COMPARING PERFORMANCE OF APPLICATIONS WRITTEN IN
SMARTPHONE DEVELOPMENT ENVIRONMENTS

A Thesis
Presented to the
Faculty of
San Diego State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
in
Computer Science

by
Yashodhar Rajnikant Patel

Fall 2011
SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Yashodhar Rajnikant Patel:

Comparing Performance of Applications Written in Smartphone Development Environments

Joseph Lewis, Chair
Department of Computer Science

Roger Whitney
Department of Computer Science

Robert Grone
Department of Mathematics and Statistics

11-01-2011
Approval Date
DEDICATION

I dedicate this thesis work to my dear parents, for his constant encouragement and their willing to provide unconditional efforts every time I needed.
ABSTRACT OF THE THESIS

Comparing Performance of Applications Written in Smartphone Development Environments
by
Yashodhar Patel
Master of Science in Computer Science
San Diego State University, 2011

There are basically three smartphone application development environments to build an application: (1) Software development kit provided by native platforms; (2) Web frameworks created over the native SDK which supports multiple platforms; (3) HTML5 with JavaScript frameworks and CSS3.

Recently, companies like PhoneGap, Appcelerator and MotherApp have started to release the web frameworks created for smartphone application development environments. Many Web developers can take advantage of these frameworks and build their application once and use it in many platforms like Android, Windows Mobile, iOS, PalmOS etc. This allows them to develop an application in their native environment without learning SDKs of these platforms.

This purpose of this thesis project is to compare the performance of applications written over these three development environments for the Android platform. Applications can be compared in three basic areas: algorithmic speed, graphical performance and database performance. Applications focusing on each of these areas were written to test the performance of the different development environments. The results of the tests will give an idea to choose which environment to be used based on the requirements of an application which can be either be cost effectiveness and ease of development or performance.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
</tr>
<tr>
<td>1 INTRODUCTION</td>
</tr>
<tr>
<td>1.1 Background</td>
</tr>
<tr>
<td>1.2 Basic Difference between Three Environments</td>
</tr>
<tr>
<td>1.3 Phone Application Background and Their Relation with Frameworks</td>
</tr>
<tr>
<td>1.4 Limitation of Study</td>
</tr>
<tr>
<td>2 SMARTPHONE HISTORY, APPLICATION MARKET TREND AND INTRODUCTION TO WEB FRAMEWORKS</td>
</tr>
<tr>
<td>2.1 Why Smartphones Are Gaining Popularity?</td>
</tr>
<tr>
<td>2.2 About PhoneGap</td>
</tr>
<tr>
<td>2.3 Pros of the Web Frameworks</td>
</tr>
<tr>
<td>2.4 Cons of the Web Frameworks</td>
</tr>
<tr>
<td>2.5 Why Development with Native SDK is Painful?</td>
</tr>
<tr>
<td>3 TOOLS USED IN THREE DIFFERENT APPLICATION ENVIRONMENTS</td>
</tr>
<tr>
<td>3.1 Native Software Development Kit Features</td>
</tr>
<tr>
<td>3.1.1 Android Architecture</td>
</tr>
<tr>
<td>3.1.2 Android Development Tools (ADT)</td>
</tr>
<tr>
<td>3.1.3 Dalvik Debug Monitor Server (DDMS)</td>
</tr>
<tr>
<td>3.2 Web Frameworks</td>
</tr>
<tr>
<td>3.3 HTML5 with CSS3 and JavaScript</td>
</tr>
<tr>
<td>3.3.1 Steps to Write an Application Using HTML, CSS, jQuery</td>
</tr>
<tr>
<td>3.3.2 Client-Side Storage for Web Frameworks and HTML5</td>
</tr>
<tr>
<td>3.3.2.1 How is Client-Side Storage Achieved?</td>
</tr>
</tbody>
</table>
3.3.2.2 Session Storage ................................................................. 17
3.3.2.3 Local Storage ................................................................. 17
3.3.2.4 Database Storage ........................................................... 18

4 COMPARISON TYPES AND APPROACH ................................................. 19
  4.1 Device Used for the Test Application .............................................. 19
  4.1 Execution Speed Comparison ....................................................... 19
    4.1.1 Fibonacci Algorithm Implementation for Native SDK Applications .... 20
    4.1.2 Fibonacci Algorithm Implementation for HTML5 and PhoneGap Applications .......................................................... 22
  4.2 Graphical Function Execution Speed Comparison ............................. 22
    4.2.1 Graphical Test for Native SDK Applications .............................. 23
    4.2.2 Graphical Test for HTML5 and PhoneGap Applications ................ 24
  4.3 Database Performance Comparison ................................................. 25
    4.3.1 Database Test for the Native SDK Applications ......................... 26
    4.3.2 Database Test for the PhoneGap and HTML5 Applications ............ 26

5 COMPARISON AND ANALYSIS ............................................................ 28
  5.1 Framework Comparison Analysis .................................................. 28
  5.2 With Respect to Exponential Algorithm ........................................... 28
  5.3 With Respect to Graphical Functions .............................................. 30
  5.4 With Respect to Database Performance .......................................... 32

6 MATRIX OF APPLICATIONS COMPARISON AND CONCLUSION ............... 35
  6.1 Matrix of Applications Comparison .............................................. 35
  6.2 Future Enhancements ............................................................... 36

BIBLIOGRAPHY .................................................................................. 37

APPENDIX
A FIBONACCI TEST APPLICATION CODE IN NATIVE SDK ................... 38
B FIBONACCI TEST APPLICATION CODE IN HTML5 ............................ 41
C FIBONACCI TEST APPLICATION CODE IN PHONEGAP ...................... 43
D EVENT DISPATCH THREAD AND THREAD POOL ................................ 45
E GRAPHICS TEST APPLICATION CODE IN NATIVE SDK ..................... 48
F GRAPHICS TEST APPLICATION CODE IN PHONEGAP .......................... 52
G GRAPHICS TEST APPLICATION CODE IN HTML5 .............................. 55
H  DATABASE PERFORMANCE TEST APPLICATION CODE IN NATIVE SDK
..........................................................................................................................58
I  OPEN DATABASE METHOD TO INITIALIZE THE DATABASE .......................63
J  DATABASE PERFORMANCE TEST APPLICATION CODE IN PHONEGAP ..........................................................................................................65
K  DATABASE PERFORMANCE TEST APPLICATION CODE IN HTML5 ........68
LIST OF TABLES

Table 2.1. Application Market For Phone Applications ............................................................5
Table 4.1. Phone Specification Used for the Test Applications ..............................................19
Table 5.1. Algorithm Comparison between Environments (Android 2.3 HTC Sensation) .....................................................................................................................29
Table 5.2. Graphical Execution Time per 5 Cycles (Android 2.3 HTC Sensation) ...............31
Table 5.3. Graphical Execution Frame per Time.....................................................................32
Table 5.4 Database execution time taken by update query (Android 2.3 HTC Sensation) .....................................................................................................................33
Table 6.1. Application Comparison Matrix .............................................................................35
LIST OF FIGURES

Figure 3.1. Android architecture .......................... 11
Figure 3.2. DDMS monitoring .......................... 13
Figure 4.1. Fibonacci call tree .......................... 21
Figure 5.1. Algorithm performance graph (Android 2.3 HTC Sensation) .......................... 31
Figure 5.2. Graphical execution comparison graph .......................... 32
Figure 5.3. Database execution comparison graph .......................... 34
Figure D.1. Event dispatch thread .......................... 46
ACKNOWLEDGEMENTS

I take this opportunity to sincerely thank my thesis advisor, Dr. Joseph Lewis, for the Guidance, unrelenting support, and backing me during every step of my research work. Without his help it would not have been possible for me to accomplish this feat. I would also like to thank Professor Roger Whitney and Professor Grone for being the readers of this thesis and for giving their valuable suggestions.
CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The field of communication is evolving now like it was never before. With the introduction of smartphones, people can now be connected, not only to their family and friends, but to the entire globe. This has brought about an ever increasing demand for the mobile devices and the current age is witnessing an explosive growth rate in the sales of smartphones. As the demand for these devices is rising, so is the demand for applications which run on these devices. As such, not only existing companies, but many new companies have come up trying to wash their hands with mobile application development.

As is the case with any software development platform, mobile application development too has variety of available languages with their supported frameworks. Having multiple frameworks and languages available for development, it is very important to consider the trade-offs associated with each of them, and then choose the most suitable framework as per the business requirement. Making the right decision not only helps in improving the overall quality of the application, but at the same time helps to reduce the cost.

To choose a suitable framework, developer needs to understand the basic idea of how different framework works and the features it supports. Since, phone application development is fairly new as compared to website application development or desktop application development, most developers are currently naïve about different available frameworks in this field and trade-offs associated with each of them. With the general trend, where in developers focus more on the application requirement rather than learning a new framework, the Aim of this thesis is to provide some insight into available mobile application development frameworks supported by some concrete results, which will help developers in choosing the appropriate framework for their phone application.

Currently, there are three major framework environments available to develop the phone application which includes Native software development kit (SDK), Basic web languages (HTML5, CSS3 and JavaScript) and Web frameworks created for mobile
applications (PhoneGap, Appcelerator, MotherApp, etc.). Before phone application developers decide to work with any one of these environments, it is important for them to consider several factors which include ease of development, easiness to learn the framework, cost of development, cross platform capability, experience with framework, features of framework, etc. A little research to choose the framework environment based on the requirements of the application can help a lot in future for further enhancements and possibilities of growing the application users. Thus the main focus of this thesis is to compare different frameworks based on the above mentioned criteria. Test results created by this study can be used by developers to have general idea about available options supported by different frameworks.

1.2 Basic Difference Between Three Environments

Native software development kit has complete access to phone hardware which includes camera, vibrator, gravity sensor, speaker, geo location support, Wi-Fi support, finger print scanner, multi-touch support, file system IO support and database support. It also provides powerful API support which makes the task easy for developer, thus almost any type of complicated application can be built on top of this SDK. On the other hand, web languages like HTML5, CSS3 and JavaScript have restrictive access to the phone hardware. Web languages cannot communicate to phone hardware directly, but this does not imply that the applications developed using these languages cannot take advantage of the features provided by phone hardware. Web frameworks like PhoneGap, allows you to use phone hardware features and also lets you use the web languages. Though, this framework cannot provide as much power as SDK can do to applications developed on them.

1.3 Phone Application Background and Their Relation with Frameworks

Communications (e-mail clients, IM clients, news clients, etc.), games (puzzles, strategy, sports, etc.), multimedia (presentation viewer, audio players, video players, etc.), productivity (notepad, word processors, spreadsheets, banking apps, etc.), travel (city guides, currency converters, translators, GPS, maps, etc.), and utilities (profile manager, task manager, call manager, file manager, etc.) are the major categories of mobile applications.
Some of these applications require lot of features that only native SDK can support. For example, Task Manager mostly requires native SDK functional support. So in such cases developers does not have a choice other than to use the SDK framework. On the other hand, most of the applications do not require all the native SDK support. For example, Social network clients, E-mail clients, Calendars, Banking apps, small games, etc. These applications can be very easily built without learning how native SDK languages. In building these applications, this study results will help to focus on the application requirements.

Writing basic applications in each of these available frameworks is the simplest way of comparing how these frameworks perform. Thus in this study, basic phone applications has been developed in all the available frameworks and has been tested on algorithmic performance, graphical performance and database performance. Algorithmic performance has been tested by executing an exponential Fibonacci algorithm which uses more execution and memory power. To compare the graphical performance, test applications have been written which performs a simple graphical function of moving an object on the screen. Graphical performance is an important aspect for most applications and since web languages uses browser and native SDK uses its own graphical interface, it was important to test and compare this aspect. Finally, database performance is tested by running update query in the database for number of times. Web languages uses database within the browser. While native SDK uses database within its own framework. It is hard to find out how it handles the database operations but easy just to perform them and see the difference.

1.4 LIMITATION OF STUDY

This study does not compare each and every features of each environment in available frameworks. It does not describe everything about these frameworks and their capabilities in writing applications. Test results in this study have been performed in simulation environment which means these results cannot be directly used to determine how fast real time application will perform.

Limiting this comparison allows more focused study. Choosing these three criteria of comparison allows more encompassing comparison. Comparing more features of frameworks creates more complicated results. Generalized study is enough for basic need of choosing a framework.
CHAPTER 2

SMARTPHONE HISTORY, APPLICATION MARKET TREND AND INTRODUCTION TO WEB FRAMEWORKS

In 1992, IBM introduced the first Smartphone in history. The main features of this device were a calculator, an address book, an email, games, notepad and fax. Later on the first BlackBerry device was introduced in 1999 which has two-way pager functionality. In 2002, another new smartphone (BlackBerry) was released which had Web browsing, text messaging, Internet faxing, email and other wireless information services. This is also an example of telecommunication device which has calling capability [1].

The Nokia Communicator series was the first of Nokia’s smartphones. Nokia 9000 was the first device in this series released in 1996. This device has distinctive palmtop computer style. It was a result of personal digital assistant (PDA) developed by Hewlett-Packard combined with Nokia’s best-selling phone. This device was featured with QWERTY keyboard, high-resolution display of at least 640 x 200 and PDA user interface under the fliptop. The software was based on the GEOS V3.0 operating system, featuring email communication and text-based web browsing. In 1998, it was followed by Nokia 9110, and in 2000 by Nokia 9110i, with improved web browsing capability [1].

BlackBerry gained popularity by bringing various models with different price ranges and features such as Bold, Pearl, Curve and Storm. In 2004, Palm Treo released 320 x 320 pixel screen with Bluetooth support and data speeds up to EDGE/CDMA standards. In 2007, Apple changed the smartphone landscape overnight with the release of new iPhone. Though the first generation was lacking 3G capabilities and the only way to deliver apps was via Web, the Webkit-powered Mobile Safari set a very high bar of the future of the mobile Web. In 2008, iPhone 3G (higher-speed UMTS data connectivity) was released and it became widespread in the US. Apple also launched the App Store which gained significant amount of application power for iPhones. In same duration, HTC built one of the first Android-powered
smartphones. The device launched in the US as the T-Mobile G1, the first “Google-powered” smartphone [1].

Smartphone has become mass market mobile computing platform. Bizjournals has published an article about smartphone applications gaining popularity in health care [2]. According to this article, in survey reported by Pyramid Research, there are 200 million applications being used by doctors and patients and more than 600 million applications. Smartphone applications are gaining popularity in Auto Dealer market. Indian Telecom Online have released an article [3] last year, which shows that only in India, nearly 7.5 million smartphones have been sold to the customers with the growth of 30% compared to last year. The popularity of the smartphones is mainly due to the drop in the prices of handsets, rise in social networking sites and aggressive marketing by the handset manufacturers. The smartphones now provide ultimate convergence of music players, digital cameras, radios, gaming and navigation devices. See Table 2.1 for number of apps available and downloaded from App Store [4].

Table 2.1. Application Market For Phone Applications

<table>
<thead>
<tr>
<th>Date</th>
<th>Available Apps</th>
<th>Downloads to Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>April 8, 2010</td>
<td>185,000+</td>
<td>4,000,000,000+</td>
</tr>
<tr>
<td>April 29, 2010</td>
<td>200,000+</td>
<td>4,500,000,000+</td>
</tr>
<tr>
<td>June 7, 2010</td>
<td>225,000+</td>
<td>5,000,000,000+</td>
</tr>
<tr>
<td>September 1, 2010</td>
<td>250,000+</td>
<td>6,500,000,000+</td>
</tr>
<tr>
<td>October 20, 2010</td>
<td>300,000+</td>
<td>7,000,000,000+</td>
</tr>
<tr>
<td>Jan 16, 2011</td>
<td>300,000+</td>
<td>9,000,000,000+</td>
</tr>
<tr>
<td>Jan 22, 2011</td>
<td>350,000+</td>
<td>10,000,000,000+</td>
</tr>
</tbody>
</table>


This numbers shows that every month thousands of applications have been created. This large requirement of applications in the market has created requirement of smartphone application developer. Tim Heard from Techrepublic have talked about this in article released in November 2010 [5]. He has mentioned that there is a huge opportunity in mobile application market if any software developer can find a combination of good product, timing and publicity. He also said that a growing market is the business application. While there are
some good apps that have been developed for smartphones, this is clearly a market with lots of growth potential, especially as smartphone functionality continues to evolve. In the later discussion, he mentioned that this high requirement rate of developer will come from the existing pool of developers. Some companies like PhoneGap, Appcelerator and MotherApp have started to build web framework which can run over the native SDK and lets the web developer use web languages to build application.

Mobile Beat has released one article [6] in which Matt says that HTML5 movement has so much momentum that it can defeat Native App in less than two years. He also says that web publisher companies can develop it once and distribute it across any device via Internet browser. HTML5 will save billions of development costs which will allow publishers to direct those savings towards more innovative and productive projects. Native apps are still extremely popular for many developers because HTML5 is still working to close the performance gap. HTML5 graphics performance for fast-moving games that have a lot of animation can’t match native’s performance, and probably won’t for some time to come. But for pretty much anything else, HTML5 is good enough, an increasing number of developers are saying. The benefits gained from a slightly faster native experience will be so marginal for the vast majority of apps in game area.

2.1 WHY SMARTPHONES ARE GAINING POPULARITY?

Smartphones are the gadgets of this new technologically advanced generation. They rule the way we talk, we do business and we get ourselves entertained. Smartphone features and applications make life easier and simple. The operating system in these smartphones makes it possible to use different kinds of applications. The absolute reason for the rising popularity of these smartphones is the support for social networking and file sharing, they provide through a wide range of applications.

Also, the features are simple and easy to use. It is the operating system and apps that makes a huge difference. The purpose of the smartphone is not just to call or text someone; it has become handy due to available apps. For ex. Ebay provides communication between buyers and sellers. Ebay website has lots of functionalities. Seller can set the product Web Pages through computer interface. Buyer can browse through the products and select the appropriate one. Now if we imagine a smaller version of this application in which main tasks
of the applications can be considered related to buyers and sellers. Seller wants to monitor their sells and buyer wants to search the product and buy it. Smaller version of this large Ebay application can be built for smartphone. This idea of creating a small application for smartphone has now taken a huge part of smartphone application development market. The various applications provide various features which make this device multipurpose. Recently a stethoscope iPhone application made iPhone a doctor’s favorite possession by turning the gadget into a far superior stethoscope which has got advanced functionality than a regular one. These are the reasons why smartphone applications are being required in the market for every small or large business. Floorbiz.com has released an article describing about smartphone application popularity for carpet companies like Shaw, Mannington and Laticrete. Shaw’s Carpet Stain Center provides stain removal procedures for most common household stains. It shows step by step cleaning instructions from the mill are free for download from the iPhone app store and is compatible with the new iPad. Mannington’s app is designed to serve designers’ process. Using a smartphone’s camera and scanning technology, the app integrates sample books to show coordinating collections and enable sample orders. By scanning the bar code printed on a Mannington sample book, a designer can immediately see coordinating patterns and colors across Mannington’s carpet and hard surface collections. Technology is compatible with iPhone, Android and Blackberry smart mobile phones. Laticrete’s free product information app is a full-service, jobsite reference, tool kit that offers quick and easy retrieval of the company’s product information from any iPhone, iPad or iPod, explained Henry Rothberg, co-owner.

2.2 ABOUT PHONEGAP

PhoneGap is an open-source mobile development framework developed by Nitobi Software. It enables software programmers to build applications for mobile devices using JavaScript, HTML and CSS, instead of less-popular languages such as Objective-C. PhoneGap currently supports development for the iPhone, Palm webOS, Google Android, Windows Mobile, Symbian OS and BlackBerry operating systems (support for recent versions, such as BlackBerry 5 and 6 and Windows Phone 7, is being implemented now). Bada (operating system; used by the new Samsung Wave) is not supported. This framework allows developer to access hardware features of phone via APIs.
2.3 Pros of the Web Frameworks

Main advantages of using web frameworks are listed below.

1. Multi-Platform support has been provided by these frameworks.
   An application written in one platform can be used on other platforms. Web framework provides one API which has different implementation in different platforms. While writing an application, developer does not have to worry about which platform it is going to run.

2. Web developers can simply get all the features that one normally uses in Native Framework without learning it.
   Most of the phone hardware features required by the phone application are supported by the API. This API can be used without learning the native framework SDK including its basic life cycle.

3. Less configuration settings are required.
   Each native framework requires configuration settings to run the application. For example, android requires all the activities listed in the configuration file.

4. It reduces the development time by supporting all frameworks.

2.4 Cons of the Web Frameworks

Some disadvantages of using web frameworks are listed below.

1. There may be a case of non-availability of all the features.
   For example, PhoneGap does not expose native UI components to JavaScript API. These frameworks cannot support all features because features differ in different platforms. It is possible that one feature is available in android but not in iPhone.

2. It is not possible for a developer to modify the behavior of a function implemented by native framework.
   It is hard to create a new feature since developers do not have direct access to Native APIs. Let us take an instance from the Appcelerator problem: One developer was trying to change the Tab name dynamically but code was not working with framework.

   Someone from the Appcelerator answered that it is not possible. Developers can only use API given from Appcelerator from which they can perform certain tasks. Developers expect to change the functional behavior by modifying or extending existing classes that is expected with Native code but from these frameworks it is difficult to access the Native SDK classes and modify their behavior by extending them from JavaScript.)
   Native frameworks are created to support applications for the particular platform while web frameworks might have issues with integration.

4. Harder to debug and find the issue in application.
   For example, PhoneGap does not throw exceptions (May be available in future.) like Native SDK which makes it hard to trace bugs and debug the code while development and testing applications.

2.5 Why Development with Native SDK is Painful?

It is simpler and less expensive to create a Smartphone application than creating applications for personal computers. As of now, almost all the applications are built using the native SDK for each mobile platform. This usually requires a developer to learn a new language like Objective C for the iPhone applications, Java for Android phone applications or C# for Windows Mobile, etc. Also, it asks a developer to learn a new Application Programming Interface (API). Though, Smartphones share a majority of features, but still their SDKs have some very significant differences and some quite subtle.

It is a real challenge is to stay competitive, to create an application that works on all the major mobile platforms, iPhone, which has a large and growing market, Android, which is coming on strong, or, be it be RIM (BlackBerry) is maintaining its corporate presence.

To create an application to make it available on multiple platforms, will take lots of efforts in writing the same login in multiple programming language. Also developer needs to learn using multiple SDKs which will require extra development time, complexity and cost. This will multiply the possibility of introducing bugs in the code, thus multiplying the amount of testing needed. Maintenance of such an application will ask the developer to maintain separate lines of code in the different programming languages.
CHAPTER 3

TOOLS USED IN THREE DIFFERENT APPLICATION ENVIRONMENTS

This chapter explains tools, frameworks and their introduction used to compare the performance of Native Software Development Kit versus Web Frameworks customized for Smartphone applications which supports multiple platforms versus HTML5 with JavaScript.

3.1 NATIVE SOFTWARE DEVELOPMENT KIT FEATURES

Native SDK provides lot of features and libraries to develop native application. Some of the features are listed below.

- Application framework enabling reuse and replacement of components
- Dalvik virtual machine optimized for mobile devices
- Integrated browser based on the open source WebKit engine
- Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- SQLite for structured data storage
- Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- GSM Telephony (hardware dependent)
- Bluetooth, EDGE, 3G, and WiFi (hardware dependent)
- Camera, GPS, compass, and accelerometer (hardware dependent)
- Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE

3.1.1 Android Architecture

Figure 3.1 shows the major components of the Android operating system. Each section is described in more detail below.

Android software development kit can be used with the Eclipse IDE and android plugins like ADT and DDMS.
3.1.2 Android Development Tools (ADT)

Android Development Tools (ADT) can be installed as an Eclipse IDE plugin which is designed to support easy integrated environment with Eclipse interface to build Android applications [7].

ADT expands the capabilities of Eclipse so that you can quickly setup the new Android projects. Also it allows you to create Application UI and build the UI components based on the supported Android Framework API. You can also export the signed or unsigned apk files in order to distribute your applications.

ADT allows fastest way to start developing an application and it is also recommended by Google. It provides custom XML editors, SDK integration, and project setup UI
components and debug output pane. This plugin is very useful to boost the development process.

ADT plugin provides GUIs and wizards to create Android projects and Library projects:

- An Android project contains all of the files and resources that are needed to build a project into an apk file for installation. You need to create an Android project for any application that you want to eventually install on a device.

- You can also designate an Android project as a library project, which allows it to be shared with other projects that depend on it. Once an Android project is designated as a library project, it cannot be installed onto a device.

### 3.1.3 Dalvik Debug Monitor Server (DDMS)

The debugging tool for Android SDK is known as the Dalvik Debug Monitor Server (DDMS), it provides port-forwarding services, thread and heap information on the device, logcat, process, screen capture on the device, radio state information, location data spoofing, incoming call and SMS spoofing, and more [8].

Every application on Android runs in its own process, and each process runs in its own virtual machine (VM). Each VM exposes a unique port that a debugger can attach to.

DDMS connects to Android Debug Bridge (adb) as soon as it starts. Then, VM monitoring service is created between adb and DDMS when a device is connected. This service notifies DDMS about when a VM on the device is started or terminated. Once a VM is running, DDMS retrieves the VM's process ID (pid), via adb, and opens a connection to the VM's debugger, through the adb daemon (adbd) on the device. Now using a custom wire protocol, DDMS can talk to the VM.

Debugging port is assigned to each VM on the device by the DDMS. Typically, for the first debuggable VM, DDMS assigns port 8600, the next on 8601, and so on. All the traffic is forwarded to the debugger from the associated VM when a debugger connects to one of these ports. You can only attach a single debugger to a single port, but DDMS can handle multiple, attached debuggers.

DDMS also listens on another debugging port, the DDMS “base port” (8700, by default). The base port is a port forwarder, which can accept VM traffic from any debugging port and forward it to the debugger on port 8700. This allows to debug all the VMs on a
device by attaching one debugger to port 8700. By the currently selected process in the DDMS Devices view the forwarded traffic is determined.

Figure 3.2 shows a typical DDMS screen in Eclipse. If you are starting DDMS from the command line, the screen is slightly different, but much of the functionality is identical. Notice that the highlighted process, com.example.android.notepad, that is running in the emulator has the debugging port 8700 assigned to it as well as 8609. This signifies that DDMS is currently forwarding port 8609 to the static debugging port of 8700.

![DDMS screen](image.png)

Figure 3.2. DDMS monitoring.

This Software development kit supports all the phone functions including database. We can expect it to be faster than the other options since it has least overhead while running it phone.

### 3.2 WEB FRAMEWORKS

PhoneGap Framework consists of one JavaScript file and one jar file which extends Android API classes and allows your HTML, JavaScript to control phone features like
Camera, Accelerometer, Geolocation, File, Contacts, Compass, Media, Vibrator and Storage. PhoneGap Framework can be used with Eclipse IDE.

The interesting topic to discuss here is how this js and jar file communicates with each other. We can do this by binding JavaScript code to Android code.

For developing a web application that's designed specifically for the WebView in your Android application, interfaces can be created between JavaScript code and client-side Android code. For example, instead of using JavaScript's alert() function JavaScript code can call a method in Android code to display a Dialog [9].

To bind a new interface between your JavaScript and Android code, call addJavaScriptInterface(), passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

For example, you can include the following class in your Android application:

```java
public class JavaScriptInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    JavaScriptInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

In this example, the JavaScriptInterface class allows the web page to create a Toast message, using the showToast() method. You can bind this class to the JavaScript that runs in your WebView with addJavaScriptInterface() and name the interface Android. For example:

```java
WebView webView = (WebView) findViewById(R.id.webview);
webView.addJavaScriptInterface(new JavaScriptInterface(this), "Android");
```

This creates an interface called Android for JavaScript running in the WebView. At this point, your web application has access to the JavaScriptInterface class. For example, here's some HTML and JavaScript that creates a toast message using the new interface when the user clicks a button:

```html
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />
```
The WebView automatically makes it available to your web page therefore no need to initialize the Android interface from JavaScript. So, at the click of the button, the `showAndroidToast()` function uses the Android interface to call the `JavaScriptInterface.showToast()` method.

The object that is bound to your JavaScript runs in another thread and not in the thread in which it was constructed.

Using `addJavaScriptInterface()` allows JavaScript to control your Android application which can be a very useful feature or a dangerous security issue. When the HTML in the WebView is untrustworthy (for example, part or all of the HTML is provided by an unknown person or process), then an attacker can include HTML that executes your client-side code and possibly any code of the attacker's choosing. Therefore, you should not use `addJavaScriptInterface()` unless you wrote all of the HTML and JavaScript that appears in your WebView. User should also not be allowed to navigate to other web pages that are not your own, within your WebView (instead, allow the user's default browser application to open foreign links—by default, the user's web browser opens all URL links, so be careful only if you handle page navigation as described in the following section).

### 3.3 HTML5 with CSS3 and JavaScript

To develop an Application using HTML5, CSS3 and JavaScript we don’t need any fancy editor or any framework. One browser and notepad is enough.

#### 3.3.1 Steps to Write an Application Using HTML, CSS, jQuery

Writing an application using HTML5 and jQuery frameworks requires some extra attention compared to web application. Below are the steps that can be used to write HTML application.

1. Convert the existing web page to iPhone friendly format.
2. To show/hide navigation panels, use dynamic HTML.
3. For a normal website, convert it into a full-screen AJAX application, using progress indicators, a native-looking back button, and a custom Web Clip icon.

4. You can use jQTouch to add native-looking animations.

5. You can use two ways to store user data on the client: (a) Key/Value storage (b) Client-side SQL database.

3.3.2 Client-Side Storage for Web Frameworks and HTML5

In a few years, all our files and documents will be stored in the cloud. We will not be storing anything on our desktops. While this has several advantages, such as location independence, data accessed over the internet can never be as speedy as accessing data locally. No matter how fast the internet connection, it can't compete with data being read locally from your hard drive. At a macro level, everything is moving towards the cloud and less and less data is been stored locally by users. However, at a micro level, specifically with browsers, a lot of data is being stored on the client side. The following are the reasons [10]:

- more responsiveness
- reduced load on server
- increased accessibility (you are remembered on your computer, you don't need to login every time)

Client-side storage is probably one of the most talked about features in HTML5. It has received a lot of criticism because of its lack of security, but it is nonetheless an interesting innovation.

3.3.2.1 How is Client-Side Storage Achieved?

The most popular method for local storage in all browsers has been the good old HTTP cookies. Cookies are key-value pairs of strings that are stored locally in a text file and are sent to the server with every HTTP request that goes to the same domain name. There are a few other browser-specific or plugin-dependent methods like:

- User Data: in Internet Explorer 5.5 and above.
- Local Shared Object: part of the Adobe Flash Player browser plugin.
- Google Gears: a plugin for Mozilla Firefox and Internet Explorer.
In an attempt to resolve this chaos and standardize client-side storage, the Web Hypertext Application Technology Working Group (WHATWG) came up with a well-structured client-side storage solution, which is part of the HTML5 specifications approved by W3C. As you must already be aware, HTML5 is a work in progress and is being implemented in browsers in phases. Fortunately, for us, client-side storage is one of those features that have already been implemented in most popular browsers like Safari, Firefox and Internet Explorer.

Client-side storage is divided into three methodologies:

- Session storage
- Local storage
- Database storage

### 3.3.2.2 SESSION STORAGE

Session Storage is very similar to Http cookies and has some additional benefits.

- Cookies are limited to 4 kilobytes of space, which can sometimes be a little restricting. Session storage allows much more space. Depending on the browser implementation, the exact space can vary, but it usually runs into megabytes.
- Cookies are sent to the server with every Http request. However Session data are not sent automatically. This reduces the payload that goes with every request. The developer can choose which key-value pairs need to be sent with each request.
- Most e-commerce sites use cookie-based authentication to remember which user is logged in on a computer, or more specifically browser session. If two users want to shop from the same site from two windows of the same browser, it is not possible because cookies are tied to the browser session. With session storage, this can be done as it is tied to the browser tab/window. As each tab/window maintains its own session information, as far as the site is concerned, the users are accessing their accounts from different systems.

sessionStorage should be used to store ephemeral data related to a single browser window as it doesn't persist after the window is closed. Using sessionStorage is extremely easy. It is like defining and using a regular variable, except that it needs to be prefixed with sessionStorage.

### 3.3.2.3 LOCAL STORAGE

The early drafts of HTML5 introduced a storage mechanism called Global Storage. In short, it is a memory space given by the browser that websites can use to store persistent data
that doesn't need to be sent to the server. The data is accessible by JavaScript and therefore by Flash (this feature is mainly used by Flash games). According to the specifications, you can define global storage objects in the following manner.

The specifications also mention that all the domains at an equal or lower level of the one defined for a global storage object can access its storage data. Fortunately browser developers didn't implement it as per specifications. They never supported public data storage and specifically not top-level-domain data access. Firefox in fact went to the extent of blocking access to any domain other than the one you specify.

After the browsers implemented their version of global storage, and their implementation made sense, WHATWG modified the specifications. They replaced global storage with local storage where you cannot specify domains; the data you store is automatically associated with the domain under which the script is running.

The localStorage JavaScript object is functionally identical to the sessionStorage object. They only differ in persistence and scope:

- Persistence: localStorage is used for long-term storage. Its data persists even after the window is closed. The data stored in sessionStorage is lost when the browser window closes.
- Scope: localStorage data is accessible across all browser windows while sessionStorage data is confined to the browser window that it was created in.

**3.3.2.4 DATABASE STORAGE**

With HTML5, you get database storage, which allows you to save structured data in the client's machine using a real SQL database. It uses SQLite database, which is light and fast but has a few limitations like its lack of foreign key constraints. You should remember that this is not the primary database, it is merely to store temporary data that can be accessed when the client is disconnected for a brief period of time. You will need to spend some time to figure out what data you really want to store in the client-side database.
CHAPTER 4

COMPARISON TYPES AND APPROACH

4.1 DEVICE USED FOR THE TEST APPLICATION

For the comparison, HTC sensation phone was used (see Table 4.1). This device runs under Android 2.3.4 and Table 4.1 shows the specification of the phone. While this testing, device was not connected to the computer and phone was running under normal circumstances. All the programs were closed except contacts, email client.

Table 4.1. Phone Specification Used for the Test Applications

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1.2 GHz dual-core processor, Adreno 220 GPU, Qualcomm MSM 8260 Snapdragon</td>
</tr>
<tr>
<td>Android Version</td>
<td>2.3.4</td>
</tr>
<tr>
<td>Model Number</td>
<td>Z710E</td>
</tr>
<tr>
<td>RAM</td>
<td>768 MB</td>
</tr>
<tr>
<td>Internal Storage</td>
<td>1 GB</td>
</tr>
<tr>
<td>HSPA/WCDMA</td>
<td>US: 900/AWS/2100 MHz</td>
</tr>
<tr>
<td>Quad-band GSM/GPRS/EDGE</td>
<td>850/900/1800/1900 MHz</td>
</tr>
<tr>
<td>Dimension</td>
<td>126.1 x 65.4 x 11.3 mm</td>
</tr>
<tr>
<td>Weight</td>
<td>148g</td>
</tr>
<tr>
<td>Size</td>
<td>540x960 pixels, 4.3 inch</td>
</tr>
</tbody>
</table>

4.1 EXECUTION SPEED COMPARISON

Every phone application has set of functions/tasks to perform and show results when triggered by either user or some sort of notification by other processes. These set of functions are nothing but a set of machine level instructions to compute results. Some function (for ex. finding a max of numbers) in different application environment can generate same or different set of instructions but the execution time will always be different. In general, running the same algorithm in different application environment and counting the time to perform them will give us basic idea of execution speed. This analysis can be very helpful if the application requires lot of computational functions/tasks.
In this study, Recursive Exponential Fibonacci algorithm will be executed in the three environments and the execution time of each would be recorded and compared to analyze the execution speed of each of the environment. The Recursive Exponential Fibonacci algorithm works on the following recurrence relation:

\[
\begin{align*}
    f(n) &= f(n-1) + f(n-2) \\
    f(1) &= f(0) = 1
\end{align*}
\]

```c
int fib(int n)
{
    if (n <= 2) return 1;
    else return fib(n-1) + fib(n-2);
}
```

The four internal nodes of this tree for fib(5), executes two operations each, while the five leaves executes one line, so the total number of lines executed in all the recursive calls is \((4 \times 2 + 5 \times 1) = 13\). Solving this recurrence relation for \(n = 4\), we would get the stack trace shown in Figure 4.1.

Note that, for any call to fib, the Fibonacci number \(F(i)\) at each internal node is just the number of leaves below that node, so the total number of leaves in the tree is just \(F(n)\). Remember that leaves count as one operation and internal nodes count as two operations. To count internal nodes, use basic fact about binary trees (trees in which each node has 2 children): the number of internal nodes always equals the number of leaves minus one. (This can be proved by induction: it's true if there's one leaf and no internals, and it stays true if 2 children are added to a leaf.)

So there are \(f(n)\) lines executed at the leaves, and \(2f(n)-2\) at the internal nodes, for a total of \(3f(n)-2\). Let's double check this on a simple example: \(f(5) = 3(5)-2 = 13\). This is kind of slow e.g. for \(n=45\) it takes over a billion steps which will give our comparison good amount of processing task. Thus this recursive relation seems ideal to test the execution speed of all environments, as not only does it grow exponentially, it also involves deep levels of nested method invocations, which requires the compilers to put current execution state onto the stack and jump over to another memory location and continue execution.

### 4.1.1 Fibonacci Algorithm Implementation for Native SDK Applications

As can be followed from the code, the button on the screen is used to initiate the fib(i) function calls. From 0 to 33, fib(i) gets called for each number and every time it
exponentially increases the amount of operations to be computed. To count the time between two function calls, we record the time before and after the function call.

```java
private int fib(int n) {
    return n <= 1 ? n : fib(n - 1) + fib(n - 2);
}
```

```java
long startTime = System.currentTimeMillis();
for (int i = 0; i < 35; i++) {
    int val = fib(i);
    result += (i + " : " + (System.currentTimeMillis() - startTime) + "\n");
}
resultView.setText(result);
```

Please refer to Appendix A for the full implementation.

IO operations are always slower than the memory operations. In the test application, IO operations (finding time elapsed after every function) have been avoided to remove any additional delay. Prime objective is to compare find only operation time to compute Fibonacci values.
4.1.2 Fibonacci Algorithm Implementation for HTML5 and PhoneGap Applications

In HTML5 and PhoneGap applications, JavaScript is used to perform any operations. When button gets clicked on the screen, it initiates the function call same as Native application. Time is recorded before and after the method call. To get the current time at any moment while execution is in progress, date object needs to be initialized.

```javascript
var d = new Date();
var curr_sec = d.getSeconds();
var curr_msec = d.getMilliseconds();
var start = (curr_sec * 1000) + curr_msec;
var i = 0;
var result = "start time: " + start + " :";
for(i=0; i<35 ;i++)
{
    var fibValue = fib(i);
    result += i + ",";
    var e = new Date();
    result += ((e.getSeconds() *1000) + e.getMilliseconds() - start) + "<br/>";
}
```

Here time is calculated in milliseconds from the date object. It takes four operations to find the current time in milliseconds. Since the number of operations to compute Fibonacci is much higher compared to this, it can be ignored. Again time can be saved by storing this time in variables rather than updating on the screen.

Even though both HTML5 and PhoneGap applications use JavaScript to perform operations, there can be difference in performance due to the overhead difference while executing the code in different environments. This will be discussed in next chapter when we do analysis of the results.

Please see the Appendix B and Appendix C for the full test code used for HTML5 and PhoneGap test.

4.2 Graphical Function Execution Speed Comparison

Almost all game applications require a decent graphics performance from the application frameworks. Graphics performance depends on speed of reading and writing data to RAM and speed of rendering objects onto the screen. Since different environment handles graphics operations in different ways, performance of the same graphical function can be
different. This is why it is necessary to compare the graphics performance in the available application development environments.

A simplest graphical test would be to render an object on the screen and move it. Moving a graphical object can be described in two programmatic operations.

1. Create an object at new location on the screen
2. Render the newly created object

A Graphical test can be performed by moving a rectangle object on the screen for finite period of time and measure the time each environment takes to render objects on the screen and move it. If the distance is same in different test applications, time taken by the applications can be very easily compared.

Visual speed depends on how many pixel object moves per second. Object can be redrawn farther than the previous location to achieve faster visual speed. This makes the object move faster without having faster ability of environment to respond quickly for graphical operation. So to summarize this, frame rate can be same for two objects moving with different speed. Objective of this study is to measure how fast environment can switch the frame which does not depend on visual speed.

### 4.2.1 Graphical Test for Native SDK Applications

In this test, a graphic view object is created which contains a rectangle, start line and end line by drawing lines for the appropriate dimensions and co-ordinates.

```java
public Rect r = new Rect(20, 20, 80, 80);
float[] startLine = {20,20,300,20};
float[] endLine = {20,320,300,320};

@Override
public void onDraw(Canvas canvas) {
    this.setBackgroundColor(Color.CYAN);
    canvas.drawRect(r, paintScr);
    canvas.drawLine(startLine[0],startLine[1],startLine[2],startLine[3], paintScr);
    canvas.drawLine(endLine[0],endLine[1],endLine[2],endLine[3], paintScr);
}
```

Graphic Test class initializes the graphic view object when application starts the execution. It also initializes the other necessary objects like button and text view to display the result. When the “Start” button is clicked by the user, it starts a new thread to make the graphical changes on the screen. Since the GUI application runs under the event dispatch
thread or main thread, it must not be used to update screen. The reason is explained in Appendix D. In this new thread, application updates the co-ordinate of the rectangle until it reaches the end of line, when it starts moving the rectangle in the other direction and it continues to oscillate it between start and end line. After updating the rectangle location by one unit in the memory, application calls the postInvalidate method. This method invalidates the specific area rather than updating the whole screen. This thread also measures the time taken by rectangle to finish the finite oscillations.

Please see the Appendix E for the full code used for this test.

4.2.2 Graphical Test for HTML5 and PhoneGap Applications

In this test HTML5 component <canvas> is being used to render rectangle. <canvas> creates a fixed size drawing surface that exposes one or more rendering contexts. JavaScript functions clearRect, fillStyle and fillRect are being used to render and move the rectangle on the canvas.

The draw function gets the canvas element then obtains the 2D context. The context object can then be used to actually render to the canvas. The fillStyle function uses rgba() to specify an alpha value along with the color.

User click on the button initiates the animation function which is written in JavaScript. This function records the starting time first and then it clears the screen before rendering the rectangle at a new coordinate with the use of clearRect function.

The code has slightly been modified compared to Native SDK. The reason is HTML5 does not update anything on the screen in the loop. This shows that it is running under the main UI thread. We have to trigger this kind of functions with the use of StartInterval() function provided by JavaScript, which gets trigger automatically when UI updates are done. In this example, we can take 0 time delay as we want to run it as fast as it can to achieve the maximum graphical performance. This example only moves one pixel at a time. So it might seem slower in movement. To increase the visual speed, one can increase two pixels at a time. This approach can double the movement on the screen but it still does the same number of frame updates. Our aim is to count the number of frames it can change.

```javascript
function startAnimation()
{
    count = 0;
}
```javascript
var elem = document.getElementById('myCanvas');
if (!elem || !elem.getContext) { return; }
// Get the canvas 2d context.
var context = elem.getContext('2d');
if (!context) { return; }
var dir = 0;
var d = new Date();
var curr_sec = d.getSeconds();
var curr_msec = d.getMilliseconds();

start = (curr_sec * 1000) + curr_msec;
start = new Date().valueOf();
intervalId = setInterval(drawRect,0);

}

function drawRect(){
    context.clearRect(0,0,320,450);
    context.fillStyle = '#00f';   // blue
    if (x + dx > WIDTH || x + dx < 0){
        dx = -dx;
        count++;
        if(count > 5){
            clearInterval(intervalId);
            ShowTimeInterval();
        }
    }
    x += dx;
    context.fillRect(20, x, 40, 40);
}
```

Please see the Appendix F and Appendix G for the full test code used for PhoneGap and HTML5 test.

4.3 DATABASE PERFORMANCE COMPARISON

Every application uses data, including phone applications. Database support is lifeline of every application. Unless any application deals only with simple data, it needs a database system to store structured data defined for the application. Database created for the phone application should be very small because phone devices are not designed to handle large amount of data processing. If any application requires large data processing, a separate database server has to be allocated and a web server can provide the resulting data with any supported communication between web server and phone device. For the average
application, a phone device should be capable of running simple data transaction which is widely used in basic Calendar, Phonebook, Navigation, Email and Social Media related applications. Since Android and HTML5 browsers use SQLite as database, similar data operational rate can be expected but still the performance can differ because framework affects the performance of handling the data requests with the database. Android framework has many classes which help in building the applications specific to database by minimizing the efforts to convert the result data into the objects. Next chapter will discuss further about the performance.

4.3.1 Database Test for the Native SDK Applications

In the test application, SQLiteOpenHelper can be used because this class provides all the basic interface methods needed to perform the simple database operations. To initialize this helper, application context has to be passed in the argument because in android you cannot access database created in one application from outside of this application. SQLiteOpenHelper supports OnCreate, OnUpgrade, insert, update, query, delete methods to create, upgrade, insert, update, query, delete on database tables. This support makes it very easy for developer to work with database operations compared to HTML5. DatabaseTest class initializes this DatabaseHelper (derived from SQLiteOpenHelper) instance which creates demo table if not exists. When user clicks on Insert button, it first creates one row in test table and then it updates this row for finite amount of times. It also measures the time taken to perform the update queries and shows it on the screen. To see the full application code, please see the Appendix H.

It is very important that all the applications created for comparison uses same approach in executing queries. For the SDK application, it is not uncommon to use batch transaction as other environments are not creating separate batches.

4.3.2 Database Test for the PhoneGap and HTML5 Applications

PhoneGap and HTML5 application for test is almost identical. It is necessary to test both this applications because they are getting executed in different environments. In the test application, jQuery function has been used to begin the database initialization when the page has been loaded into the browser. This function calls window.openDatabase method which
returns database object (db) if browser supports the database. (For more information see Appendix I.) If db object is initialized then it attaches the loaded event which creates the table demo if not exists. If there are already rows created in this database, then it returns the row data one by one and browser shows this data on the screen. There are three methods ClearTable(), InsertData(), and PopulateData() created in this test application. ClearTable() clears the existing demo table data.

```javascript
function ClearTable() {
    db.transaction(function(tx) {
        tx.executeSql("DELETE FROM DEMO");
        $('#DataDiv').html('');
    });
    return false;
}
```

InsertData() creates one row in the database tables and then calls the update query for finite number of times. It also measures the time taken to perform the update queries. This finite number of time was discovered by performing it multiple times to reach the point where it takes considerable amount of time.

```javascript
db.transaction(function(tx) {
    tx.executeSql('INSERT INTO DEMO (id, data) VALUES (1, "First row")', []);
    for(var i = 1; i<=5000 ; i++)
    
        tx.executeSql("UPDATE DEMO SET id = ?, data = ?", [i, "Row No: " + i]);

This function gets called when user initiates it by pressing click button and PopulateData shows the result on the screen.

Please see the Appendix J and Appendix K for the full test code used for PhoneGap and HTML5 test.
CHAPTER 5

COMPARISON AND ANALYSIS

5.1 FRAMEWORK COMPARISON ANALYSIS

Android SDK application runs over the Android operating system directly. There is no intermediate application or framework running in this case. Any application will only initialize the basic application context to run the application. While PhoneGap application runs over the PhoneGap framework. The Android Webview (android object which displays the web page) incorporates the application written in PhoneGap. When PhoneGap application starts, it loads the html document including JavaScript and CSS files in the Webview which is on top of the application context initialization. PhoneGap framework consists of a jar file and a JavaScript file. When application starts it adds an additional time taken for the initialization by these libraries apart from the application code initialization. HTML5 applications runs in browser which itself is an application in Android. Since HTML5 application does not need to initialize any objects besides loading the html document in the browser, there should not be much overhead of initialization in this case.

Android operating system will need to allocate memory and other resources needed by the application and a part of these resources will be used by the PhoneGap framework. PhoneGap application displays the webview which acts like a browser for the application. In this case, application has to carry the extra burden of running webview and has to display it as a browser inside an application. Also any JavaScript function uses a part of the resources being allocated to the PhoneGap application. If JavaScript function allocates the memory dynamically, there needs to be more memory management. Recursive function call adds tremendous amount of stack size. As a result, application which does greater stack management can take different amount of execution time.

5.2 WITH RESPECT TO EXPONENTIAL ALGORITHM

In this part, we will discuss and compare about three different application environments with respect to an exponential algorithm.
In all the three applications, code written to compute the \( \text{fib}(n) \) value is almost same besides calculating the time in milliseconds. In SDK, `System.currentTimeMillis()` function has to compute the time in milliseconds before returning it. Also when the value of \( n \) increases, \( \text{fib}(n) \) has to perform very large number of operations which will reduce the effect of time taken by time function. Let’s look at the timing figures and Table 5.1.

**Table 5.1. Algorithm Comparison between Environments (Android 2.3 HTC Sensation)**

<table>
<thead>
<tr>
<th>Fibonacci index</th>
<th>Value</th>
<th>PhoneGap (time in millis)</th>
<th>SDK (time in millis)</th>
<th>HTML5 (time in millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>34</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>55</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>89</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>144</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>233</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>377</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>610</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>987</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>17</td>
<td>1597</td>
<td>2</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>18</td>
<td>2584</td>
<td>4</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>19</td>
<td>4181</td>
<td>6</td>
<td>16</td>
<td>7</td>
</tr>
<tr>
<td>20</td>
<td>6765</td>
<td>9</td>
<td>24</td>
<td>17</td>
</tr>
<tr>
<td>21</td>
<td>10946</td>
<td>12</td>
<td>37</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>17711</td>
<td>18</td>
<td>49</td>
<td>37</td>
</tr>
<tr>
<td>23</td>
<td>28657</td>
<td>30</td>
<td>70</td>
<td>52</td>
</tr>
<tr>
<td>24</td>
<td>46368</td>
<td>50</td>
<td>105</td>
<td>59</td>
</tr>
<tr>
<td>25</td>
<td>75025</td>
<td>79</td>
<td>159</td>
<td>70</td>
</tr>
<tr>
<td>26</td>
<td>121393</td>
<td>114</td>
<td>245</td>
<td>88</td>
</tr>
<tr>
<td>27</td>
<td>196418</td>
<td>163</td>
<td>391</td>
<td>118</td>
</tr>
<tr>
<td>28</td>
<td>317811</td>
<td>237</td>
<td>624</td>
<td>167</td>
</tr>
<tr>
<td>29</td>
<td>514229</td>
<td>366</td>
<td>995</td>
<td>245</td>
</tr>
<tr>
<td>30</td>
<td>832040</td>
<td>598</td>
<td>1604</td>
<td>373</td>
</tr>
<tr>
<td>31</td>
<td>1346269</td>
<td>926</td>
<td>2593</td>
<td>598</td>
</tr>
<tr>
<td>32</td>
<td>2178309</td>
<td>1434</td>
<td>4172</td>
<td>940</td>
</tr>
<tr>
<td>33</td>
<td>3524578</td>
<td>2248</td>
<td>6704</td>
<td>1475</td>
</tr>
<tr>
<td>34</td>
<td>5702887</td>
<td>3643</td>
<td>10827</td>
<td>2348</td>
</tr>
</tbody>
</table>
Since the Emulator figures do not match with Phone figures, this study will not consider the Emulator values. This proves that Android Emulator do not replicate the same performance as actual phone.

This result shows that HTML5 and PhoneGap application function took less time in calculating the time in milliseconds while java function in SDK test application increases rapidly.

Table 5.1 shows that SDK application takes more time in performing large amount of operations compared to HTML5 and PhoneGap applications. HTML5 applications are slightly faster compared to PhoneGap when the n increases. This shows that HTML5 has slightly high capability of performing large operations. The reason could be stack memory management delay to run the same amount of operations. HTML5 only depends on how the memory management happens in browser versus PhoneGap and SDK also depends on stack memory management provided by SDK. The application which takes more memory takes more time to allocate large amount of memory compared to application which takes less memory. There is no easy way to find different between the stack memory management handled by Browser application running in Android compared to the application runs independently. The possible reason could be the stack memory management or dynamic memory allocation management handled in these environments. Here the important result is HTML5 applications are faster compared to PhoneGap applications which is faster compared to SDK applications in performing large amount of operations. See Figure 5.1.

5.3 WITH RESPECT TO GRAPHICAL FUNCTIONS

In this part, we will compare three application environments with graphical functions. In this test, PhoneGap and HTML5 applications are using the same JavaScript code. To calculate the difference between the performances, it is realistic to increase the number of cycles and then take the average because browser performance might slightly change depending on the circumstances at that moment.

Table 5.2 shows the result (units are in milliseconds).

In one cycle, each application has moved the square 600 times. It implies 3000 frame changes for five cycles. Table 5.3 shows the frame rates (in milliseconds) for each
Figure 5.1. Algorithm performance graph (Android 2.3 HTC Sensation).

Table 5.2. Graphical Execution Time per 5 Cycles (Android 2.3 HTC Sensation)

<table>
<thead>
<tr>
<th>Iteration/5 cycle</th>
<th>SDK (time in millis)</th>
<th>PhoneGap (time in millis)</th>
<th>HTML5 (time in millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>217</td>
<td>21187</td>
<td>20311</td>
</tr>
<tr>
<td>2</td>
<td>201</td>
<td>20536</td>
<td>21231</td>
</tr>
<tr>
<td>3</td>
<td>125</td>
<td>20731</td>
<td>21164</td>
</tr>
<tr>
<td>4</td>
<td>213</td>
<td>20609</td>
<td>20424</td>
</tr>
<tr>
<td>5</td>
<td>174</td>
<td>20328</td>
<td>21157</td>
</tr>
<tr>
<td>6</td>
<td>125</td>
<td>20361</td>
<td>19630</td>
</tr>
<tr>
<td>Average: 1 cycle</td>
<td>35.15833</td>
<td>4125.067</td>
<td>4130.567</td>
</tr>
</tbody>
</table>

environment. Here frame does not mean the full screen. It means the 60x60 pixel square used in all test applications. This is the reason for seeing the high rate.

According to the numbers, SDK application can deliver more than 100 times better performance compared to PhoneGap and HTML5. This difference is not negligible when performance matters the most in developing the game. It would be still possible to use the HTML5 for small games like puzzles, social games, strategy games, shooting, etc. These types of games do not require high frame rate. See Figure 5.2.
Table 5.3. Graphical Execution Frame per Time

<table>
<thead>
<tr>
<th></th>
<th>SDK (frame/millis)</th>
<th>PhoneGap (frame/millis)</th>
<th>HTML5 (frame/millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.8217</td>
<td>0.141596</td>
<td>0.147703</td>
<td></td>
</tr>
<tr>
<td>14.9513</td>
<td>0.146085</td>
<td>0.141303</td>
<td></td>
</tr>
<tr>
<td>23.9904</td>
<td>0.144711</td>
<td>0.14175</td>
<td></td>
</tr>
<tr>
<td>14.0943</td>
<td>0.145567</td>
<td>0.146886</td>
<td></td>
</tr>
<tr>
<td>17.21664</td>
<td>0.14758</td>
<td>0.141797</td>
<td></td>
</tr>
<tr>
<td>24.0096</td>
<td>0.147341</td>
<td>0.152827</td>
<td></td>
</tr>
<tr>
<td>17.06566</td>
<td>0.145452</td>
<td>0.145259</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>Average</td>
<td>Average</td>
<td></td>
</tr>
<tr>
<td>17.8708</td>
<td>0.145476</td>
<td>0.145361</td>
<td></td>
</tr>
</tbody>
</table>

Figure 5.2. Graphical execution comparison graph.

5.4 WITH RESPECT TO DATABASE PERFORMANCE

As we know that writing anything on the disk is the expensive operation compared to reading. For our comparison test in three environments, this operation can be used to see the difference of execution time. Here the aim of the study is not only to see the actual query execution time but also to include the execution time taken by code which is required in the normal circumstances.
By number of trials, a finite number was taken which was taking decent time to finish the execution. This number and the execution time can be considered to compare performance of three platforms. See Table 5.4.

Table 5.4 Database execution time taken by update query (Android 2.3 HTC Sensation)

<table>
<thead>
<tr>
<th>Number of Update Query</th>
<th>SDK (time in millis)</th>
<th>HTML5 (time in millis)</th>
<th>PhoneGap (time in millis)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>135</td>
<td>216</td>
<td>141</td>
</tr>
<tr>
<td>5000</td>
<td>220</td>
<td>143</td>
<td>122</td>
</tr>
<tr>
<td>5000</td>
<td>136</td>
<td>137</td>
<td>135</td>
</tr>
<tr>
<td>5000</td>
<td>134</td>
<td>162</td>
<td>139</td>
</tr>
<tr>
<td>5000</td>
<td>116</td>
<td>124</td>
<td>183</td>
</tr>
<tr>
<td>5000</td>
<td>144</td>
<td>114</td>
<td>201</td>
</tr>
<tr>
<td>Average</td>
<td>147.5</td>
<td>149.3333</td>
<td>153.5</td>
</tr>
</tbody>
</table>

As the number shows, all the three environment have same database performance. The difference in execution time is very less and can be ignored. The reason why we are receiving same performance is due to SQLite. All the three environments are using same database server.

SDK provides various framework classes to perform complicated queries to retrieve the data. These extra framework classes which supports number of features.

SQLiteDatabase: It exposes methods to manage a SQLite database. SQLiteDatabase has methods to create, delete, execute SQL commands, and perform other common database management tasks.

SQLiteOpenHelper: A helper class to manage database creation and version management. You can create subclass implementing onCreate(SQLiteDatabase), onUpgrade (SQLiteDatabase, int, int) and optionally onOpen(SQLiteDatabase), and this class takes care of opening the database if it exists, creating it if it does not, and upgrading it as necessary. Transactions are used to make sure the database is always in a sensible state.

This class makes it easy for ContentProvider implementations to defer opening and upgrading the database until first use, to avoid blocking application startup with long-running database upgrades.

These features are not available in HTML5 and PhoneGap unless used a framework which supports them. It is more likely that their execution time will go up if extra code used to perform these additional tasks.
Code used in these test applications are used in normal circumstances. There can be better way to use other optimized way to perform the same tasks but the aim of the study is to consider the normal case. See Figure 5.3.

Figure 5.3. Database execution comparison graph.
CHAPTER 6

MATRIX OF APPLICATIONS COMPARISON AND
CONCLUSION

6.1 MATRIX OF APPLICATIONS COMPARISON

Let’s summarize the comparison by giving the rank to three areas. See Table 6.1.

Table 6.1. Application Comparison Matrix

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Graphics</th>
<th>Database</th>
<th>Multi platform support</th>
<th>Phone hardware feature access</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhoneGap</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>HTML5</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>SDK</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>No</td>
</tr>
</tbody>
</table>

SDK application can perform better graphics operations compared to others and it performs equally in database.

PhoneGap and HTML5 are slower in graphics operations but faster in performing large amount of operations. They also support multi-platform support.

The major difference here is SDK applications have to be written in their own IDE and with Native Frameworks. Developer has to learn all different Frameworks and IDE and has to rewrite the code for all different platforms. To create an application to make it available on multiple platforms, will take lots of efforts in writing the same login in multiple programming language. Also developer needs to learn using multiple SDKs which will require extra development time, complexity and cost. This will multiply the possibility of introducing bugs in the code, thus multiplying the amount of testing needed. Maintenance of such an application will ask the developer to maintain separate lines of code in the different programming languages.

While PhoneGap and HTML5 applications can run on different platforms without any extra effort of learning and cost of development. Since HTML5 applications will run in browsers, search engines can easily find them and can increase the popularity of an
application better than native applications. Marketing of these applications will become much easier which will save the marketing cost.

If the application does not require best graphical performance then PhoneGap or HTML5 would be the better option because newer phones has high end processors and with enough RAM to run multiple applications simultaneously. Certainly HTML5 does not give the access to phone hardware features for ex. Accelerometer, Vibrate, Camera, Contacts and Internal application access, etc.

By considering these results, one can pick your environment to develop the application.

6.2 FUTURE ENHANCEMENTS

Further study can be done on why there is a difference between exponential function executions in these three environments. This study can include stack memory management and dynamic memory allocation management in the above three environments. A similar analysis can be done for iPhone, Windows and Palm OS. This analysis can also include different approaches in comparison.

There are many web frameworks available now a day. All these frameworks are not focused on HTML5. Some of them convert the code from one language to others to support multiplatform support. This study can include the comparison amongst the framework performance and supported features. A separate analysis can be done to compare them with native SDK. This analysis can include available features on each framework.


APPENDIX A

FIBONACCI TEST APPLICATION CODE IN
NATIVE SDK
package cs.sdsu.app1;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class SDK_FIBONACCI extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button b = (Button) findViewById(R.id.StartButton);
        final TextView resultView = (TextView) findViewById(R.id.ResultTextView);

        b.setOnClickListener(new OnClickListener() {
            public void onClick(View v) {
                String result = "";
                long startTime = System.currentTimeMillis();
                for (int i = 0; i < 35; i++) {
                    int val = fib(i);
                    result += (i + " : " + (System.currentTimeMillis() - startTime) + "\n");
                }

                resultView.setText(result);
            }
        });
    }

    private int fib(int n) {
        return n <= 1 ? n : fib(n - 1) + fib(n - 2);
    }
}

Main Layout Xml

<?xml version="1.0" encoding="utf-8"?>

<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/ScrollView01" android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/hello" />
        <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/StartButton"
            android:text="Click to start fibonacci series"></Button>
    </LinearLayout>
</ScrollView>
<TextView android:layout_width="wrap_content" android:layout_height="wrap_content" android:id="@+id/ResultTextView"></TextView>
</LinearLayout>
</ScrollView>
APPENDIX B

FIBONACCI TEST APPLICATION CODE IN HTML5
<script>
function CheckSpeed()
{
    var d = new Date();
    var curr_sec = d.getSeconds();
    var curr_msec = d.getMILLISECONDS();
    var curr_min = d.getMinutes();
    var start = (curr_sec * 1000) + curr_msec;
    var i = 0;
    var result = "start time: " + start + " :
    for(i=0;i<35;i++)
    {
        var fibValue = fib(i);
        result += i + ","
        var e = new Date();
        result += (e.getSeconds()*1000) + e.getMILLISECONDS() - start + "<br />";
    }
    //document.getElementById("disp").innerHTML = fib(3);
    //for ( i = 0; i < 40; i++ ) {
    //    result += " "+ (d.getSeconds()*1000) + d.getMILLISECONDS() - start) + " "+ fib(i);
    //}
    document.getElementById("disp").innerHTML = result;
}
function fib(n) {
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}
</script>

<h2>Local Url:
content://com.android.htmlfileprovider/sdcard/DCIM/static.html</h2>

<a href="#" class="btn large" onclick="CheckSpeed();">Click HERE</a>

<div id="disp"></div>
APPENDIX C

FIBONACCI TEST APPLICATION CODE IN PHONEGAP
function CheckSpeed()
{
    var d = new Date();
    var curr_sec = d.getSeconds();
    var curr_msec = d.getMilliseconds();
    //var curr_min = d.getMinutes();
    var start = (curr_sec * 1000) + curr_msec;
    var i = 0;
    var result = "start time: " + start + " :";
    for( i=0; i<35; i++)
    {
        var fibValue = fib(i);
        result += i + ",";
        var e = new Date();
        result += ((e.getSeconds() * 1000) + e.getMilliseconds() - start) + 
        "<br />";
    }
    document.getElementById("disp").innerHTML = result;
}

function fib(n) {
    return n <= 1 ? n : fib(n-1) + fib(n-2);
}

<a href="#" class="btn large" onclick="CheckSpeed();">Click HERE</a>
APPENDIX D

EVENT DISPATCH THREAD AND THREAD POOL
Android applications run in a native Linux process, in the underlying Linux OS. This process houses activities (screens), widgets, and services (non visual long running application parts). When working with Android apps, it is important to remember to keep long running code running in threads that are not tied to the main thread or event dispatch thread, in order to get an “application not responding” error. A common mistake that is made is long running tasks are performed in this EDT/main thread, and this leads to lots of application failures [11].

**What is the EDT or main thread?**

Figure D.1 outlines what the role of the EDT or main thread is (I will refer to it as the EDT from now on). All UI operations are performed in a single thread in the native Linux process. This is pretty much the way all graphics on desktop and mobile platforms are performed. This single thread of execution is called the EDT, and it is spawned by the OS when you have a graphical app (as opposed to a command line app or an entirely non visual one). All UI operations, paints, updates, etc. are done on this thread, and it is important to keep long running, high latency tasks off of this thread of execution. E.g., it is a bad idea to resolve a URL and load content over HTTP in this thread.

![Figure D.1. Event dispatch thread.](image)
In order to combat the single threaded nature of the EDT, a simple event dispatching model is used. In Android, this is implemented using Handler and Looper classes. It goes something like this, if you want the EDT to do something, create a Runnable and inject it into the EDT’s task queue. The EDT will then process this Runnable, when it gets around to it. You can’t mess with this queue, except to “post Runnables” to it. Alternatively, if you have a long running process that must do something in a non-EDT or background thread (BGT), you have to post a Runnable that then injects the results of this long running background task to the EDT at some point. This is encapsulated by the Android AsyncTask class.

**Strategies**

Based on this event dispatch model, there are a few patterns that emerge, that are reusable across a large variety of applications.

1. Perform long running tasks in a BGT. This BGT can be created by you calling new Thread(), or more preferably, by using the Executor framework. You will find a large variety of executors to meet the needs of many different kinds of requirements. Eg, you can replace a Timer with a scheduled executor. You can use a single thread executor instead of spawning new threads all the time, or you can use a thread pool executor for that. There are a lot of options at your disposal, and you should be able to find something that meets your requirements.

2. Once long running tasks are completed in the BGT, post those results to the EDT in a Runnable. Sometimes, your long running tasks will not require any UI updates, other times, they will. In those cases, where UI updates are needed, you have to create a new Runnable closure/functor/anon inner class implementation that delivers the results of the computation to the UI (by updating a model, or creating some widget, etc.). In order to this, you will need a Context, and it’s Handler to call post() or postDelayed() on.

3. Inside of your UI code, there are times when you need some UI operation to happen at a later time. In these cases you can post or postDelayed Runnables on a Context’s Handler just like in the previous case. There are various reasons for doing this, eg: showing a popup at a later time, or performing an animation, or even “chunking” long running UI operations into smaller chunks.
APPENDIX E

GRAPHICS TEST APPLICATION CODE IN
NATIVE SDK
GraphicsView.java

```java
package cs.sdsu.graphics;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.view.View;

public class GraphicView extends View {
    private Paint paintScr = new Paint();
    Canvas can;

    public Rect r = new Rect(20, 20, 80, 80);
    float[] startLine = {20, 20, 300, 20};
    float[] endLine = {20, 320, 300, 320};

    public GraphicView(Context context) {
        super(context);
    }

    @Override
    public void onDraw(Canvas canvas) {
        this.setBackgroundColor(Color.CYAN);
        canvas.drawRect(r, paintScr);
        canvas.drawLine(startLine[0], startLine[1], startLine[2], startLine[3], paintScr);
        canvas.drawLine(endLine[0], endLine[1], endLine[2], endLine[3], paintScr);
    }
}
```

GraphicsTest.java

```java
package cs.sdsu.graphics;

import android.app.Activity;
import android.graphics.Rect;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.TextView;

public class GraphicsTest extends Activity {
    /** Called when the activity is first created. */
    long startTime, interval;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public void onClick(View v) {
        super.onClick(v);
    }
}
```
LinearLayout layout = new LinearLayout(this);
layout.setOrientation(1);
final TextView textView = new TextView(this);
textView.setText("Graphic Test");
layout.addView(textView);
Button btn = new Button(this);
btn.setText("Start Animation");
final Button timeBtn = new Button(this);
timeBtn.setText("Time Elapsed");
timeBtn.setEnabled(false);
LinearLayout btnLayout = new LinearLayout(this);
btnLayout.addView(btn);
btnLayout.addView(timeBtn);
btnLayout.setOrientation(0);
layout.addView(btnLayout);
final GraphicView view = new GraphicView(this);
layout.addView(view);
setContentView(layout);
btn.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        Thread thr = new Thread(new Runnable() {
            public void run() {
                Rect r = view.r;
                int count = 0;
                int dir = 0;
                startTime = System.currentTimeMillis();

                while (count < 100) {
                    if (r.bottom == 320) {
                        dir = 1;
                    } else if (r.bottom == 80) {
                        dir = 0;
                        count++;
                    } else {
                        r.set(r.left, r.top + 1, r.right, r.bottom + 1);
                    }
                    interval = System.currentTimeMillis() - startTime;
                    view.postInvalidate();
                }
            }
        });
        thr.start();
        timeBtn.setEnabled(true);
    }
});
timeBtn.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
}});
if (interval == 0)
    textView.setText("Timer: " + (System.currentTimeMillis() -
    startTime));
else
    textView.setText("Timer: " + interval);
}
});
}

Layout.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:animationCache="true">
    <TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello" />
    <Button android:text="Start" android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/StartButton"></Button><View
    android:layout_width="wrap_content" android:layout_height="wrap_content"
    android:id="@+id/GraphicView"></View>
</LinearLayout>
APPENDIX F

GRAPHICS TEST APPLICATION CODE IN PHONEGAP
<doctype html>
<html>
<head>
<meta name="viewport" content="width=320; user-scalable=no" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>PhoneGap</title>
<link rel="stylesheet" href="master.css" type="text/css" media="screen" title="no title" charset="utf-8">
<script type="text/javascript" src="jquery.min.js"></script>
<script type="text/javascript" charset="utf-8" src="phonegap-0.9.3.js"></script>
<style type="text/css">
.View {
  background-color: blue;
  width: 320px;
  height: 500px;
}
.StartLine {
  margin-top: 19px;
  background-color: white;
  height: 1px;
}
.Rect {
  background-color: white;
  position: absolute;
  margin-top: 1px;
  margin-left: 20px;
  width: 60px;
  height: 60px;
}
.clearFix {
  clear: both;
  height: 0;
  width: 100%;
  display: block;
}
</style>
<script><!--
var elem;
var context;
var x = 0;
var y = 450;
var dx = 1;
var WIDTH = 320;
var HEIGHT = 450;
var intervalId;
var count = 0;
var start;
$(document).ready(function(){
  elem = document.getElementById('myCanvas');
  if (!elem || !elem.getContext) {  return;  }
  // Get the canvas 2d context.
  context = elem.getContext('2d');
  context.drawImage(canvas, 0, 0, 320, 450);
  start = setInterval(function(){
    x = x + dx;
    if (x >= WIDTH) { dx = -1; } else if (x < 0) { dx = 1; }
    context.drawImage(canvas, x, 0, 320, 450);
  }, 200); // 200 ms
  stop = clearInterval(start);
  console.log("Canvas rendered.");
});
</script>
context = elem.getContext('2d');
if (!context) { return; }

function startAnimation()
{
    count = 0;
    var elem = document.getElementById('myCanvas');
    if (!elem || !elem.getContext) { return; }
    // Get the canvas 2d context.
    var context = elem.getContext('2d');
    if (!context) { return; }
    var dir = 0;
    var d = new Date();
    var curr_sec = d.getSeconds();
    var curr_msec = d.getMilliseconds();
    start = (curr_sec * 1000) + curr_msec;
    start = new Date().valueOf();
    intervalId = setInterval(drawRect, 0);
}

function ShowTimeInterval()
{
    var e = new Date();
    var end = new Date().valueOf();
    var result = "Start TimeStamp: " + start + " <br /> Interval: " + (end - start) + " <br />");
    $('#Result').html(result);
}

function drawRect()
{
    context.clearRect(0, 0, 320, 450);
    context.fillStyle = '#00f'; // blue
    if (x + dx > WIDTH || x + dx < 0){
        dx = -dx;
        count++;
        if (count > 5){
            clearInterval(intervalId);
            ShowTimeInterval();
        }
    }
    x += dx;
    context.fillRect(20, x, 40, 40);
}

</head>
<body id="stage" class="theme">
<h1>Welcome to PhoneGap!</h1>
<h2>this file is located at assets/index.html</h2>
<div id="Result" class="clearFix">
    Time Interval</div>
<a href="#" onclick="startAnimation();">Click HERE to Start</a>
<canvas id="myCanvas" width="320" height="450">
This Browser does not support Canvas.
</canvas>
</body>
</html>
APPENDIX G

GRAPHICS TEST APPLICATION CODE IN HTML5
<!DOCTYPE HTML>
<html>
<head>
<meta name="viewport" content="width=320; user-scalable=no" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8"/>
<title>HTML5 Graphics Test</title>
<script type="text/javascript" src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
<script><!--
var elem;
var context;
var x = 0;
var y = 450;
var dx = 1;
var WIDTH = 320;
var HEIGHT = 450;
var intervalId;
var count = 0;
var start;
$(document).ready(function() {
    elem = document.getElementById('myCanvas');
    if (!elem || !elem.getContext) { return; }
    // Get the canvas 2d context.
    context = elem.getContext('2d');
    if (!context) { return; }
    var dir = 0;
    var d = new Date();
    var curr_sec = d.getSeconds();
    var curr_msec = d.getMilliseconds();
    start = (curr_sec * 1000) + curr_msec;
    start = new Date().valueOf();
    intervalId = setInterval(drawRect,0);
})

function startAnimation(){
    count = 0;
    var elem = document.getElementById('myCanvas');
    if (!elem || !elem.getContext) { return; }
    // Get the canvas 2d context.
    var context = elem.getContext('2d');
    if (!context) { return; }
    var dir = 0;
    var d = new Date();
    var curr_sec = d.getSeconds();
    var curr_msec = d.getMilliseconds();
    start = (curr_sec * 1000) + curr_msec;
    start = new Date().valueOf();
    intervalId = setInterval(drawRect,0);
}

function ShowTimeInterval()
{
    var e = new Date();
    var end = new Date().valueOf();
    var result = "Start TimeStamp: "+start+" Interval: "+( end - start) + ":";
    $('#Result').html(result);
}-->
function drawRect()
    {
        context.clearRect(0,0,320,450);
        context.fillStyle = '#00f';  // blue
        if (x + dx > WIDTH || x + dx < 0){
            dx = -dx;
            count++;
            if(count > 5){
                clearInterval(intervalId);
                ShowTimeInterval();
            }
        }
        x += dx;
        context.fillRect(20, x, 40, 40);
    }
</script>
</head>
<body id="stage" class="theme">
<h1>Welcome to HTML5!</h1>
<h2>Graphics Test</h2>
<div id="Result" class="clearFix"> Time Interval</div>
<a href="#" onclick="startAnimation();">Click HERE to Start</a>
<canvas id="myCanvas" width="320" height="450">
This Browser does not support Canvas.
</canvas>
</body>
</html>
APPENDIX H

DATABASE PERFORMANCE TEST
APPLICATION CODE IN NATIVE SDK
DatabaseHelper.java

```java
package edu.sdsu.DatabaseTest;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "Demo.db";
    private static final int DATABASE_VERSION = 1;
    private static final String ID = "_id";
    private static final String DATA = "data";
    private static final String TABLE = "Demo";

    public DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE if not exists " + TABLE + " (" + ID + " integer primary key autoincrement, " + DATA + " text " + ");");
    }

    public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {
    }
}
```

TestApplicationDatabaseHelper.java

```java
package edu.sdsu.DatabaseTest;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;

public class TestDatabaseHelper {
    private DatabaseHelper mDbHelper;
    private SQLiteDatabase mDb;
    private final Context mCtx;

    public TestDatabaseHelper(Context ctx) {
        this.mCtx = ctx;
    }

    public synchronized TestDatabaseHelper open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
    }
```
public void close() {
    mDbHelper.close();
}

public long createNewRow(String data) {
    ContentValues initialValues = new ContentValues();
    initialValues.put(DATA, data);
    return mDb.insert(TABLE, null, initialValues);
}

public void massUpdateRow(int updateTimes) {
    mDb.beginTransaction();
    for (int i = 0; i < 500; i++) {
        ContentValues value = new ContentValues();
        value.put(DATA, "Row No " + i);
        mDb.update(TABLE, value, null, null);
    }
    mDb.setTransactionSuccessful();
    mDb.endTransaction();
}

public Cursor getAllData() {
    return mDb.query(TABLE, new String[] {ID, DATA}, null, null, null, null, null);
}

public boolean deleteData() {
    int rowsDeleted = mDb.delete(TABLE, null, null);
    if(rowsDeleted > 0)
        return true;
    else
        return false;
}
)

DatabaseTest.java

package edu.sdsu.DatabaseTest;

import android.app.Activity;
import android.database.Cursor;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class DatabaseTest extends Activity {
    /** Called when the activity is first created. */
    private TestDatabaseHelper mDbHelper;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        mDbHelper = new TestDatabaseHelper(getApplicationContext());
        mDbHelper.getWritableDatabase();

        Button button = (Button) findViewById(R.id.button);
        button.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                // Button clicked, do something
            }
        });

        mDbHelper.close();
    }
}

public class TestDatabaseHelper {
    public static final String DATABASE_NAME = "test.db";
    public static final int DATABASE_VERSION = 1;

    public TestDatabaseHelper(Context context) {
        super();
    }

    public SQLiteDatabase getWritableDatabase() {
        return SQLiteDatabase.openDatabase(DATABASE_NAME, null, SQLiteDatabase.CREATE_IF_NECESSARY);
    }

    public void close() {
        SQLiteDatabase db = getWritableDatabase();
        db.close();
    }
}

// More code follows...
setContentView(R.layout.main);
mdbHelper = new TestDatabaseHelper(this);
mdbHelper.open();
mdbHelper.createNewRow("First Row");
Button insertBtn = (Button) findViewById(R.id.InsertBtn);
Button clearBtn = (Button) findViewById(R.id.ClearBtn);
Button displayBtn = (Button) findViewById(R.id.DisplayBtn);
final TextView messageView = (TextView) findViewById(R.id.MessageText);
final TextView timerView = (TextView) findViewById(R.id.TimerText);

insertBtn.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        Cursor cur = mdbHelper.getAllData();
        if (cur.getCount() == 0) {
            mdbHelper.createNewRow("First Row");
        }
        runOnUiThread(new Runnable() {
            public void run() {
                long currentTime = System.currentTimeMillis();
                mdbHelper.massUpdateRow(5000);
                timerView.setText("Time Taken: " + (System.currentTimeMillis() - currentTime));
            }
        });
    }
});
clearBtn.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        timerView.setText("Data Row Cleared: " + mdbHelper.deleteData());
    }
});
displayBtn.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        Cursor cur = mdbHelper.getAllData();
        cur.moveToFirst();
        timerView.setText("Data : ");
        while (cur.isAfterLast() == false) {
            timerView.append(" , " + cur.getString(1));
            cur.moveToNext();
        }
        cur.close();
    }
});

Main.xml

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello"/>
<LinearLayout android:layout_width="wrap_content"
    android:id="@+id/LinearLayout01"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Button android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Insert Data"
        android:id="@+id/InsertBtn"></Button>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Display Data"
        android:id="@+id/DisplayBtn"></Button>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Clear Data"
        android:id="@+id/ClearBtn"></Button>
</LinearLayout>

<TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:id="@+id/MessageText"/>

<TextView android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    android:id="@+id/TimerText"/>
APPENDIX I

OPEN DATABASE METHOD TO INITIALIZE THE DATABASE
The openDatabase() method returns a Database object. The method takes four arguments: a database name, a database version, a display name, and an estimated size, in bytes, of the data that will be stored in the database. The openDatabase() method must use and create databases from the origin of the Document of the Window object on which the method was invoked. If the database version provided is not the empty string, and the database already exists but has a different version, or no version, then the method must raise an INVALID_STATE_ERR exception. Otherwise, if the database version provided is the empty string, or if the database doesn't yet exist, or if the database exists and the version provided to the openDatabase() method is the same as the current version associated with the database, then the method must return a Database object representing the database that has the name that was given. If no such database exists, it must be created first.
APPENDIX J

DATABASE PERFORMANCE TEST
APPLICATION CODE IN PHONEGAP
<!DOCTYPE HTML>
<head>
<meta name="viewport" content="width=320; user-scalable=no" />
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>PhoneGap Database</title>
<link rel="stylesheet" href="master.css" type="text/css" media="screen" title="no title" charset="utf-8"/>
<script type="text/javascript" src="jquery.min.js"></script>
<script type="text/javascript" charset="utf-8">

var db = null;
$(document).ready(function(){
  try {
    if (window.openDatabase) {
      db = openDatabase("DatabaseTest", "1.0", "HTML5 Database API example", 200000);
      if (!db)
        alert("Failed to open the database on disk. This is probably because the version was bad or there is not enough space left in this domain's quota");
      else
        alert("Couldn't open the database. Please try with a WebKit nightly with this feature enabled");
    } catch (err) {
      db = null;
      alert("Couldn't open the database. Please try with a WebKit nightly with this feature enabled");
    }
    if (db != null)
      addEventListener('load', loaded, false);
  });

  function loaded() {
    db.transaction(function(tx) {
      tx.executeSql("SELECT COUNT(*) FROM Demo", [], function(result) {
        loadData();
      }, function(tx, error) {
        tx.executeSql("CREATE TABLE IF NOT EXISTS DEMO (id unique, data)", [], function(result) {
          loadData();
        });
      });
    });
  }

  function loadData() {
    db.transaction(function(tx) {
      tx.executeSql("SELECT id, data FROM Demo", [], function(tx, result) {
        for (var i = 0; i < result.rows.length; ++i) {
          var row = result.rows.item(i);
          $('#DataDiv').append("<span>" + row['id'] + "</span>"));

        }
      });
    });

</script>
```javascript
(function() {
    var d = new Date();
    var curr_min = d.getMinutes();
    var start = (d.getSeconds() * 1000) + d.getMilliseconds();
    for(var i = 1; i <= 5000; i++)
        tx.executeSql("UPDATE DEMO SET id = ?, data = ?", [i, "Row No: " + i]);
    var e = new Date();
    $('#Timer').html("Time to run update query for 5000 times: " + ((e.getSeconds() * 1000) + e.getMilliseconds() - start) + "<br />");
});
```

```html
<h1>PhoneGap Database Example</h1>

<a onclick="InsertData()" href="#">Click To Insert</a><br />
<a onclick="ClearTable()" href="#">Clear Data</a><br />
<a onclick="PopulateData()" href="#">Populate Data</a><br />
<br />
<br />
</html>
```
APPENDIX K

DATABASE PERFORMANCE TEST
APPLICATION CODE IN HTML5
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"  
"http://www.w3.org/TR/html4/strict.dtd">  
<html>  
<head>  
<title>Database Example</title>  
<script type="text/javascript"  
src="http://code.jquery.com/jquery-1.4.2.min.js"></script>  
<script type="text/javascript" charset="utf-8">  
var db = null;  
$(document).ready(function() {  
try {  
  if (window.openDatabase) {  
    db = openDatabase("DatabaseTest", "1.0", "HTML5 Database API example", 200000);  
    if (!db)  
      alert("Failed to open the database on disk. This is probably because the version was bad or there is not enough space left in this domain's quota");  
    } else  
      alert("Couldn't open the database. Please try with a WebKit nightly with this feature enabled");  
  } catch(err) {  
    db = null;  
    alert("Couldn't open the database. Please try with a WebKit nightly with this feature enabled");  
  }  
  if (db != null)  
    addEventListener('load', loaded, false);  
  //var db = window.openDatabase("Database", "1.0", "Database Demo", 200000);  
  //db.transaction(populateDB, errorCallback, successCallback);  
})};  
function loaded()  
{  
  db.transaction(function(tx) {  
    tx.executeSql("SELECT COUNT(*) FROM Demo", [],  
      function(result) {  
        loadData();  
      },  
      function(tx, error) {  
        tx.executeSql("CREATE TABLE IF NOT EXISTS Demo (id unique, data)", [],  
          function(result) {  
            loadData();  
          });  
    });  
  });  
});  
function loadData()  
{  
  db.transaction(function(tx) {  
    tx.executeSql("SELECT id, data FROM Demo", [],  
      function(tx, result) {  
        for (var i = 0; i < result.rows.length; ++i) {  
          var row = result.rows.item(i);  
          $("#DataDiv").append("<span>" + row['id'] +  
"</span>");  
        }  
      });  
  });  
}  
</html>
```javascript
$("#DataDiv").append("<span>
" + row['data'] + "</span>"));
}

if (!result.rows.length)
{
    alert("Table exists with 0 rows");
}
}, function(tx, error) {
    alert('Failed to retrieve data from database - ' + error.message);
    return;
});

function ClearTable()
{
    db.transaction(function(tx){
        tx.executeSql("DELETE FROM DEMO");
        $("#DataDiv").html("");
        return false;
    });

    function InsertData(){
        ClearTable();

        db.transaction(function(tx) {
            tx.executeSql('INSERT INTO DEMO (id, data) VALUES (1, "First row")',[]);
            var d = new Date();
            //var curr_min = d.getMinutes();
            var start = (d.getSeconds() * 1000) + d.getMilliseconds();

            for(var i = 1; i<=5000 ; i++)
                tx.executeSql("UPDATE DEMO SET id = ?, data = ?",[i, "Row No: " + i]);

            var e = new Date();
            $("#Timer").html("Time to run update query for 5000 times: " + ((e.getSeconds() *1000) + e.getMilliseconds() - start) + "<br />");
            return false;
        });

        function PopulateData(){
            loadData();
            return false;
        }
    }
</script>
</head>
<body>
<h1>Example</h1>
<p>Database</p>
<a onclick="return InsertData()" href="#">Click To Insert</a>
<br />
<a onclick="return ClearTable()" href="#">Clear Data</a>
<br />
<a onclick="return PopulateData()" href="#">Populate Data</a>
<br />
```
<br />
<div id="Timer"></div>
<br />
<div id="DataDiv"></div>
</body>
</html>