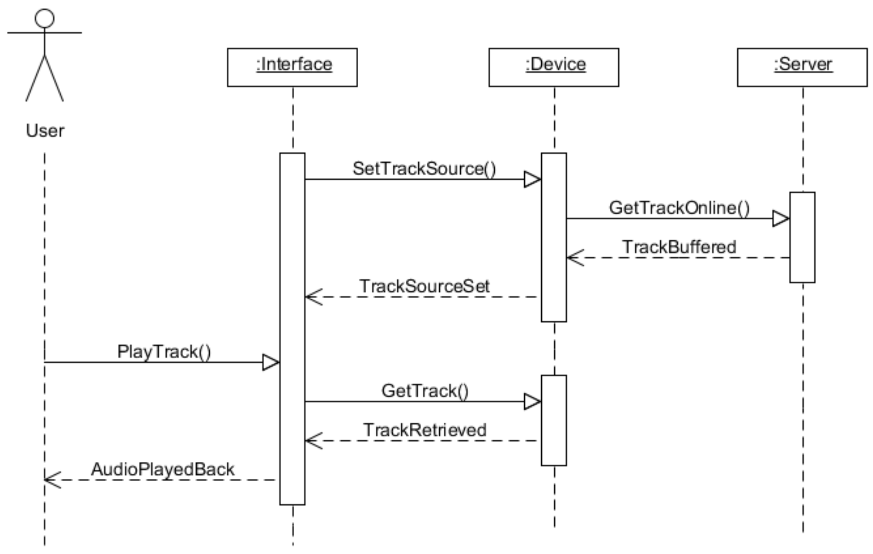


Figure 3.5: Buffer Sequence Diagram

Load web page Using/Not Using Cache Use Case

The purpose of the cache task was to load a web page containing images from an Apache server. The user had the option of loading the page with or without the cache. Loading the page without the cache meant the page was loaded fresh from the server each time. Loading the page using the cache meant the page would be loaded from the device's cache memory, and so would load in a fraction of the time it would have from the server. Figure 3.6 gives a graphical representation of how the user interacted with the application to load the page both with and without the cache. Table 3.5 describes in detail what steps must be done in order to carry out the task while using the cache to load the page.

Use Case	Load web page Using Cache
Objective	A web page is loaded having been cached.
Precondition	1. The device must be connected to the internet. 2. The Apache server must be running.
Main Flow	1. The user checks the check-box to enable caching. 2. The user clicks the load page button. 3. The web page is loaded and is cached. 4. The user clicks the load page button again. 5. The web page reloads faster.
Alternative Flow	1. The device is not connected to the internet. 2. The Apache server is down.
Post Condition	The web page is cached and loads faster.

Table 3.5: Load web page Using Cache Use Case Description.

Table 3.6 describes in detail what steps must be done in order to carry out the cache task without using the cache to load the page.

Use Case	Load web page Not Using Cache
Objective	A web page is loaded without being cached.
Precondition	1. The device must be connected to the internet. 2. The Apache server must be up.
Main Flow	1. The user un-checks the check-box to disable caching. 2. The user clicks the load page button. 3. The web page is loaded and is not cached. 4. The user clicks the load page button again. 5. The web page takes the same amount of time to load.
Alternative Flow	1. The device is not connected to the internet. 2. The Apache server is down.
Post Condition	The web page is not cached and takes the same amount of time to load.

Table 3.6: Load web page Not Using Cache Use Case Description.

Figure 3.6 shows how the user interacted with the application in order to load the page using the cache and without using the cache. The Interface and Device are the same as the other tasks, however in this case the Server refers to an Apache server that is hosting the website to be loaded.

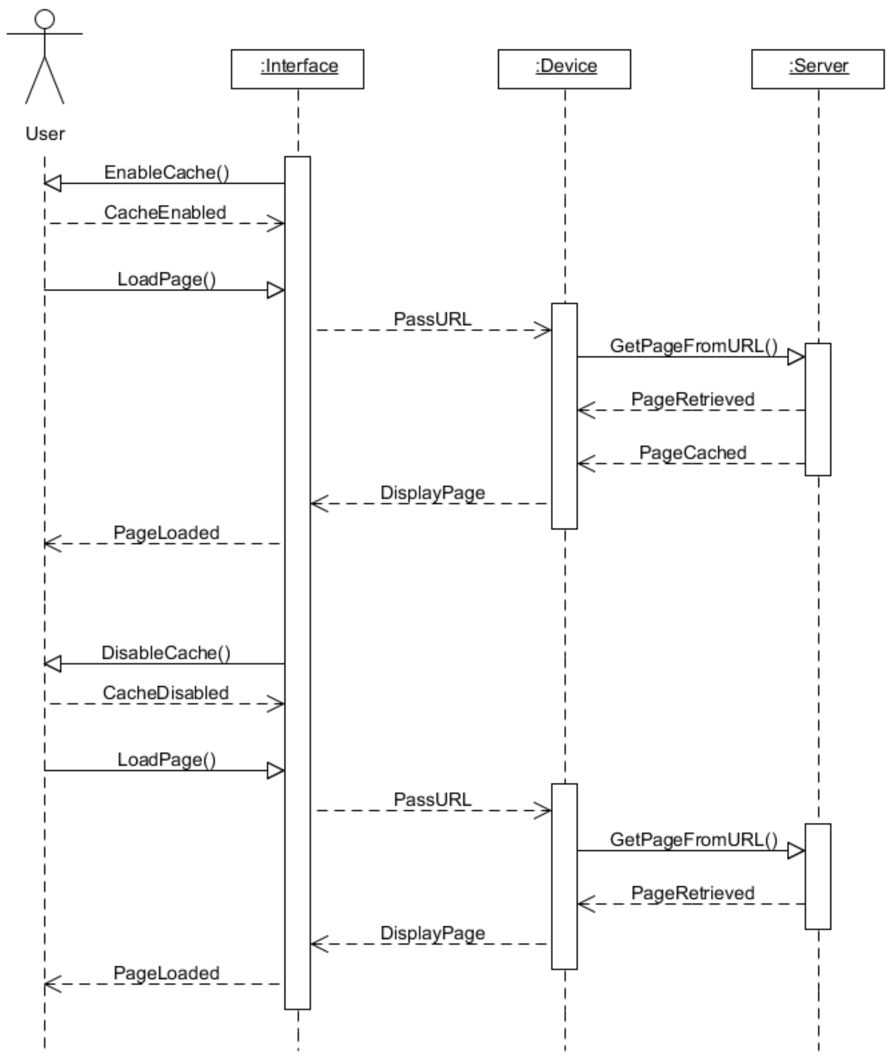


Figure 3.6: Cache Sequence Diagram

3.4.2 Class Diagrams

The class diagrams show the layout of the classes for the Android and Windows Phone 7 versions of the PerformanceApp software. The class layout is slightly different between the two operating systems as the Android application requires an additional class named PerformanceApp. This class solely exists to contain the navigation tabs for the application.

Android Class Diagram

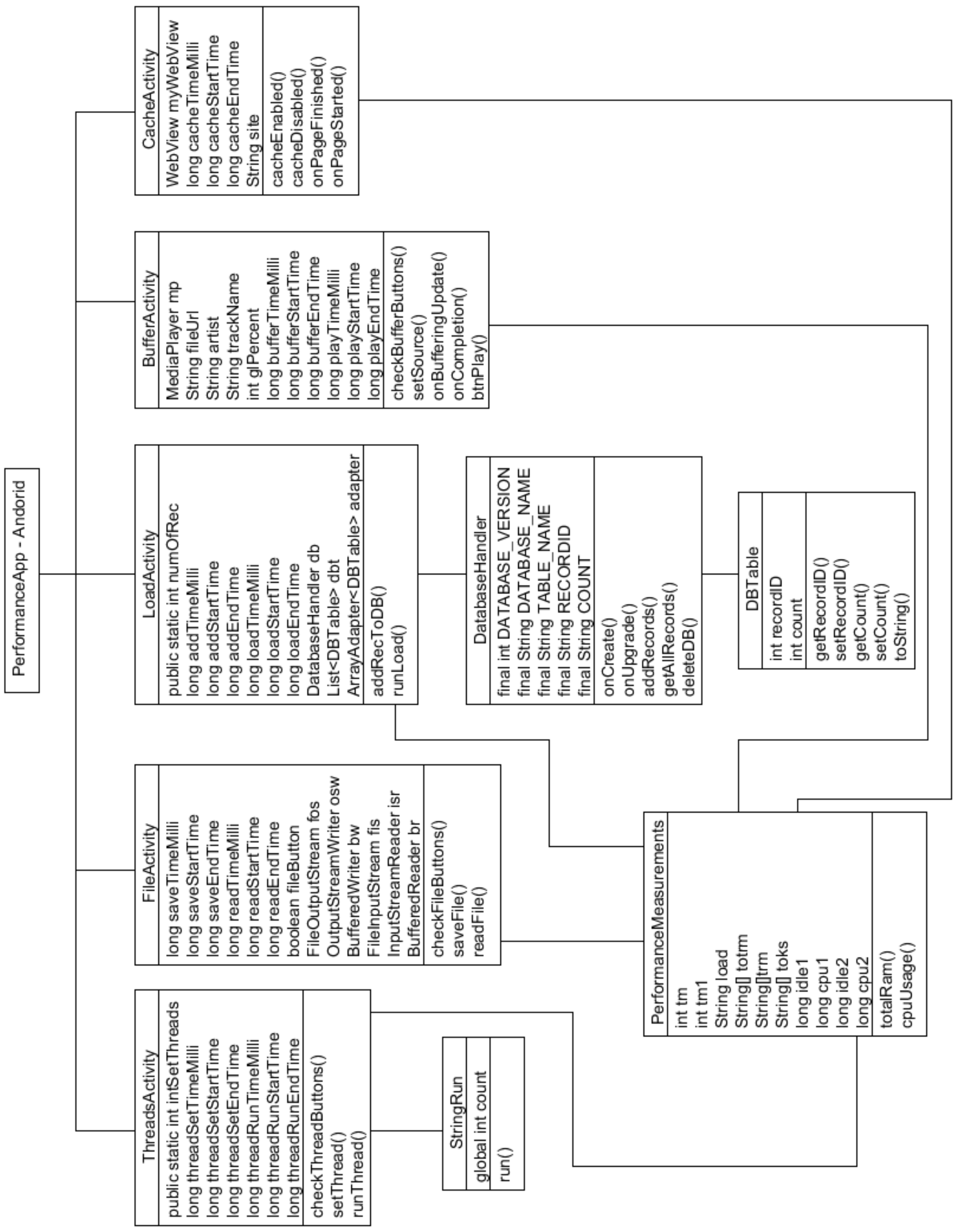


Figure 3.7: Android Class Diagram

Windows Phone 7 Class Diagram

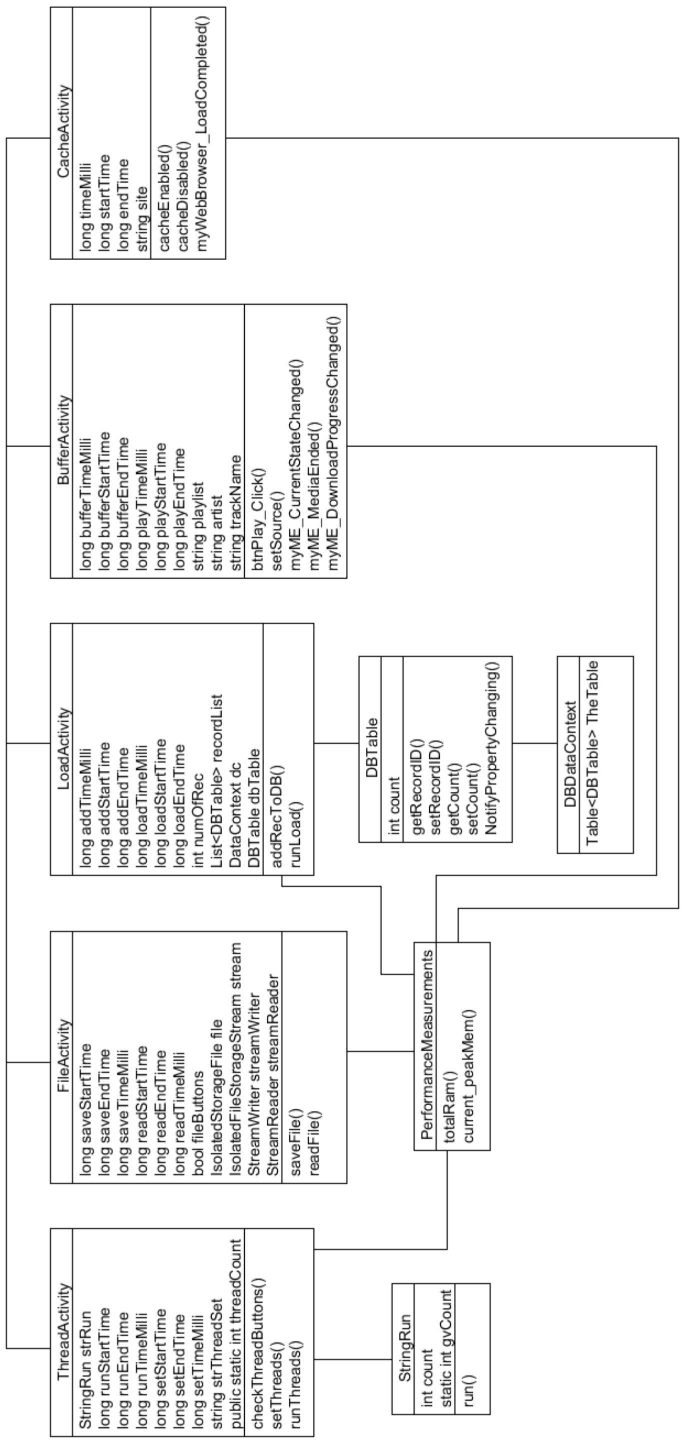


Figure 3.8: Windows Phone 7 Class Diagram

3.4.3 System Flow Diagram

The system flow diagram gives an overview of what the application does with various interactions.

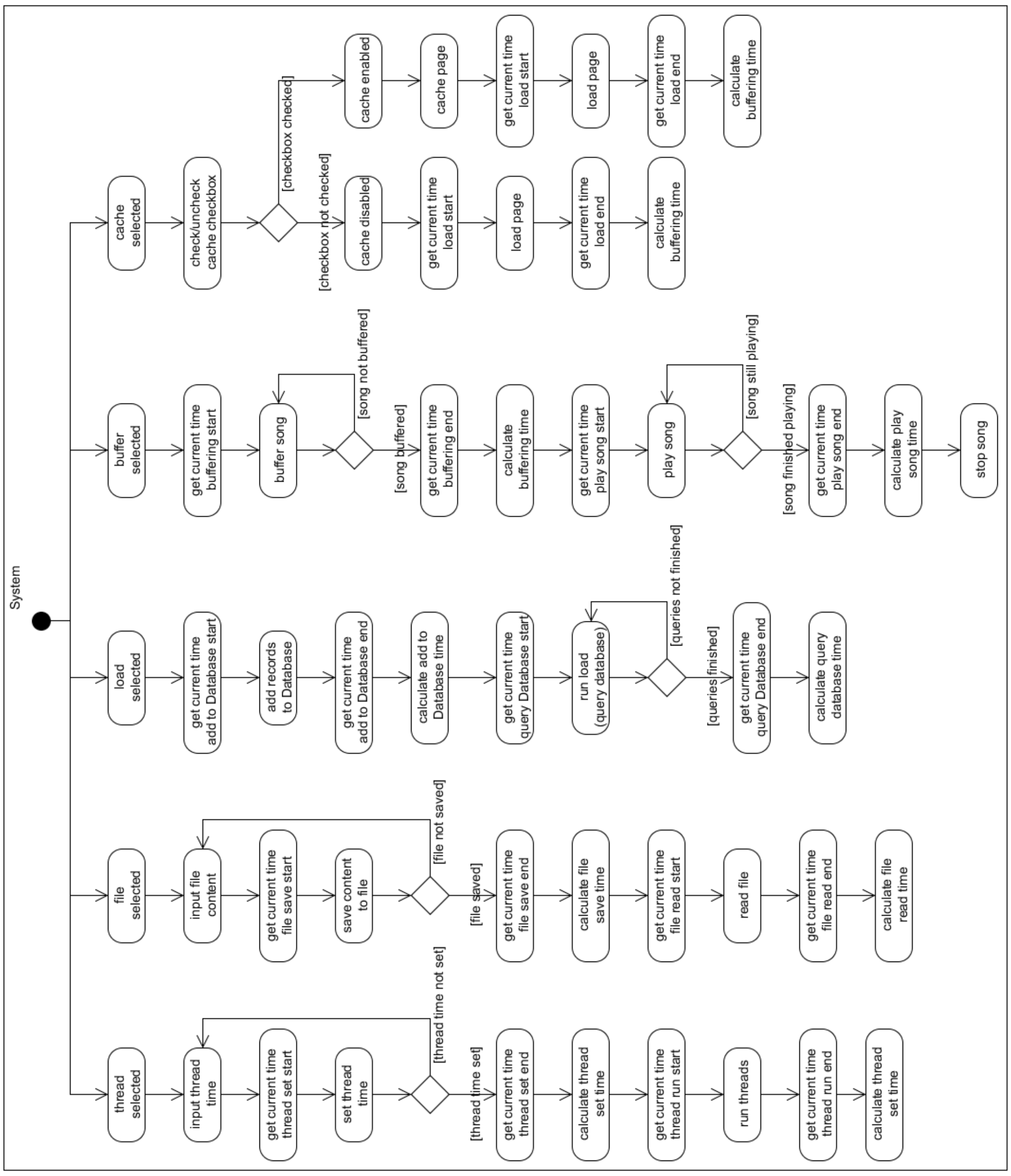

System
cache selected
check/uncheck cache checkbox
[checkbox checked]
[checkbox not checked]
cache enabled
cache page
get current time load start
load page
get current time load end
calculate buffering time
cache disabled
get current time load start
load page
get current time load end
calculate buffering time
buffer selected
get current time buffering start
buffer song
[song not buffered]
[song buffered]
get current time buffering end
calculate buffering time
get current time play song start
play song
[song still playing]
[song finished playing]
get current time play song end
calculate play song time
stop song
load selected
get current time add to Database start
add records to Database
get current time add to Database end
calculate add to Database time
get current time query Database start
run load (query database)
[queries not finished]
[queries finished]
get current time query Database end
calculate query database time
file selected
input file content
get current time file save start
save content to file
[file not saved]
[file saved]
get current time file save end
calculate file save time
get current time file read start
read file
get current time file read end
calculate file read time
thread selected
input thread time
get current time thread set start
set thread time
[thread time not set]
[thread time set]
get current time thread set end
calculate thread set time
get current time thread run start
run threads
get current time thread run end
calculate thread set time

Figure 3.9: System Flow Diagram

3.5 Software Implementation

The following code snippets are from the Android and Windows Phone 7 versions of the PerformanceApp software. The purpose of which is to provide a brief explanation of how the various tasks were completed on both operating systems. Issues encountered during the development of this software are highlighted here. As mentioned in Chapter refch:literatureReview, the Android version of the PerformanceApp software was developed using the Eclipse IDE, and the Windows Phone 7 version was developed using Microsoft's Visual Studio 2010 Express for Windows Phone.

3.5.1 Threads

The purpose of the thread task was to retrieve a count number from the user. This number was then used to spawn off a count from zero to the number desired. The response time for the task to complete was recorded. This was the first task that was developed for this thesis.

Android

Listing 3.1 shows the Android code for the thread task. This code demonstrates how the run() method was used to put the main thread to sleep in order for a background thread to count to the desired number. The biggest issue with the thread task was deciding on the logic to use in order to run the task. In the original design, the user could not set the thread count. The thread count was pre-set to 100. However, this was changed to the current state in order to make the system more dynamic and to retrieve more meaningful results from testing.

```java
// runThread() from ThreadsActivity.java
public void runThread() {
    ...
    strRun.start();
    do{
        try{
            Thread.sleep(1);
        }
        ...
    }while(strRun.count != intSetThreads);
    ...
}

// run() from StringRun.java
public void run(){
    ...
    do{
        count++;
    } while(count < ThreadsActivity.intSetThreads);
    ...
}
```

Listing 3.1: Android Thread Code

Windows Phone 7

Listing 3.2 shows the Windows Phone 7 code for the thread task. Listing 3.1 and 3.2 evidently show that despite the difference in programming languages, both versions of the PerformanceApp software use the exact same logic. The code for both is almost identical.

```
// runThread() from ThreadActivity.xaml.cs
public void runThread(){
    ...
    strRunThread.Start();
    do{
        Thread.Sleep(1);
    } while (StringRun.gvCount != threadCount);
    ...
}

// run() from StringRun.cs
public void run(){
    ...
    do{
        count++;
    } while (count < ThreadActivity.threadCount);
    ...
}
```

Listing 3.2: Windows Phone 7 Thread Code

3.5.2 File

The purpose of the file task was to have text that was input by the user saved to a file stored in internal memory. The file was then read back from internal memory and its content displayed on the screen. This was the second task developed for the PerformanceApp software.

Android

Listing 3.3 shows how the file task was carried out on the Android version of the PerformanceApp software. The code demonstrates how the file task was performed. Overall the file task was the easiest of all the tasks on both operating systems. Time permitting, a larger file such as an image would have been used. However, for the scale of this project, the text file was sufficient.

```java
// saveFile() from FileActivity.java
public void saveFile() {
    ...
        FileOutputStream fos = openFileOutput("textFile.txt", Context.
            MODE_PRIVATE);
        OutputStreamWriter osw = new OutputStreamWriter(fos);
        BufferedWriter bw = new BufferedWriter(osw);
        bw.write(txtFileText.getText().toString());
        ...
    ...
}

// readFile() from FileActivity.java
public void readFile() {
    ...
        FileInputStream fis = openFileInput("textFile.txt");
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);
        lbFileContent.setText(br.readLine());
        ...
    ...
}
```

Listing 3.3: Android File Code

Windows Phone 7

Listing 3.4 and Listing 3.3 show that the logic for the file task on both operating systems are identical. The difference in how the Android version and the Windows Phone 7 version was implemented was the method names. Although named differently, the method functionality was the same.

```
// saveFile() from FileActivity.xaml.cs
void saveFile(){
    IsolatedStorageFile file = IsolatedStorageFile.
        GetUserStoreForApplication();
    IsolatedStorageFileStream stream = file.CreateFile(@"
        performAppTextFiles/textFile.txt");
    StreamWriter streamWriter = new StreamWriter(stream);
    streamWriter.Write(txtFileText.Text);
    ...
}

// readFile()
void readFile(){
    IsolatedStorageFile file = IsolatedStorageFile.
        GetUserStoreForApplication();
    ...
        IsolatedStorageFileStream stream = file.OpenFile(@"
            performAppTextFiles/textFile.txt", FileMode.Open);
        StreamReader streamReader = new StreamReader(stream);
        lbFileContent.Text = streamReader.ReadToEnd();
    ...
}
```

Listing 3.4: Windows Phone 7 File Code

3.5.3 Load

The purpose of the load task was to have the user input a number to determine how many records would be added to the database. Upon the records being added to the database, a query could then be run to retrieve all the records and have them displayed on the screen. The load task was the third and most challenging task as the implementation between the two operating systems was unavoidably different. Originally the load task automatically added 1000 records to the database when opened. This was changed to allow more interaction between the user and the system.

Android

Listing 3.5 shows how the load task was performed on the Android operating system. The Android operating system used SQLite for the load task. As previously mentioned, the load task was the most difficult task to develop as both operating systems required different methods of database connectivity to complete the task. The biggest challenge was making the logic and syntax as similar as possible between both methods. As Listing 3.5 shows, methods where created in a DatabaseHandler.java class in order to carry out the functions. These methods were called in the LoadActivity.java class.

```java
// addRecords() from DatabaseHandler.java
public void addRecords() {
    ...
    for(int i=1; i<=LoadActivity.numOfRec; i++){
        values.put(KEY_COUNT, i); // count
        db.insert(TABLE_NAME, null, values);
    }
            ...
}

// getAllRecords() from DatabaseHandler.java
public List<DBTable> getAllRecords() {
    ...
    String selectQuery = "SELECT  * FROM " + TABLE_NAME;
    ...
    Cursor cursor = db.rawQuery(selectQuery, null);
    // looping through all rows and adding to list
    if (cursor.moveToFirst()) {
        do {
            ...
            dbt.setRecordID(Integer.parseInt(cursor.getString(0)));
            dbt.setCount(Integer.parseInt(cursor.getString(1)));
            recordList.add(dbt);
        } while (cursor.moveToNext());
    }
    ...
    return recordList;
}
```

Listing 3.5: Android Load Code

Windows Phone 7

The Windows Phone 7 operating system used Linq-to-SQL for database connectivity in the load task. This differed to SQLite used for Android. Linq-to-SQL used an instance of the DBDataContext.cs class and used predefined methods to carry out the load task. Using two different methods resulted in the load task taking the longest to develop, whereas with the majority of the other tasks, the logic for both versions was similar, if not identical, requiring only syntax changes.

```
// addRecToDB() from LoadActivity.xaml.cs
public void addRecToDB(){
    ...
    using (var dc = new DBDataContext()){
        dc.DeleteDatabase();
        if (!dc.DatabaseExists()){
            dc.CreateDatabase();
        }
        ...
        for (int i = 1; i <= numOfRec; i++){
            var dbTable = new DBTable(){
                Count = i
            };
            dc.TheTable.InsertOnSubmit(dbTable);
        }
        dc.SubmitChanges();
        ...
    }
}

// runLoad() from LoadActivity.xaml.cs
public void runLoad(){
    ...
    using (var dc = new DBDataContext()){
        var dbTableQuery = from record in dc.TheTable select record;
        ...
        List<DBTable> recordList = dbTableQuery.ToList();
        ...
    }
}
```

Listing 3.6: Windows Phone 7 Load Code

3.5.4 Buffer

The purpose of the Buffer task was to have a MP3 file buffered to the device from an on-line source. The user could then request to have the MP3 played back via an instance of the Android MediaPlayer, or the Windows Phone 7 MediaElement. This was the fourth task developed for the PerformanceApp software.

Android

Listing 3.7 shows the implementation for the buffer task on the Android version of the PerformanceApp software. The Android version of the buffer task presented more challenges than the Windows Phone 7 version. With the Android version there were multiple options for media playback. Some of these were specifically for audio and some were for video. The Android MediaPlayer was selected to playback the MP3 file. Originally the buffer task was to buffer and play a video as this would have shown the effect of buffering better. Unfortunately the Android emulator does not currently support video playback, therefore it was necessary to buffer an audio track. An issue that arose while developing the buffer task on the Android operating system was determining when the MP3 file was finished buffering. This was resolved by using predefined methods for the MediaPlayer, such as the onBufferingUpdate() method.

```java
// onBufferingUpdate() from BufferActivity.java
public void onBufferingUpdate(MediaPlayer arg0, int percent) {
    ...
    glPercent = percent;
    if(glPercent > 0 && glPercent < 10){
        Log.d(null, "buffer started");
    }
    else if(glPercent < 100){
        Log.d(null, "onBufferingUpdate percent:" + percent);
        ...
    }
    else if(glPercent == 100){
        ...
        Log.d(null, "buffer finished");
        ...
    }
}

// onCompletion() from BufferActivity.java
public void onCompletion(MediaPlayer arg0) {
    ...
    Log.d(null, "track finished");
    ...
}

// setSource() from BufferActivity.java
public void setSource() {
    ...
    try {
        ...
        mp.setDataSource(fileUrl);
            ...
        mp.prepare();
    }
    ...
}

// btnPlays() from BufferActivity.java
btnPlays.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        ...
        mp.start();
        ...
    }
});
```

Listing 3.7: Android Buffer Code

Windows Phone 7

Listing 3.8 and 3.7 reveal that the logic and methods used for carrying out the buffer task on both operating systems share striking similarities. With Windows Phone 7, the MediaElement is used to playback both audio and video, thus the code did not require modification when video playback failed on the Android version. An issue while developing the Windows Phone 7 version of the PerformanceApp software was determining when the MP3 file had completed buffering. This was solved using the predefined MediaElement method DownloadProgressChanged().

```
// myME_DownloadProgressChanged() from BufferActivity.xaml.cs
public void myME_DownloadProgressChanged(object sender,
    RoutedEventArgs e){
    if (myME.DownloadProgress == 1.00){
        ...
        Debug.WriteLine(DateTime.Now + ": buffer finished");
        ...
    }
    Debug.WriteLine(DateTime.Now + ": buffering percent: " + myME.
        DownloadProgress * 100.00);
    ...
}

// myME_MediaEnded() from BufferActivity.xaml.cs
public void myME_MediaEnded(object sender, RoutedEventArgs e){
    ...
    Debug.WriteLine(DateTime.Now + ": track finished");
    ...
}

// setSource() from BufferActivity.xaml.cs
public void setSource(){
    ...
    myME.Source = new Uri(playlist, UriKind.RelativeOrAbsolute);
}

// btnPlay_Click() from BufferActivity.xaml.cs
private void btnPlay_Click(object sender, RoutedEventArgs e){
    ...
    myME.Play();
    ...
}
```

Listing 3.8: Windows Phone 7 Buffer Code

3.5.5 Cache

The purpose of the Cache task was to load a web page containing images from an Apache server. The page could be loaded with the cache enabled or disabled. This was the fifth and final task developed for the PerformanceApp software. For the scale of this thesis, loading a web page was sufficient, however if more time was available this task could be expanded. An area which would provide further research would be to use the cache memory for loading maps. The concept of using maps for the cache task was considered for this thesis. Regrettably for the time that was available this was not possible as the use of Google Maps on Android requires a special key from Google. This key must be requested from Google as it is required in to use the Google Maps API.

Android

Listing 3.9 details how the cache task was performed on the Android version of the PerformanceApp software. The Android version of the cache task was easier to develop than the Windows Phone 7 version. Android allows a developer to make changes to aspects of a devices cache, such as clear the cache. This made it possible to clear the cache in order for the web page to be loaded fresh from the Apache server when required.

```
// cacheEnabled() from CacheActivity.java
```

```
public void cacheEnabled(){
    ...
    myWebView.loadUrl(site);
}

// cacheDisabled() from CacheActivity.java
public void cacheDisabled(){
    ...
    myWebView.clearCache(false);
    ...
    myWebView.loadUrl(site);
    }

// onPageFinished() from CacheActivity.java
public void onPageFinished(WebView view, String url) {
    ...
    Log.d(null, "page load time: " + timeMilli);
}
```

Listing 3.9: Android Cache Code

Windows Phone 7

Listing 3.10 and Listing 3.9 demonstrate the similarities between both versions of the cache task. The biggest difference, and biggest challenge with the cache task was stopping the web page being saved to the cache memory of the Windows Phone 7 device. Unfortunately Windows Phone 7 does not allow developers any access to the devices cache memory, unlike Android, which allows the cache to be cleared. In order to over come this issue a way of avoiding saving to the cache memory was developed. By loading the intended page from a dummy page that redirected to the intended page allowed the it to be loaded without using the cache memory.

```
// cacheEnabled() from CacheActivity.xaml.cs
void cacheEnabled(){
    ...
    myWebBrowser.Navigate(new Uri(site, UriKind.Absolute));
}

// cacheDisabled() from CacheActivity.xaml.cs
void cacheDisabled(){
    ...
    myWebBrowser.Navigate(new Uri(site, UriKind.Absolute));
}

// muWebBrowser_LoadCompleted() from CacheActivity.xaml.cs
private void myWebBrowser_LoadCompleted(object sender,
    NavigationEventArgs e){
```

```
    ...
    Debug.WriteLine(DateTime.Now + ": loaded: page loaded");
    ...
}
```

Listing 3.10: Windows Phone 7 Cache Code

3.5.6 Performance Metrics

In order for the performance of the two versions of the PerformanceApp software to be compared, information had to be gathered. The Android and Windows Phone 7 versions of the PerformanceApp software gathered this information in different ways. The methods are discussed here.

Android

For the Android version, the CPU usage and response times were recorded through code that was written and the current memory usage was gathered using the *Memory Usage* application. Listing 3.11 shows the cpuUsage() method from PerformanceMeasurements.java, and a snippet from setThread() method from ThreadActivity.java. As Sub-Section 2.2.1 described, the Android architecture is based on Linux, because of this system information is stored the same way. The cpuUsage() method used a formula, that is also used on Linux, to gain access to the system information. This allowed the CPU usage on the device to be determined. The devices system information was accessed via a RandomAccessReader. This passed all the information into the String variable load. The load variable was then split up and added to a String array toks. The information was passed in to long variables according to their requirement. This was carried out twice and Equation 3.1 was used to determine the CPU usage.

$$(cpu2 - cpu1)/((cpu2 + idle2) - (cpu1 + idle1)) \tag{3.1}$$

The setThread() method shows a snippet of how the response time was calculated for each task. At the beginning of each task a long variable, in this case threadSet-StartTime, was set to the current time in milli seconds. Once the task had completed a second long variable, in this case threadSetEndTime, was set to the current time. A third long variable, threadSetTimeMilli, was then set to the end time minus the start time, thus determining the time taken to complete the task. The response time calculation followed Equation 2.1 defined in Sub-Section 3.5.6. It should be noted that the response times were gathered separate to gathering the memory and CPU usage information, as gathering this information at the same time would have increased the response time and provided inaccurate results.

```java
// cpuUsage() from PerformanceMeasurements.java
public float cpuUsage() {
    try {
        RandomAccessFile reader = new RandomAccessFile("/proc/stat", "r
            ");
        String load = reader.readLine();
        String[] toks = load.split(" ");
        long idle1 = Long.parseLong(toks[5]);
        long cpu1 = Long.parseLong(toks[2]) + Long.parseLong(toks[3]) +
             Long.parseLong(toks[4]) + Long.parseLong(toks[6]) + Long.
            parseLong(toks[7]) + Long.parseLong(toks[8]);
        ...
        reader.seek(0);
        load = reader.readLine();
        ...
        toks = load.split(" ");
        long idle2 = Long.parseLong(toks[5]);
        long cpu2 = Long.parseLong(toks[2]) + Long.parseLong(toks[3]) +
             Long.parseLong(toks[4]) + Long.parseLong(toks[6]) + Long.
            parseLong(toks[7]) + Long.parseLong(toks[8]);
            ...
    ...
}

// setThread() from ThreadActivity.java
public void setThread() {
    ...
    threadSetStartTime = System.currentTimeMillis();
    ...
    threadSetEndTime = System.currentTimeMillis();
    threadSetTimeMilli = threadSetEndTime - threadSetStartTime;
}
```

Listing 3.11: Android Performance Metrics Code

Windows Phone 7

For the Windows Phone 7 version the current memory usage and response times were recorded through code written, and the CPU usage was gathered using the *Windows Phone Performance Analysis Tool*. Listing 3.12 shows the current_peakMem() method from PerformanceMeasurements.cs, and a snippet from setThread() method from ThreadActivity.cs. Silverlight allows access to the DeviceStatus class. This class provides easy access to device information such as the current memory usage. As Listing 3.12 shows, this was the approach taken to retrieve the devices memory usage while running the application. Silverlight contains a class called Analytics which provides system CPU usage details. Unfortunately this is only available for developing desktop sites and so could not be used for this thesis. As previously stated, the CPU usage was retrieved via the *Windows Phone 7 Performance Analysis Tool*. The setThread() method shows a snippet of how the response time was calculated for each task. At the beginning of each task a long variable, in this case setStartTime, was set to the current time in milli seconds. Once the task had completed a second long variable, in this case setEndTime, was set to the current time. A third long variable, setTimeMilli, was then set to the end time minus the start time, thus determining the time taken to complete the task. The response time calculation followed the Equation 2.1 defined in Sub-Section 3.5.6.

As Listing 3.11 and 3.12 both show, the methods used to gather the CPU usage and memory usage were unavoidably different. The methods used to gather the response times were exactly the same. It should be noted that the response times were gathered independently from the memory and CPU usage information. Retrieving this information at the same time would have increased the response time and provided inaccurate results.

```
// current_peakMem() from PerformanceMeasurements.cs
```

```
public void current_peakMem(){
    Debug.WriteLine(DateTime.Now + ": application current memory usage
        : " + DeviceStatus.ApplicationCurrentMemoryUsage.ToString() + "
        bytes, or " + (DeviceStatus.ApplicationCurrentMemoryUsage /
        1048576L) + "mb");
    ...
}

// setThread() from ThreadActivity.xaml.cs
void setThread(){
    setStartTime = DateTime.Now.Ticks / 10000;
    ...
    setEndTime = DateTime.Now.Ticks / 10000;
    setTimeMilli = setEndTime - setStartTime;
    ...
}
```

Listing 3.12: Windows Phone 7 Performance Metrics Code

3.5.7 Summary

There were various differences in developing the Android and Windows Phone 7 versions of the PerformanceApp software. During the development of the PerformanceApp software, it was observed that the Microsoft Visual Studio 2010 for Windows Phone IDE was more beneficial to developers as it made good use of intellisense which provided suggestions while writing code. These suggestions saved a lot of time. It was also observed that certain aspects of Silverlight's C#/XAML were more developer 'friendly'. With C#/XAML, tasks such as changing the font size of a UI element were much more straight forward, whereas, with Android SDK some of these tasks required changes to Android UI element source files.

3.6 Logs

Log files were recorded while running both versions of the PerformanceApp software. These log file contained details of the response times, CPU usage, and memory usage. This information was later used to compare both operating systems and provided data for comparison, which can be seen in Chapter 5. The following is an extract from a sample log taken from read and writing text to a file in internal storage on the Android operating system. The log for the same task on Windows Phone 7 contains the same kind of information.

```
05−24 12:05:43.521: D/(866): fos, osw, bw closed
05−24 12:05:43.541: D/(866): file saved
05−24 12:05:43.541: D/(866): save time: 49
05−24 12:05:45.512: D/(866): file contents: hi there, here is some
    text
05−24 12:05:45.512: D/(866): fis, isr, br closed
05−24 12:05:45.512: D/(866): read time: 34
05−24 12:05:45.512: D/(866): total file task time: 83
```

Listing 3.13: Android File Task Log Sample

A log file was recorded for both versions of the PerformanceApp software. The information from all the events, such as, when the threads finished running, were logged. The times taken to perform tasks such as query the database, etc, were also recorded in the log. The log is used to gather the task times which were then used to determine which operating system performed a task faster. The log files provides unambiguous results. More events could be written to the log; however, writing to a log file takes up resources and can hinder an applications performance. In the case of this thesis, items should be written to the log only when necessary.

4 Testing

4.1 Introduction

The testing chapter examines the specifications of the Android and Windows Phone 7 device emulators. It also observes the specifications for the computer that was used for the development and testing of the two applications. The various tests carried out on both versions of the PerformanceApp software are also discussed.

4.2 Specifications

For the purpose of this thesis, the following was used:

Computer Specifications: The computer used for this thesis was an Acer TravelMate 5730 which ran on an Intel(R) Core(TM) 2 Duo CPU T6570 2.10GHz processor and had 3GB Memory. The machine had 4GB Memory installed but was running a 32-bit operating system so was restricted to 3GB.

Android Emulator: The Android emulator used the target version Android 4.0(API level 14/Ice Cream Sandwich) and had 512MB Memory available.

Windows Phone 7 Emulator: The Windows Phone 7 emulator used the target version Windows Phone 7.1, also known as Mango, and was running on 512MB Memory.

4.3 The PerformanceApp Software

The Android and Windows Phone 7 versions of the PerformanceApp software that were developed for this thesis tested five aspects that affected performance. The five areas were spawning threads, reading and writing to a file (for Disc I/O), database queries (for load), buffering a MP3 from an on-line source, and loading a web page from the devices cache memory or from the server. These five areas were chosen as they are common tasks carried out on Smartphone devices. They are also areas that can absorb a devices resources. The tests carried out were the same for both the Android and Windows Phone 7 versions of the PerformanceApp software so the testing methods discussed apply to both unless stated otherwise.

4.3.1 Threads

Threads are the processes that make an application run. They use a device's resources to execute, therefore the more threads running, the more processes there are and the more resources are being used. For this thesis it was desired to spawn threads in the background and measure the response time and the affect they had on the devices resources. To make the software more dynamic, the count of the threads was set by the user. For the purpose of these tests the thread task was run twenty times on each operating system. For consistency the count was set to 100,000 for every test. 100,000 was chosen because it put a sufficient enough load on the device to determine which performed best and was not so high that it caused the device to freeze up. The response time was measured for setting the count number, and running the threads. These two numbers were added together to determine the time taken to complete the entire task. Section 3.5.1 details how the thread task was completed.

4.3.2 File

Reading and writing to a file is a common task on Smartphones, whether it be a shopping list or a to do list. For this thesis text was retrieved from the user which was then saved to a file stored in the device's internal memory. The file content could then be retrieved from the device and displayed on the screen. Measuring the time taken to read and write text to a file stored on the device shows the disc I/O rate and response time. For the purpose of this thesis the file task was run twenty times on both operating systems. For consistency the same phrase was used throughout. The phrase was 'hi there, here is some text'. The response time was measured for writing the text to the file, and for reading it back. These times were combined to determine the response time for the entire task. The specifics of how the file task was carried out on both operating systems is discussed in Section 3.5.2.

4.3.3 Load

Databases are used for a wide array of tasks from storing website details to employee records. Databases were included in this thesis to show the affect a large database load can have on the performance of an application. In the PerformanceApp software, a database was created and a number of records were added. To make the task more dynamic, the user set the number of records to be added to the database. The user could then run a query against the database where every record in the database was called. For the purpose of this thesis, the tests were run twenty times on each operating system and the same number of records were added for each run. 1000 records were added and called from the database for every test as this was enough to put a substantial load on the device's resources, but not so much that the device froze. The response time was recorded for adding the records to the database and again for querying the database. These two times were added together to show the response time for the entire task. Section 3.5.3 details how the load task was completed.

4.3.4 Buffer

With social networking a major part of peoples lives, more time is spent on-line. This is effecting how we do other tasks too, including how we listen to music. Music is regularly streamed from on-line sources which is particularly important when it comes to Smartphones as they only have a limited amount of memory available. For this thesis when the buffer tab was selected by the user, the source for a media player was set from a url where a MP3 is stored on-line. Once the source was set the track began to buffer. The user could then play back the track from start to finish. The purpose of this was to show how loading media to the buffer and streaming media from the buffer use up device resources and effect the performance. For this thesis, the buffer task was carried out twenty times on each operating system. For consistency the MP3 file used throughout the tests was the same. The response time was measured for the time taken to buffer the file completely, and again for the length of time taken from when the user requested the track to play to when it finished. Since the same track was used, the track length never changed. This allowed the time taken by the operating system to respond to the request to be seen. The particulars of how the buffer task was carried out on both operating systems is examined in Section 3.5.4.

4.3.5 Cache

Smartphones are regularly used to browse the web. A web page containing an array of images was hosted locally on an Apache server. The user could chose to load the page from the device's cache memory, or fresh from the server. The purpose of this test was to show how enabling the cache allowed the page to be load much faster and required less resources. Disabling the cache resulted in the page loaded fresh each time, requiring more time and resources. For this thesis the cache task was carried out twenty times on each operating system. The response time was recorded for the time taken to load the page from the cache, and again from the Apache server.

5 Results

5.1 Introduction

The results chapter reviews the results gathered from the tests performed on the software and their findings. Each task was tested separately and in each case the response time, CPU usage, and memory usage was recorded.

While running the tests, it was observed there were spikes in the time taken by Android and Windows Phone 7 to completed tasks. The reason for this was a rise in CPU usage. As well as the PerformanceApp software running on the system, the emulator was also running background system processes, just as an actual device would. Figure 2.1 shows how the Android operating system runs other processes concurrently with the PerformanceApp software. These background processes can sometimes require more processing power than normal and can push other processes on hold. Even though a process may be pushed back, or put on hold, for just a fraction of a second, it can have an impact on the time taken to carry out a task. Similarly, this sometimes happened in the opposite way. From time to time background processes may end and free up resources allowing a process task to complete faster than normal.

5.2 Threads

The thread task entailed a number, that was set by the user, being used to determine what a thread would count to. The response times were recorded for the time taken to set this number, run the thread, and the total time taken to carry out the entire task. For the purpose of this thesis, the number set by the user for all the tests was 10000. The reason for the number being set to 10000 was to provide enough of a load on the system to take meaningful results. The results of each part of the thread task are discussed below.

5.2.1 Set Thread Count

Fig 5.1 shows the comparison of the average time taken by Android and Windows Phone 7 to set the thread count for the thread task. As Sub-Section 3.5.1 explained, setting the thread count determined what number the thread would count to when run. Setting the thread count entailed retrieving a number from a text-box and converting it to an integer. Fig 5.1 shows that Windows Phone 7 was ten times faster at converting a string to an integer. Windows Phone 7 carried out the task in 3.85 milli seconds (0.00385 seconds), and Android carried out the same task in 40.85 milli seconds (0.0485 seconds). These figures suggest that the Windows Phone 7 operating system may be better suited to applications that require a lot of information to be parsed to another data type. It should be noted that the higher the count number was set, the larger the gap between Windows Phone 7 and Android became.

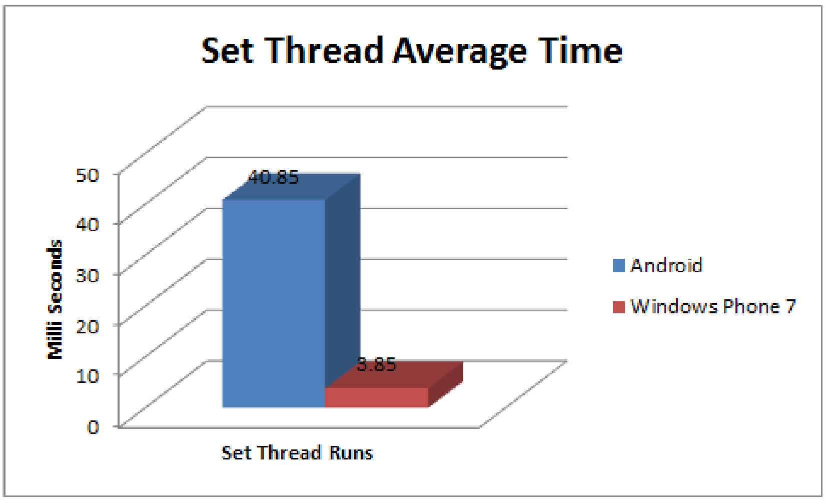


Figure 5.1: Set Thread Average Time

5.2.2 Run Threads

Fig 5.2 shows the comparison of the average time taken by Android and Windows Phone 7 to run the thread count to the number set previously be the user. Sub-Section 3.5.1 showed that the thread run part of the thread task took the number previously set by the user, and set the main thread to sleep while a background thread performed a count to the number set. Fig 5.2 shows that Windows Phone 7 ran the threads six times faster than Android. Windows Phone 7 carried out the task in 37.05 milli seconds (0.03705 seconds) and Android carried out the task in 248.45 milli seconds (0.24845 seconds). These figures clearly show that the Windows Phone 7 operating system is more suited to applications that require running many background threads as it performs the task much more efficiently than Android.

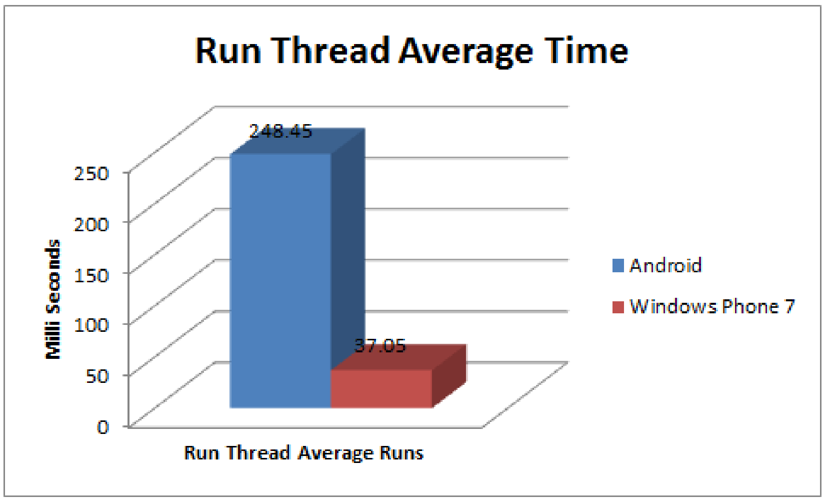


Figure 5.2: Run Threads Average Time

5.2.3 Total Thread Task

The response time was recorded for the thread task as a whole, this was the response time for setting the thread and running the thread added together. Fig 5.3 shows that Windows Phone 7 performed the entire thread task seven times faster than Android. Windows Phone 7 performed the entire task in 40.9 milli seconds (0.0409 seconds) and Android performed the same task in 289.6 milli seconds (0.2896 seconds). Fig 5.1, Fig 5.2, and Fig 5.3 all suggest that the Windows Phone 7 operating system is more efficient and better suited to applications where parsing data types and running threads are going to be a main feature.

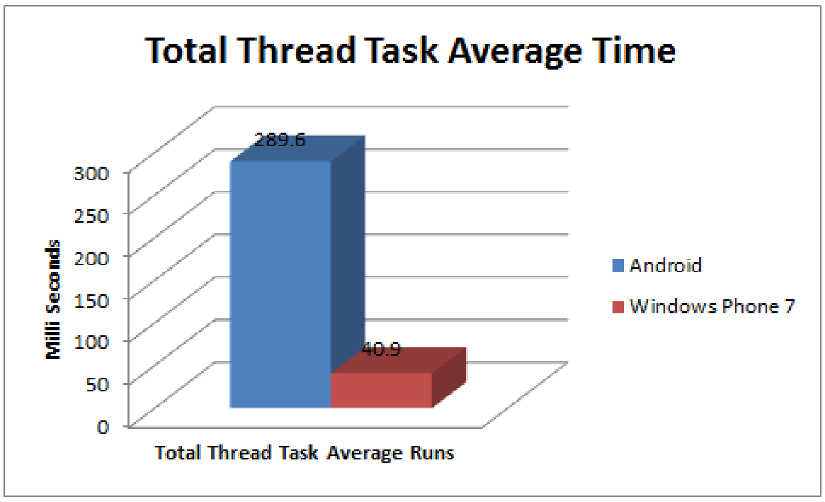


Figure 5.3: Total Thread Task Average Time

The CPU and memory usage were recorded for the thread task as a whole rather than at individual steps. This was to show how the thread task as a whole affected the devices resources. Fig 5.4 shows the average CPU usage the Android and Windows Phone 7 versions of the PerformanceApp software required to carry out the thread task.

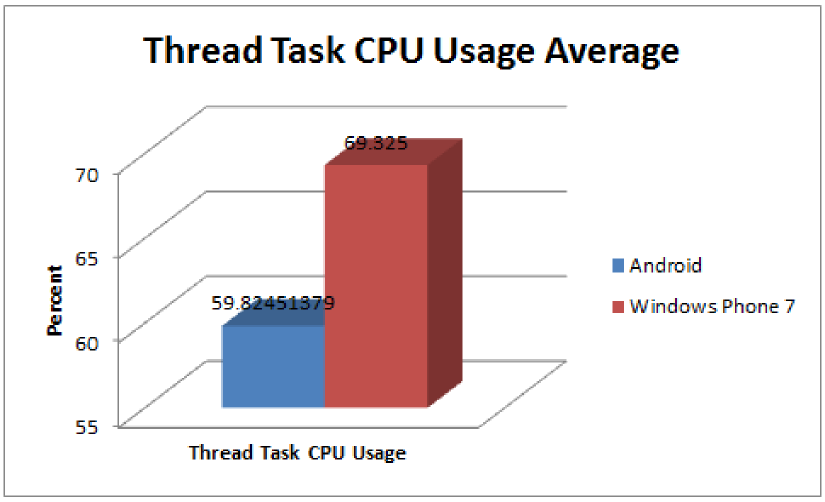


Figure 5.4: Total Thread Task CPU Usage

Fig 5.5, shows the average memory usage in MB, that the applications needed to perform the thread task.

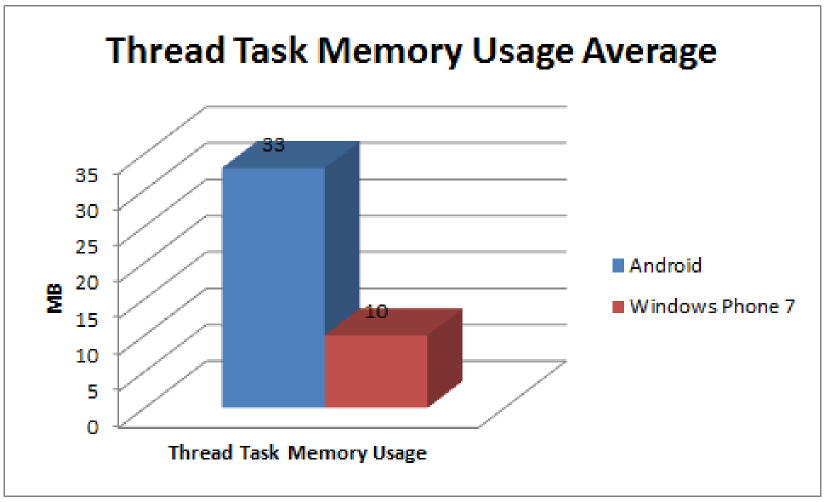


Figure 5.5: Total Thread Task Memory Usage

Fig 5.4 shows that there was, on average, just under a ten percent difference in CPU usage between Android and Windows Phone 7. On average, Android performed the whole thread task with 59.8% CPU usage, while Windows Phone 7 required 69.3% CPU usage to carry out the task. This shows that Windows Phone 7 requires more resources to perform the thread task than Android does. The extra use of resources allowed the Windows Phone 7 operating system to perform the full thread task considerably faster than the Android operating system.

5.3 File

For both the Android and Windows Phone 7 versions of the PerformanceApp software the response time, CPU usage and memory usage were recorded for the file task. This task involved a sentence of text being input by the user, this was then saved to a text file and stored in the devices internal storage. This file was then read back from internal storage and its content displayed on the device's screen. The response times were recorded for the time taken to retrieve the text from the user and save it to the file in internal memory, and again for the time taken to read the files content back from internal memory. These two response times were then added together to determine the overall response time for the entire file task. For the purpose of this thesis and to provide consistency the sentence “hi there, here is some text” was used throughout all the tests. This sentence was chosen as it was long enough to provide meaningful results while the tests were run. The results of each part of the file task are discussed below.

5.3.1 Save/Write to File

Fig 5.6 shows the comparison of the average time taken by Android and Windows Phone 7 to retrieve the sentence from the user and save that sentence to the file saved in the devices internal storage. Sub-Section 3.5.2 explained how the a sentence input by the user was retrieved from a text-box and saved to the file in internal memory. Fig 5.6 shows that, on average, Windows Phone 7 carried out the task twice as fast as the Android version of the PerformanceApp software. Windows Phone 7 carried out the task in 28.8 milli seconds (0.0288 seconds), and Android carried out the same task in 61.95 milli seconds (0.06195 seconds). These figures suggest that the Windows Phone 7 operating system may be better suited to applications that require many writes to files saved in internal storage.

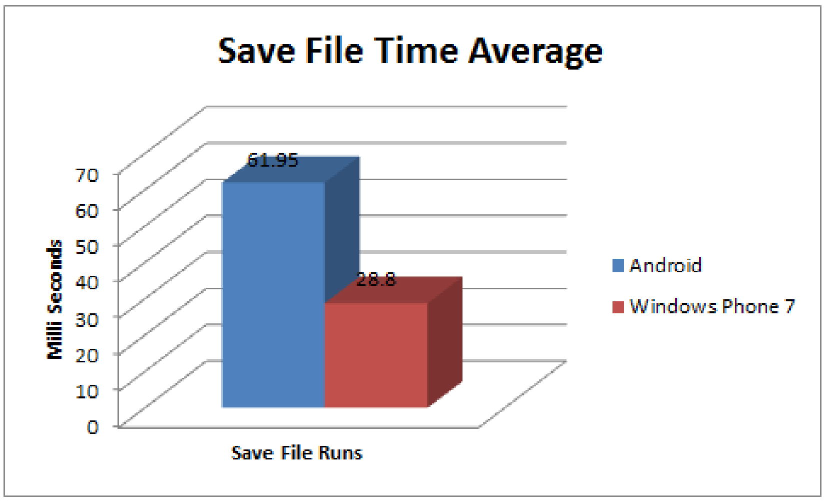


Figure 5.6: Save File Average Time

5.3.2 Read File

Fig 5.7 shows the comparison of the average time taken by Android and Windows Phone 7 to read the content of the file saved in the device's internal memory and display it on screen. Sub-Section 3.5.2 showed how both versions of the PerformanceApp software found the file in the device's internal storage and read its content back. Fig 5.7 reveals that Windows Phone 7 read the file's content back seven times faster than Android. Windows Phone 7 carried out the task, on average, in 7.75 milli seconds (0.00775 seconds) and Android carried out the task, on average, in 55.8 milli seconds (0.0558 seconds). While both these figures show the task was performed quickly by both operating system, they show clearly that the Windows Phone 7 operating system is much better suited to applications that rely heavily on reading information from the devices internal storage.

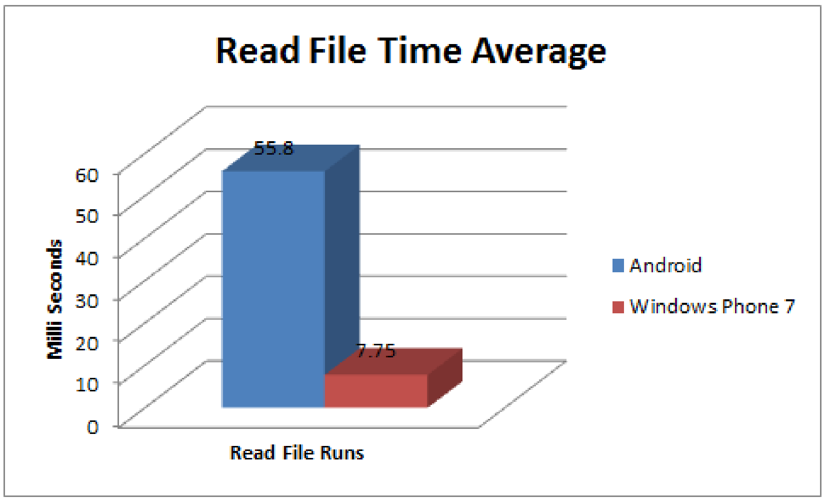


Figure 5.7: Read File Average Time

5.3.3 Total File Task

The response time was recorded for the file task as a whole, this was the response time for retrieving and saving the text to the file in internal storage and then reading the file's content back. Fig 5.8 shows that Windows Phone 7 performed the entire file task three times faster than Android. Windows Phone 7 performed the entire task in an average of 36.55 milli seconds (0.03655 seconds) and Android performed the same task in 117.75 milli seconds (0.11775 seconds). Fig 5.6, Fig 5.7, and Fig 5.8 all suggest that the Windows Phone 7 operating system is more efficient and better suited to applications where reading and writing to files saved in internal storage would be a main feature.

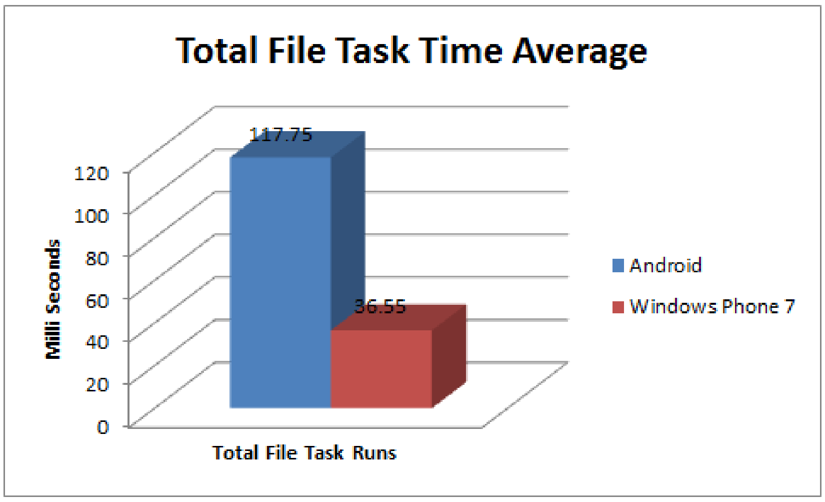


Figure 5.8: Total File Task Average Time

The CPU and memory usage were recorded for the file task as a whole rather than at individual steps. This was to show how the entire file task affected the device's resources. Fig 5.9 shows the average CPU usage the Android and Windows Phone 7 versions of PerformanceApp required to carry out the file task.

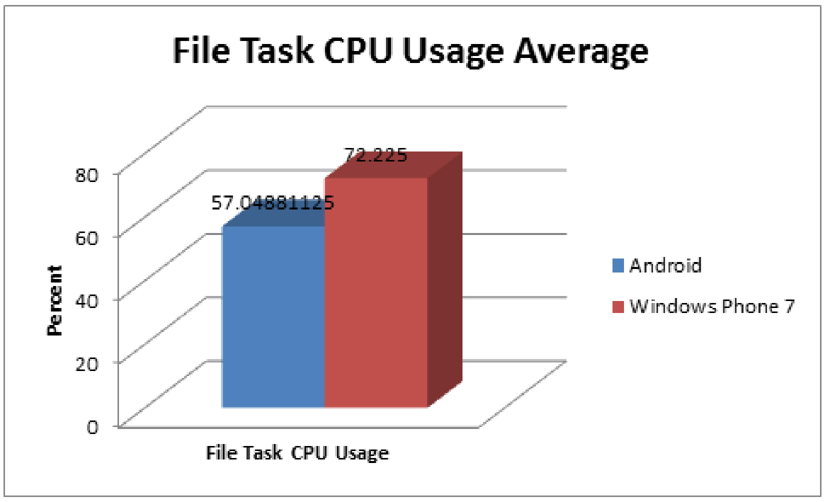


Figure 5.9: Total File Task CPU Usage

Fig 5.10, represents the average memory used by the applications to carry out the file task.

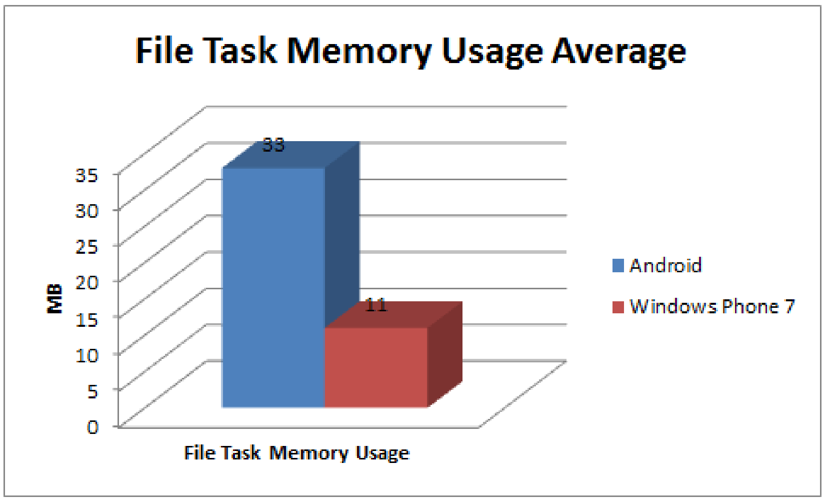


Figure 5.10: Total File Task Memory Usage

Fig 5.9 shows that on average, the Windows Phone 7 version of PerformanceApp software used up 72.225% of the CPU, while the Android version used 57.04881125%. This means Windows Phone 7 used on average, 15.17618875% more processing resources than Android. It should be noted that while the emulators were left in an idle state the Windows Phone 7 CPU usage ranged from roughly 7% to 60%, on average staying around 34.5%, and the Android CPU usage ranged from roughly 10% to 44%, on average staying around 24.2%. Fig 5.10 shows that the average memory usage for both applications was, on average, for Windows Phone 7, 11MB, and for Android, 33MB. This shows that Windows Phone 7 requires more resources to perform the file task than Android does, but less memory. The extra use of resources allowed the Windows Phone 7 operating system to perform the full file task faster than the Android operating system.

5.4 Load

For both the Android and Windows Phone 7 versions of the PerformanceApp software the response time, CPU usage and memory usage were recorded for the load task.

The load task entailed a number that was set by the user being used to to determine the number of records that would be added to a database. This task also queried the same database to retrieve all the records and display them on the screen. The response times were recorded for the time taken to set number of records and add the them to the database, as well as query the database run the thread. The response time for the total time taken to carry out the entire task was also recorded. For the purpose of this thesis, the number of records set by the user for all the tests was 1000. The reason for the number being set to 1000 was to provide enough of a load on the system to take meaningful results. The results of each part of the load task are discussed below.

5.4.1 Create/Add to DB

Fig 5.11 shows the comparison of the average time taken by Android and Windows Phone 7 to set the number of records to be inserted to the database, and add those records. Sub-Section 3.5.3 explained how setting the number of records determined how many records would be added to the database. Those sections also described how this number was used to add records to a database. Fig 5.11 presents that Windows Phone 7 was fifty one times faster at retrieving the record number from the user and adding the records to the database. Windows Phone 7 carried out the task in an average of 791 milli seconds (0.791 seconds), and Android carried out the same task in 40742.75 milli seconds (40.74275 seconds). These figures indicate that the Windows Phone 7 operating system is better suited to applications that require much writing to internal databases. It should be noted that the higher the number of records added to the database, the larger the gap between Windows Phone 7 and Android became. Upon initially viewing these results, the code was rechecked to ensure both applications were in fact performing the same task. It was found that they were. The difference in the times may be down to the emulators being used. During the development of this project it was noticed that the Windows Phone 7 emulator ran considerably smoother and faster than the Android emulator. The Android emulator could, at times, be sluggish and glitchy despite both emulators running from the same amount of RAM. Without access to actual Android and Windows Phone 7 devices there is no way of checking this theory.

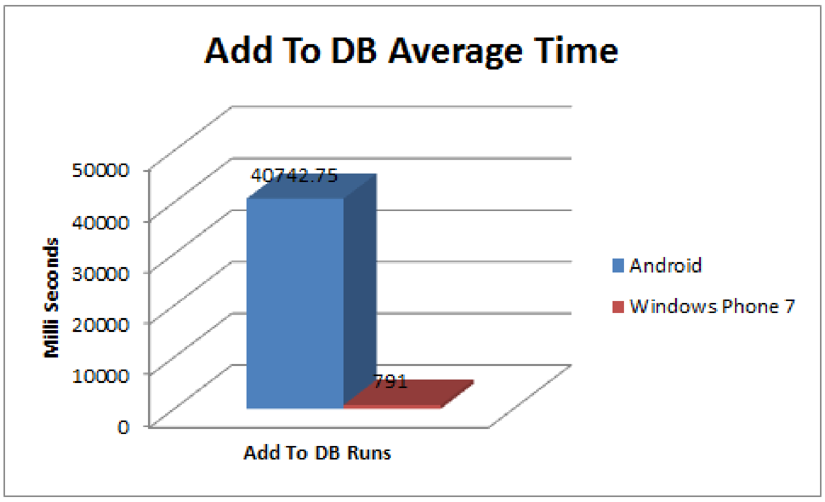


Figure 5.11: Add to DB Average Time

5.4.2 Query DB

Fig 5.12 shows the comparison of the average time taken by Android and Windows Phone 7 to carry out a query against a database. This query retrieved all the records stored in the database, which were added previously. The results of the query were displayed on the devices screen. Sub-Section 3.5.3 demonstrated how all the records stored in the database were retrieved and displayed on the screen. Fig 5.12 shows that Windows Phone 7 ran the threads nine times faster than Android. Windows Phone 7 carried out the task in 289.9 milli seconds (0.2899 seconds) and Android carried out the task in 2897.55 milli seconds (2.89755 seconds). These figures clearly show that the Windows Phone 7 operating system is much better suited to applications that rely on multiple database queries, as it performs the task much more efficiently than Android. As with adding to the database, Fig 5.12 shows that Windows Phone 7 was significantly faster than Android. As previously mentioned with adding the records to the database, the code was reviewed to ensure both applications were performing the same job. It was found that they were. Again, an explanation may be the emulators rather than the code.

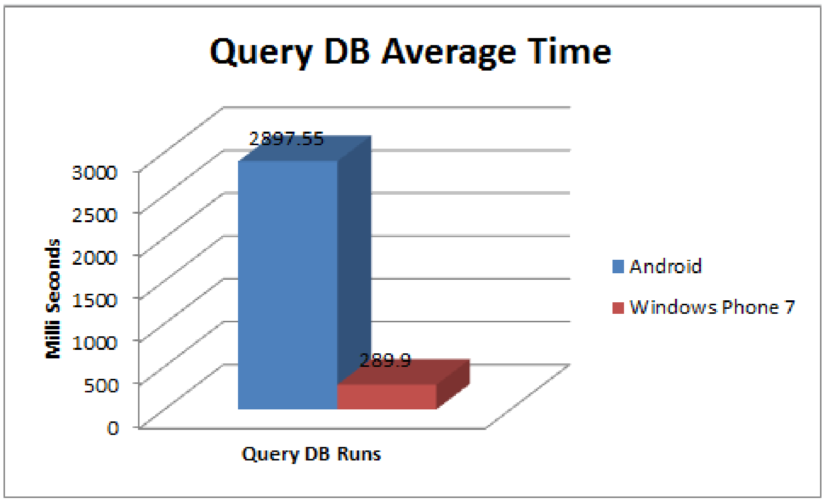


Figure 5.12: Query DB Average Time

5.4.3 Total Load Task

The response time was recorded for the load task as a whole. This was the response time for retrieving and adding the records to the database. Fig 5.3 shows that Windows Phone 7 performed the entire thread task an astonishing forty times faster than Android. Windows Phone 7 performed the entire task in 1080.9 milli seconds (1.0809 seconds) and Android performed the same task in 43640.3 milli seconds (43.6403 seconds). Fig 5.11, Fig 5.12, and Fig 5.13 all prove that the Windows Phone 7 operating system is more efficient and better suited to applications which require constant database use.

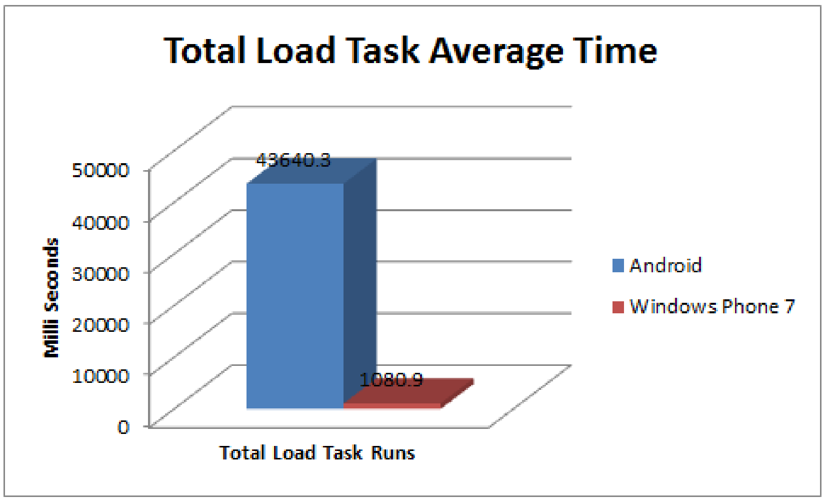


Figure 5.13: Total Load Task Average Time

The CPU and memory usage were recorded for the load task as a whole rather than at individual steps. This was to show how the load task as a whole affected the device's resources. Fig 5.14, shows the average CPU usage that was required by both versions of the application to perform the load task.

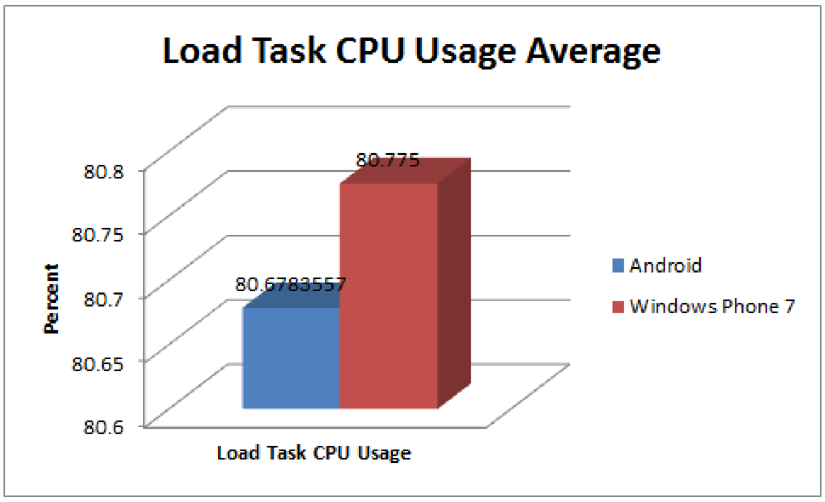


Figure 5.14: Total Load Task CPU Usage

Fig 5.15, represents the average memory used by the two versions of PerformanceApp software to carry out the entire load task.

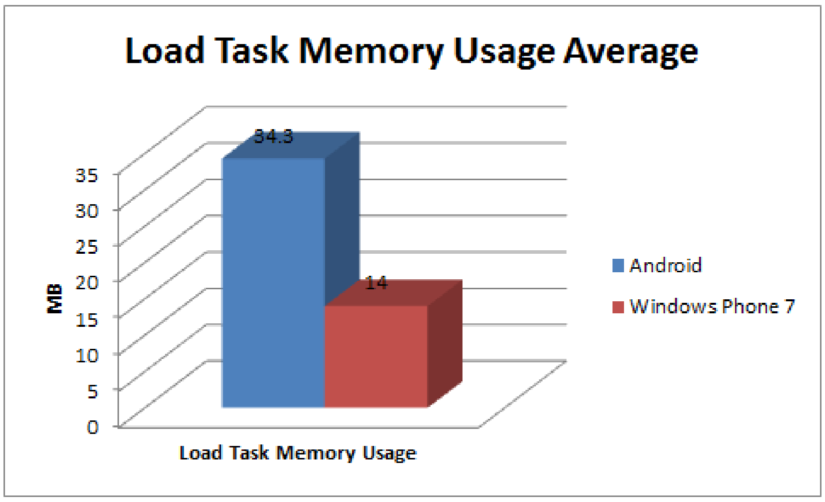


Figure 5.15: Total Load Task Memory Usage

Fig 5.14 shows that there was, on average, just under 0.01% of a difference in CPU usage between Android and Windows Phone 7. On average, Android performed the whole load task with 80.6783557% CPU usage, while Windows Phone 7 required slight more at 80.775% CPU usage. This shows that Windows Phone 7 required more resources to perform the thread task than Android. When weighed up against the speed Windows Phone 7 accomplished the load task, the small increase in resource usage seems insignificant.

5.5 Buffer

The buffer task required a MP3 file to be buffered and played back from an on-line source. The response time was recorded for the time taken to completely buffer the MP3, and again from when the user selected the track to be played, to when the MP3 finished. Since the same file was always used the track length never changed. This clearly showed how long each operating system took to respond to the user's request. As both actions were performed independent of each other, there was no overall response time recorded for the buffer task.

5.5.1 Buffer Track

Fig 5.16 shows the comparison of the average time taken by Android and Windows Phone 7 to completely buffer the MP3 file from an on-line source. Sub-Section 3.5.4 demonstrated how this task was carried out. Fig 5.16 shows that Android was 1.6 times faster at buffering the MP3 file than Windows Phone 7. Android buffered the file in 14777.6 milli seconds (14.7776 seconds), and Windows Phone 7 carried out the same task in 23969.6 milli seconds (23.9696 seconds). Upon reviewing these figures, it would be advisable to use the Android operating system for applications which would involve streaming media.

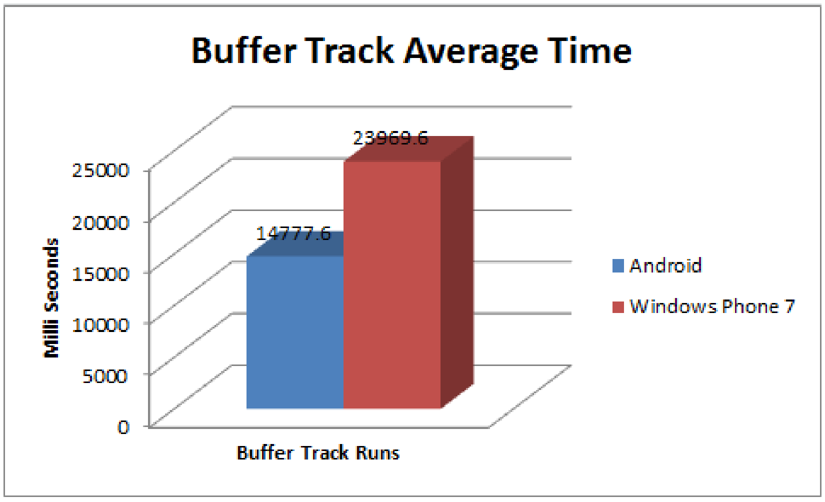


Figure 5.16: Buffer Overall Average Time

5.5.2 Total Track Play Time

Fig 5.17 shows the comparison of the average time taken by Android and Windows Phone 7 to play the buffered MP3 track through from start to finish. Sub-Section 3.5.4 demonstrated how this task was carried out. Fig 5.17 shows that Android responded to the users request to play the file. Android played the 177.797 second in 178084.15 milli seconds (178.08415 seconds), and Windows Phone 7 carried out the same task in

178114.8 milli seconds (178.1148 seconds). The extra time taken was the time taken by the operating systems to respond to the users request. The results suggest that the Android operating system would be more beneficial for applications playing media from the MediaPlayer.

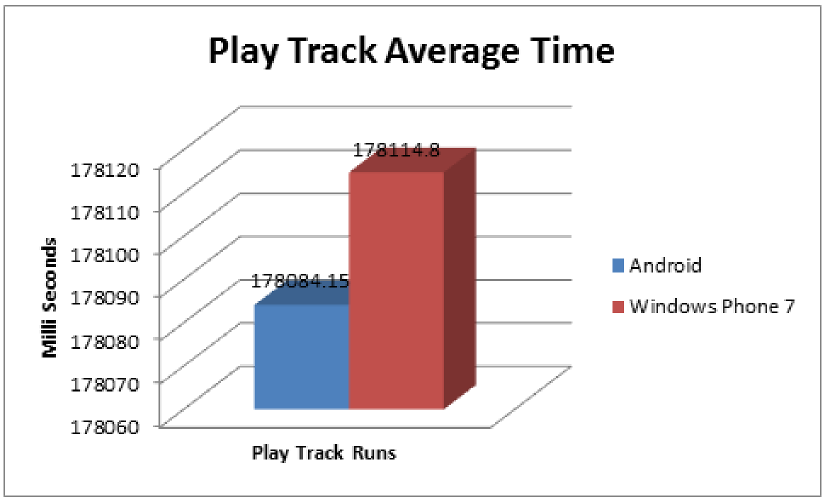


Figure 5.17: Play Track Average Time

5.5.3 Total Buffer Task

Unlike the thread, file and load tasks, the buffer task did not require a measurement to determine how long it took to carry out the entire task, as the user could begin playing the file as soon as it started buffering. The CPU and memory usage were recorded for the entire task in order to see how the task affected the system resources of each emulator. Fig 5.18, represents the average CPU usage required by the two versions of the application to performance the entire buffer task.

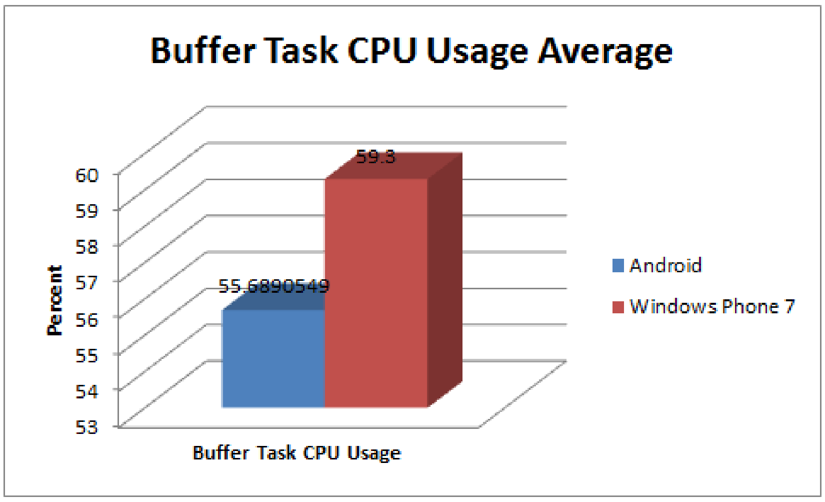


Figure 5.18: Total Buffer Task CPU Usage

Fig 5.19, provides a representation of the average memory that the applications used while performing the buffer task.

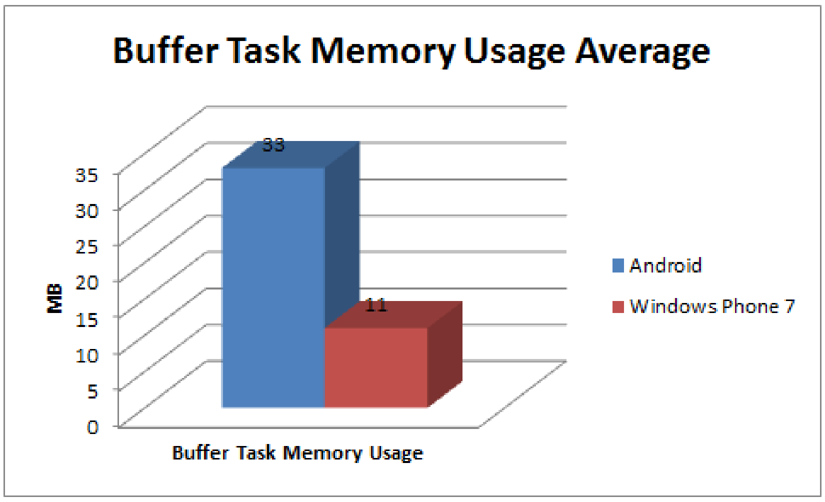


Figure 5.19: Total Buffer Task Memory Usage

Fig 5.18 shows that there was, on average, just a 3.6109451% difference in CPU usage between Android and Windows Phone 7. On average, Android performed the whole buffer task with 80.6783557% CPU usage, while Windows Phone 7 required more at 80.775% CPU usage. This shows that Windows Phone 7 required more resources to perform the buffer task than Android does. This proves that between response times and CPU usage, the Android operating system is better suited to applications constantly streaming music to a MediaPlayer.

5.6 Cache

For both the Android and Windows Phone 7 versions of the PerformanceApp software the response time, CPU usage and memory usage were recorded for the cache task. The cache task involved a web page being loaded from an Apache server. This page could be load from the device's cache memory, or loaded fresh from the server. The response times were recorded loading the page with and without the cache.

5.6.1 Load Page Without Cache

Fig 5.20 shows the comparison of the average time taken by Android and Windows Phone 7 to load the page without using the cache. Sub-Section 3.5.5 described how the applications loaded the page without using the cache. Fig 5.20 shows that Windows Phone 7 was eleven times faster at loading the page from the server than Android. Windows Phone 7 loaded the page in 2403.1 milli seconds (2.4031 seconds), and Android loaded the page in 26612.15 milli seconds (26.61215 seconds). These figures suggest that the Windows Phone 7 operating system has better performance with loading web pages.

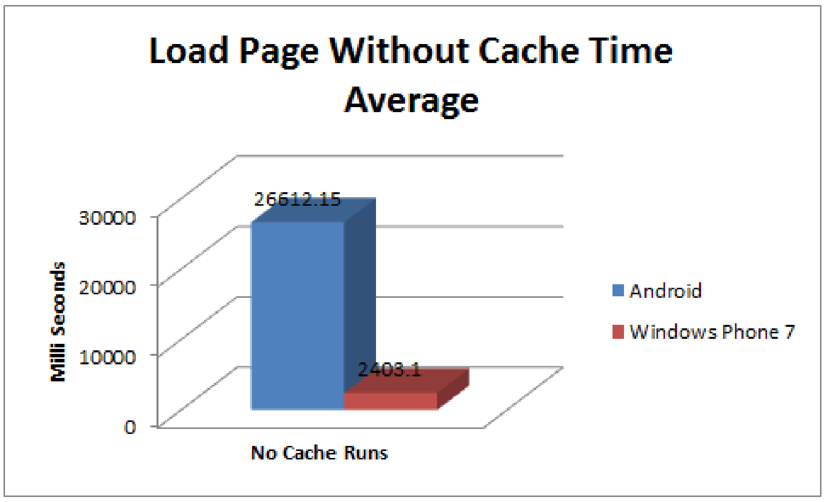


Figure 5.20: Load Page Without Cache Average Time

The CPU and memory usage were recorded for the cache task without using the cache. This was to show how loading a page without using the device's cache memory would impact on the device's resources. Fig 5.21, shows the average CPU usage required to perform loading the page without using the cache.

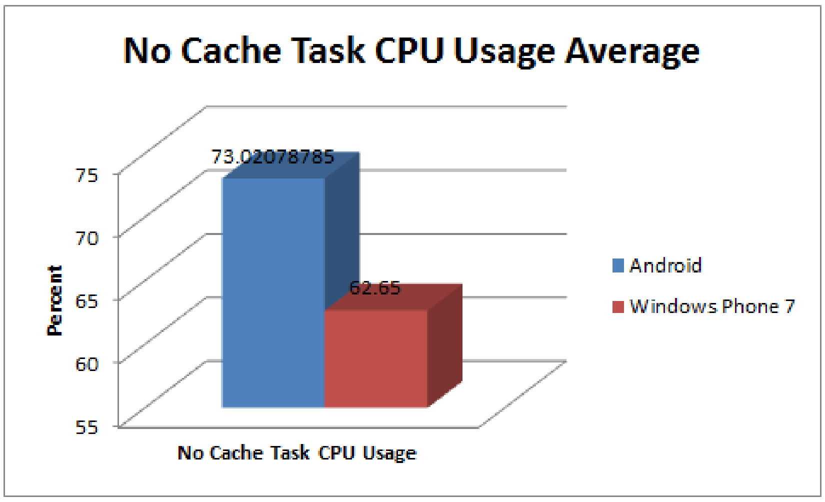


Figure 5.21: No Cache Task CPU Usage

Fig 5.22, represents the average memory required by both versions of the application to load the page without using the cache.

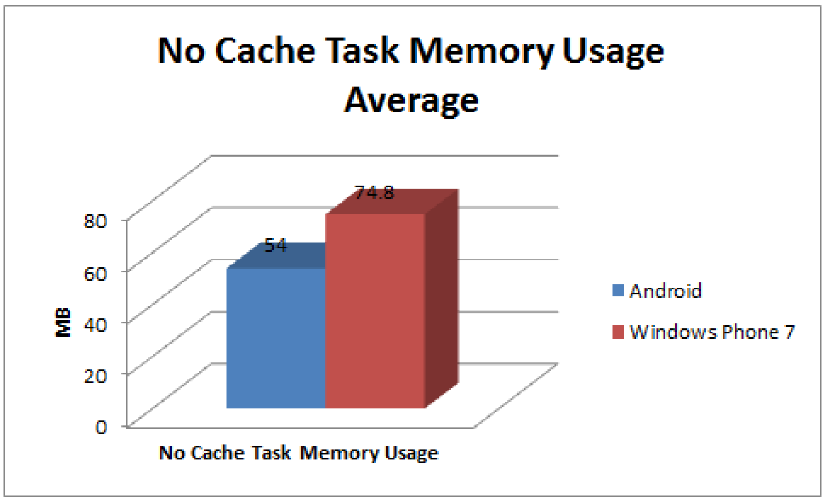


Figure 5.22: No Cache Task Memory Usage

Fig 5.21 shows that there was, on average, a ten percent difference in CPU usage between Android and Windows Phone 7. On average, Windows Phone 7 loaded the page from the server using 62.65% of the CPU. Android required 73.02078785% of the CPU. This shows that Android required more resources to load the page from the server than Android.

5.6.2 Load Page Using Cache

Fig 5.23 shows the comparison of the average time taken by Android and Windows Phone 7 to load the page from the devices cache memory. Sub-Section 3.5.5 described how the applications loaded the page from the cache memory, rather than the server. Fig 5.23 shows that Android was 1.3 times faster at loading from cache memory than Windows Phone 7. Android loaded the page 433.05 milli seconds (0.43305 seconds), and Windows Phone 7 loaded the page in 577.95 milli seconds (0.57795 seconds). This indicates that the Android operating system has faster access to its cache memory, and so would be better suited to applications where this would be a main feature.

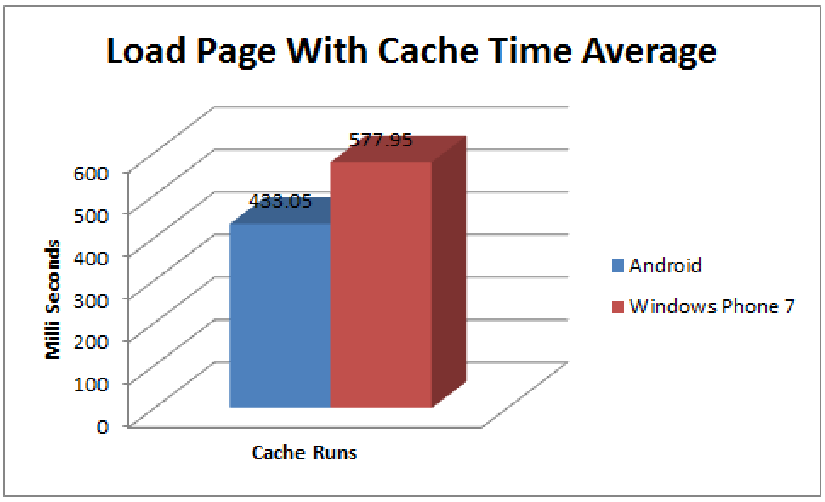


Figure 5.23: Load Page Using Cache Average Time

The CPU and memory usage were recorded for the cache task without using the cache. This was to show how loading a page without using the device's cache memory would impact on the device's resources. Fig 5.24, shows the average CPU usage required to perform loading the page while using the cache.

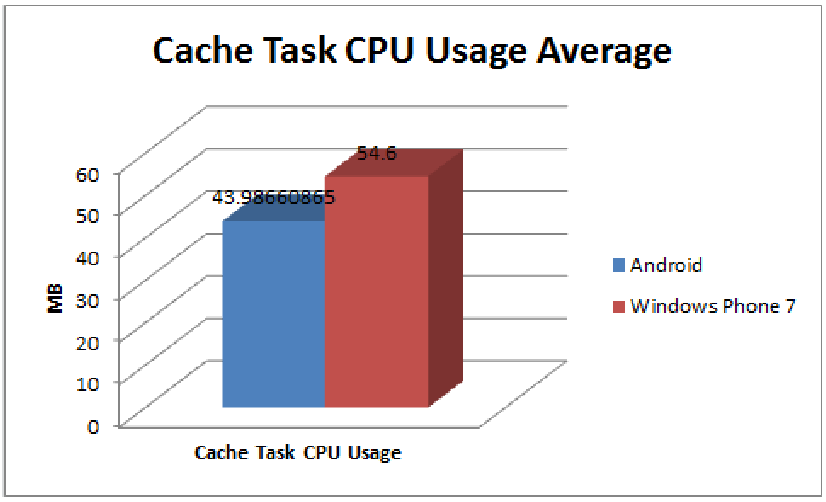


Figure 5.24: Cache Task CPU Usage

Fig 5.25, represents the average memory required by both versions of the application to load the page while using the cache.

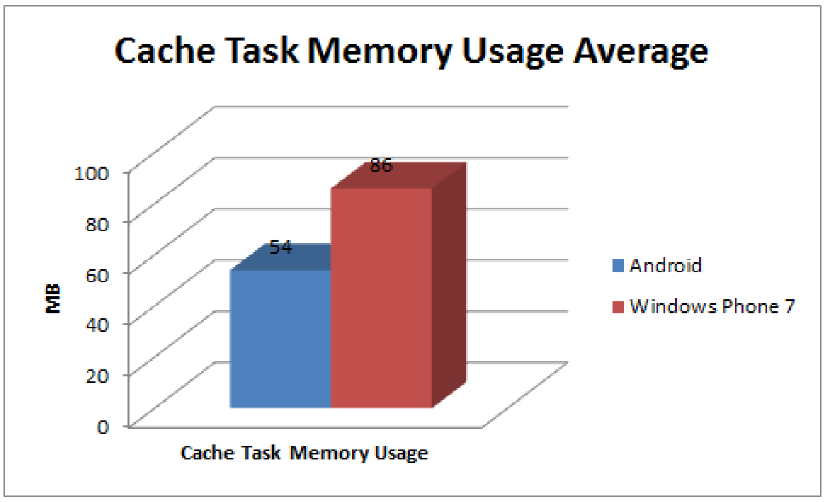


Figure 5.25: Cache Task Memory Usage

Fig 5.24 shows that there was, on average, a ten percent difference in CPU usage between Android and Windows Phone 7. On average, Android loaded the page from the cache memory using 43.98660865% of the CPU usage. Windows Phone 7 required 54.6% of the CPU usage. Android required less resources and less time to load the page from the cache memory. This results in the Android operating system being better suited for cache memory applications.

5.7 Conclusion

Five tasks were carried out on the Android and Windows Phone 7 operating systems: threads, file, load , buffer, and cache. Each of these tasks were designed to test the performance of both operating systems. While the tasks were run, the response times, CPU usage and memory usage were recorded. The results of the tests were used to determine which operating system is best suited to a number of tasks, e.g. which would be best for database use, etc. With the thread, file and load task Windows Phone 7 showed better performance and response times. Windows Phone 7 also had the best response time when loading the web page from the Apache server. The

Android operating system took longer to respond to the user's request to play the MP3 file, in spite of this, it buffered the file noticeably faster than Windows Phone 7. Android also loaded the web page from the cache memory faster than Windows Phone 7.

6 Conclusions

6.1 Introduction

This chapter reviews the results and findings of this thesis, and discusses further work and research that could be conducted.

6.2 Limited Operating System Performance

The objective of this thesis was to compare the performance of an application on limited operating systems. The purpose of this was to determine which operating system performed best while completing a selection of tasks. These tasks were:

- spawn threads
- read and write to a file saved in internal memory
- added records to a database and query them back
- buffer and play back a MP3 file stored on-line
- load a web page from a server and from cache memory

These five tasks were selected as they test the various performance aspects of a limited operating system. For this thesis two operating systems were selected: Android and Windows Phone 7. These operating systems were chosen as they are both available on an assortment of Smartphones, and are among the most popular operating systems on the market.

The PerformanceApp software was developed to ascertain which of the tested operating systems performed best under the previously mentioned tasks. This software measured the response times, CPU usage and memory usage for each task. The response time used Equation 2.1 to calculate the time taken to complete each task. Chapter 3 details how the CPU usage and memory usage were gathered for the PerformanceApp software. After carrying out research on the performance of the Android and Windows Phone 7 operating systems, it is evident that more research needs to be carried out, especially for Windows Phone 7 as it is such a new operating system.

Chapter 5 details the results of testing the performance of both operating systems. In terms of response times the Windows Phone 7 operating system was superior overall with the thread, file, and load tasks, as well as loading a web page without using the cache. The only task where Android had the better response time was with loading a web page from the system's cache. Even at that, the difference was a matter of milli seconds. The chapter also shows that in every case, except loading the web page without the cache, Android used considerably less of the CPU than Windows Phone 7. The memory usage results showed that Windows Phone 7 used less memory for every task except the cache task.

An explanation for why the CPU usage was so low for Android is that Android is the only limited operating system that has true multitasking and is therefore better set up to manage loads on the system. Windows Phone 7 uses a method called 'tomb-stoning' in order to fake multitasking where the applications state is saved when closed. The saved state is loaded up when the application is reopened give the illusion of multitasking. From the results the author believes that Windows Phone 7 was able to achieve its better response times because of this. Since the operating system is not really multitasking, it has the ability to dedicate more resources to getting tasks done and can therefore achieved better response times.

Analysis of the results revealed that they could be determined in multiple ways. As stated in Chapter 2 what the developer believes to be good performance and what the end user believes to be good performance can be two completely separate things. Depending on the users perspective they may believe that response times alone are what determines good performance. In the authors opinion a combination of all the three metrics recorded is the best way of determining good performance. In most cases a developer looks to the CPU and memory usage of a system to determine if it has good performance or not. Whereas the vast majority of end users will see response times as the marker for performance. As the results gathered from this thesis have shown, Windows Phone 7 is miles ahead of Android in terms of response times and memory usage in most cases. Android has shown much better results for the CPU usage. The developers would most likely chose Android as having the best performance and an end user would chose Windows Phone 7.

Based on the results gathered for this thesis, it would be recommended to use the Windows Phone 7 operating system for applications where the main features will include: the use of multiple threads (e.g. running functionality on a background thread), parsing a large amount of data to another data type, reading and writing to files in internal storage, employing the use of an internal database, and loading web page from servers. It would be recommended to use the Android operating system for applications which will concentrate on loading from the device's cache memory, and streaming media from on-line sources. It would not be advisable to use the Windows

Phone 7 operating system for resource heavy applications as they are not as adept as handling CPU usage loads as the Android operating system.

6.3 Further Work

The Android and Windows Phone 7 mobile operating systems performance were compared for this thesis. It would have been desirable to compare the performance of all the major operating systems available. However, for the scale of this thesis it was not possible. Further research could be carried out in this area, building on from this thesis. The research could be expanded by developing a version of the PerformanceApp software for other mobile operating systems including: Apple's iOS, Blackberry OS, and possibly Symbian. Conducting research on Symbian may not be as useful as the others as Symbian is a dying operating system Ricker (2011).

As mentioned in Chapter 2, Lee and Lee (2011) carried out research where two applications were developed for Android. One was written using the Android SDK and Java, and the other was written using Android NDK and Native C. Similar to this thesis, their application tested performance areas of the Android operating system in order to determine which language Android applications should be written in. An interesting area for further research would be to combine the research of this thesis with the research carried out by Lee and Lee (2011). This could involve developing an application similar to the PerformanceApp software, but it could be written for all the major operating systems available, as well as writing versions in different languages, e.g. Java and Native C for Android, Silverlight and XNA for Windows Phone 7. This would give a definitive answer to which mobile, or limited, operating system provides the best performance.

6.4 Conclusion

Smartphones and mobile devices are a standard feature in life today. They are used in most aspects of life, from work, to leisure, and even how people interact with others. With such a vast assortment of mobile operating systems available, distinguishing which one to chose can be a daunting task. Knowing which operating system performs best at certain tasks can greatly aid in the selection process. In the pursuit of finding which mobile operating system provides the best performance, this thesis has determined that one operating system cannot be definitively described as having the best performance. As this thesis has shown, different operating systems perform better in some areas than others. Therefore when choosing a mobile operating system, or Smartphone, the question asked should be 'what will this device be main used for?'. Based on the results obtained during this thesis, the author would recommend the Windows Phone 7 operating system be used for business use, and the Android operating system be used for more social-able use. The reason being that a Smartphone being used for business will be used for more critical tasks where getting tasks completed sooner rather than later is vital.

References

Android. What is android?, 2012a. URL http://developer.android.com/guide/basics/what-is-android.html.

Android. What is the ndk?, 2012b. URL http://developer.android.com/tools/sdk/ndk/overview.html.

Android. Platform versions, June 2012c. URL http://developer.android.com/about/dashboards/index.html.

Thiam Kian Chiew. *Web Page Performance Analysis*. PhD thesis, University of Glasgow, March 2009.

Windows Phone Interoperability. Chapter 1: Windows phone 7 platform introduced to iphone application developers, 2011. URL http://windowsphone.interoperabilitybridges.com/articles/chapter-1-windows-phone-7-platform-introduced-to-iphone-application-developers.

Stacy Joines, Ruth Willenborg, and Ken Hygh. *Performance Analysis for Java Web Sites*. Pearson Eduacation Inc, 2003.

Heejin Kim, Byoungju Choi, and W. Eric Wong. Performancetesting of mobile applications at the unit test level. *IEEE*, July:171 – 180, July 2009.

G. Kotsis. *Web Engineering: the discipline of systematic development of web applications*, chapter Performance of Web Applications, pages 247–264. John Wiley and Sons, 2006.

Tapas Kumar Kundu and Prof Kolin Paul. Improving android performance and energy efficiency. *IEEE*, January:256 – 261, January 2011.

Jae Kyu Lee and Jong Yeol Lee. Android programming techniques for improving performance. *IEEE*, September:386 – 389, September 2011.

D.J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge: Cambridge University Press, 2000.

Microsoft. Windows phone performance analysis, 2012. URL `http://msdn.microsoft.com/en-us/library/hh202934%28v=vs.92%29.aspx`.

Jason Parekh, Gueyoung Jung, Galen Swint, Calton Pu, and Akhil Sahai. Comparison of performance analysis approaches for bottleneck detection in multi-tier enterprise applications. *IEEE*, June:302 – 303, June 2006.

Thomas Ricker. Rip symbian, 2011. URL `http://www.engadget.com/2011/02/11/rip-symbian/`.

Marian K. Riedy, Suman Beros, and H. Joseph Wen. Managing business smartphone data. *Journal of Internet Law*, March:3–14, 2011.

Weiqi Song, Tao Tao, and Tiegang Gao. Performance optimization for flash memory database in mobile embedded systems. *IEEE*, March:35–39, 2010.

Vladimir Stantchev and Miroslaw Malek. Addressing web service performance by replication at the operating system level. *IEEE*, June:696 – 701, 2008.

Chris Ziegler. Microsoft talks windows 7 series development ahead of gdc, 2010. URL `http://www.engadget.com/2010/03/04/microsoft-talks-windows-phone-7-series-development-ahead-of-gdc/`.

Bibliography

Android. Designing for Performance. Android, http://developer.android.com/guide/practices/design/performance.html. 2011.

Android. Tracview. Android, http://developer.android.com/guide/developing/debugging/debugging-tracing.html. 2011.

Apple. iOS Technology Overview. Apple, http://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html. 2011.

David Booth and Hugo Haas and Francis McCabe and Eric Newcomer and Michael Champion and Chris Ferris and David Orchard. Web Services Architecture. W3C, http://www.w3.org/TR/ws-arch/wsa.pdf. February 2004.

Valeria Cardellini and Emiliano Casalicchio and Michele Colajanni. A Performance Study of Distributed Architectures for the Quality of Web Services. IEEE, January 2001.

Martin Gudgin and Marc Hadley and Noah Mendelsohn and Jean-Jacques Moreau and Henrik Frystyk Nielsen and Anish Karmarkar and Yves Lafon. SOAP: Messaging Framework. W3C, http://www.w3.org/TR/soap12-part1/. 2007.

R.J. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling. John Wiley and

Sons, 1991.

Guangzhu Jiang and Shujuan Jiang. A Quick Testing Model of Web Performance Based on Testing Flow and its Application. IEEE, pages 57-61, September 2009.

Michael J. Johnson and E. Michael Maximillien and Chih-Wei Ho and Laurie Williams. Incorporating Performance Testing in Test-Driven Development. IEEE, pages 67-73, Volume 24, Issue 3, June 2007.

Damian Koh. Microsoft on Windows Phone 7 Series. CNet, `http://asia.cnet.com/qanda-microsoft-on-windows-phone-7-series-62061278.htm`. 2010.

Changbao Li and Bo Cheng and Junliang Chen and Pingli Gu and Na Deng and Desheng Li. A Web Service Performance Evaluation Approach Based on Users Experience. IEEE, pages 734-735, July 2011.

Ravi Mandalia. Pentagon OKs Android for DoD Usage. ITProPortal, `http://www.itproportal.com/2011/12/26/pentagon-oks-android-dod-usage/`. 2011.

Carolyn McGregor and Josef Schiefer. A Framework for Analyzing and Measuring Business Performance with Web Services. IEEE, pages 405-412, June 2003.

Microsoft. Application Platform Overview for Windows Phone. Microsoft, `http://msdn.microsoft.com/en-us/library/ff402531%28v=vs.92%29.aspx`. 2012.

Microsoft. Unit Test A Windows Phone Application. Microsoft, `http://msdn.microsoft.com/en-us/hh849671.aspx`. 2012.

Ryan Paul. A Developer's Introduction to Google Android, `http://arstechnica.com/gadgets/2009/02/an-introduction-to-google-android-for-developers/`. 2009.

Christy Pettey and Laurence Goasduff. Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent. Gartner Newsroom, `http://www.gartner.com/it/page.jsp?id=1764714`. 2011.

Web Service Tutorials. Web Service to Web Service Communication. Web Service Tutorials, `http://www.wstutorial.com/web-service-to-web-service-communication/`. 2011.

Jingmin Xie and Xiaojun Ye and Bin Li and Feng Xie. A Configurable Web Service Performance Testing Framework. IEEE, pages 312-319, September 2008.