An Evaluation Framework for Cross-Platform Mobile Application Development Tools

by

Sanjeet Dhillon

A Thesis
Presented to
The University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada

ABSTRACT

AN EVALUATION FRAMEWORK FOR CROSS-PLATFORM DEVELOPMENT TOOLS

Sanjeet Dhillon
University of Guelph, 2012

Advisor:
Professor Qusay H. Mahmoud

The mobile application market is becoming increasingly fragmented with the availability of multiple mobile platforms that differ in development procedures. Developers are forced to choose to support only some platforms and specific devices due to limited development resources. To address these challenges, numerous tools have been created to aid developers in building cross-platform applications, however, there is no metric to evaluate the quality of these tools. This thesis introduces a framework for evaluating the features, performance and discuss development experience of existing and future cross-platform development tools. The framework is implemented by benchmarking several tools and the results identify a disparity in the features and performance. This research is carried out in collaboration with industrial partner Desire2Learn, through an NSERC Engage Grant.

Acknowledgements

I would like to thank all those individuals that assisted me in the process of completing this thesis. There are many staff, faculty and others who have contributed to bringing me to the point of completing this degree.

I would like to first thank those who provided me personal support, my friends and family. In particular, I would like to thank my friends Nicholas and Samaneh who have been a constant force of encouragement and motivation. The staff at the Centre for Students with Disabilities office has gone above and beyond providing guidance in completing my degree and I am most appreciative of them.

Development contributions for some applications described in this thesis were provided by Undergraduate Research Assistants at the Centre for Mobile Education and Research (CMER) who I would like to thank for their hard work, in particular Sacha Bagasan, Domenico Commisso and Justin Carvalho.

Dr. Fangju Wang has been very receptive and helpful in providing guidance in my seminar and in finalizing this thesis.

I would like to thank my advisor, Dr. Qusay H. Mahmoud for the assistance and support during the duration of this research and for the funding I received in the form of Research Assistantships through CMER which is partially funded by Research In Motion. Additional funding was provided through an NSERC Engage grant with industry partner Desire2Learn. The staff from Desire2Learn have contributed ideas and provided real-world questions to be answered by this research. Their development queries have proven invaluable in shaping the framework to what is needed by the market.

TABLE OF CONTENTS

Acronyms and Abbreviations

API	Application Programming Interface
CMER	Centre for Mobile Education and Research
CSV	Comma Separated Value
CPDT	Cross-Platform Development Tool
CSS	Cascading Style Sheet
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
JVM	Java Virtual Machine
MFLOPS	Millions of Floating Point Operations Per Second
MVC	Model View Controller
NDK	Native Development Kit
PIM	Personal Information Manager
SDK	Software Development Kit
UI	User Interface
UX	User Experience

List of Figures

List of Tables

Chapter 1
Introduction

The world of mobile operating systems has been becoming increasingly fragmented in recent years. Each company or consortium has developed their own non-standard method of developing applications for their devices using a variety of programming languages and Software Development Kits (SDKs). This means that an application built for Google's Android operating system will not run on the rival Apple iOS platform. There is further fragmentation inside platforms with some applications not being able to be run on different versions or devices of the same platform making development far more challenging.

Fragmentation of the market is caused by many factors. Aside from software platform diversity, hardware variations create difficulty porting a User Experience (UX), including interaction method and User Interface (UI) from one device to the next [3]. With over 20 million tablet devices sold in 2011, it has been difficult to enable applications to maximize the larger screens of these devices [64]. Many developers are forced to choose to support only some platforms and versions due to limited financial resources or knowledge of coding techniques for each platform. Without wide developer support, platforms are seen as weak and have difficulties in marketing products. This is true for new operating systems and new releases that break compatibility with the legacy software. An example of this is Research in Motion moving to the BB10 platform breaking compatibility with natively developed BlackBerry OS applications [57].

With the number of languages used, developers must be very versatile and businesses need to expend significant resources to have their software available on more than one platform. Even with the added expense, an August 2012 Appcelerator and IDC survey shows companies continue to be very interested in multiple platforms

despite the difficulties [7]. In 2011, developers have shown interest in running on twice as many platforms as a similar survey in the previous year and now averaging an incredible four operating systems [4] [6]. This trend has continued to increase in 2012 [7].

With so much interest in availability of applications for many platforms, there is an apparent problem facing developers in requiring the ability to have their applications available to the widest possible audience, but lack resources to build natively for each platform. In order to further the reach of applications, without expending the considerable time needed to learn the quirks of each platform, using Cross-Platform Development Tools (CPDTs) may be the answer. The CEO of InRuntime has said using CPDTs has reduced the time to market by 70% [68]. This shows that using cross-platform tools can be very helpful in many ways. However, there is no adequate measure to ensure these CPDTs are as capable as native development tools. This leaves developers unsure if they can make use of these tools to create strong applications with native like functionality and performance.

This chapter will present a discussion of the motivation for research in this area. It then will go on to introduce the thesis statement and outline the structure of this thesis.

1.1 Motivation

Prior to the start of the research in this thesis, as part of the work at CMER, we developed the BioChem Euchre Deck application for the BlackBerry platform [19]. This application provides a learning interface for students using flash cards created by Professor John Dawson of the University of Guelph. Following initial development, we were left wondering how to best provide the application to students on all platforms. While choosing a cross-platform solution and developing the application, several shortcomings were noted. The lack of adequate comparison of these tools

made the task of finding which tool would best meet our requirements very difficult and highlighted the need for an evaluation method.

PhoneGap was seen in our unstructured evaluation to be an ideal candidate for simple, widely deployed applications. The application shown in Figure 1.1, was built and deployed. The wider reach from using a CPDT allowed for over 6000 downloads on Android and 2000 on each of iOS and BlackBerry showing the benefits of going cross-platform.

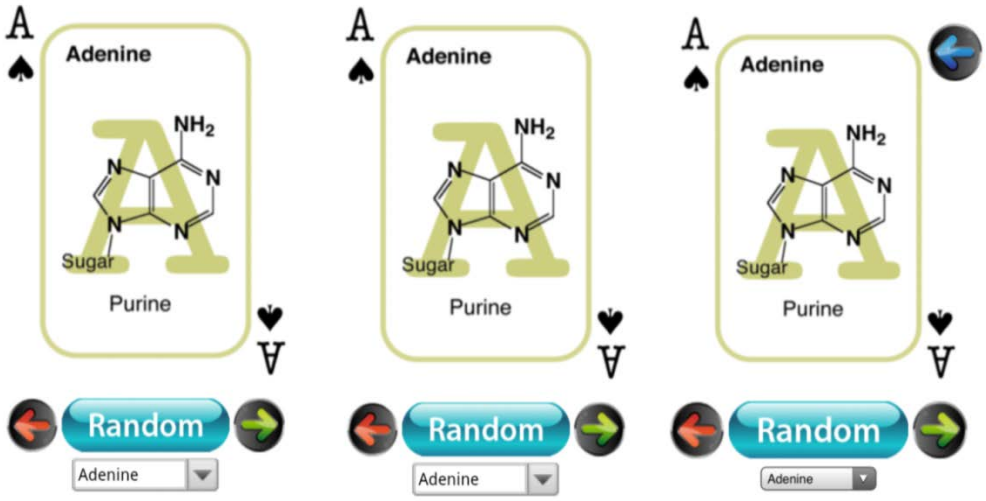

Figure 1.1: BioChem Euchre Deck Application on BlackBerry (left), Android (center) and iOS (right)

Unfortunately, with the lack of evaluation available, we did not know if the tool met the needs of our application prior to the start of development. Some limitations were apparent with the PhoneGap Build service such as the inability to use focused view on the BlackBerry platform, thus creating applications that have a mouse cursor. This is expected to be resolved in a subsequent release of PhoneGap but left us with the need to compile the BlackBerry version of the application using the standard WebWorks SDK and a different config.xml file which provides information about the application. The application itself did not need to be modified, however. Had we had an evaluation of the PhoneGap tool available to us prior to developing

this application, we would have known that this limitation made it not ideally suited to our purpose.

Some CPDTs have been developed and numerous features are implemented to work across many devices but it is difficult to ascertain which features are missing. Much of the information available for these tools comes from the vendors and there is little independent analysis available to developers. Additionally, there is a lack of comparison between these frameworks and their native development counterparts. How well do they perform? Do you lose functionality? Which are the best?

With no current benchmarks available, a process for testing the frameworks against one another and their native counterparts is required in order to provide guidance and answer the central question of finding a cross-platform development solution without trade-offs. Benchmarking protocols and tools will need to be developed in order to test the many frameworks as well as a study to be conducted into features and shortcomings of each.

1.2 Thesis Statement

This thesis focuses on exploring the viability of using CPDTs for mobile smartphone development in detail. Although cross-platform tools on mobile devices have come far from early incarnations, perceived trade-offs and inefficiency still hampers their use. The goal of this thesis is to create a framework to provide thorough comparison of CPDTs to one another and native development tools to determine if they can be used as an efficient development method in place of native tools. Through development of a feature and benchmark intensive evaluation framework for CPDTs, developers will gain the knowledge needed to determine which tool to use for their application.

In order to determine if CPDTs can be as capable as native development tools, a set of evaluation criteria for development tools has been created. A CPDT should be evaluated on the basis of features, performance and developer experience.

By creating an evaluation framework and applying it to select CPDTs, developers will gain useful knowledge of which tool best suits their purpose and if a cross-platform solution is a viable, cost-effective alternative to traditional native development.

1.3 Research Approach

The main focus of this research was the development of the evaluation framework. This includes the benchmarking tools and the discussion of the obtained quantitative results for several CPDTs in the marketplace.

In order to determine whether the central thesis has sound basis, numerous steps were taken:

1. Survey the tools available for both native and cross-platform development,
2. Discover prior research into evaluation of such development tools,
3. Create a high level evaluation framework using real-world development questions,
4. Develop benchmarks to achieve quantitative results and apply it to several CPDTs and native tools,
5. Analyze results, and make conclusions.

Although CPDTs do currently exist, it is still unknown whether or not these tools meet the performance and feature requirements to be comparable with their native counterparts.

Very little research has been conducted on this subject as these CPDTs are all quite new with most performance analysis being anecdotal, not scientific. It is believed that mobile web-based CPDTs may suffer from performance issues but it is unknown if it is true with the recent advancements in JavaScript performance [4]. A benchmarking framework must be developed in order to test these CPDTs against one another and native code. The development questions will be obtained by surveying current research, developer surveys, discussions with our industry partner and by discovering the capabilities of native tools.

During the development of this evaluation framework, certain assumptions had to be taken into account. The tools are expected to have the ability to run similarly developed benchmark tests while remaining cross-platform. The tests are developed using technologies that while tablet and smartphone friendly, only focus on smartphone testing. Testing for tablets is left for future work. It is understood that certain background processes cannot be stopped and may impact testing so care should be taken to reduce their influence on the results.

This research is conducted as part of an NSERC Engage project with the industry partner, Desire2Learn, a leader in development of e-learning software. Desire2Learn has provided real-world questions and feedback on how this framework can best benefit mobile developers. The development expertise and knowledge of development issues of the staff at Desire2Learn has been incorporated into development of each step of this research.

1.4 Organization of Thesis

In Chapter 1, we provided an introduction to the subject area, discussed the motivation behind conducting this research and described the thesis statement. The following chapters will describe an evaluation approach for mobile development tools. The thesis is structured as follows:

Chapter 2: Discusses the topic in more detail to provide necessary background information and details on some select CPDTs. It then discusses limited the prior work in this area.

Chapter 3: Describes the proposed solution, an evaluation framework that can be used for any CPDT.

Chapter 4: Introduces an implementation of the framework in more detail and the experiments used to test its viability.

Chapter 5: Presents the results of the testing of various CPDTs and native development tools using our evaluation framework.

Chapter 6: Provides concluding remarks and future work that can be used to extend this framework.

Chapter 2

Background and Related Work

It is important to understand each of the mobile platforms and development languages before being able to understand the scope of the problem solved by CPDTs. This chapter will provide background on the mobile landscape and an overview of various CPDTs. This overview is necessary to understand which aspects are most important and should be addressed in any evaluation framework. This will be followed by discussion of related work for evaluating CPDTs.

2.1 Current State of the Mobile Platform Landscape

The current landscape has eight major smartphone platforms listed in Table 2.1. However, many of these are losing importance with iOS, Android, BlackBerry 10 and Windows Phone becoming the major remaining players for the foreseeable future [7].

Table 2.1: Programming Environments for Mobile Platforms

Operating Environment	Preferred Programming Language
RIM BlackBerry OS	Java ME, HTML, JavaScript
RIM BlackBerry 10	C/C++, HTML, JavaScript, Java
Apple iOS	Objective-C
Google Android	Java (Harmony Flavored), C and C++
HP WebOS	HTML, JavaScript
Windows Phone	C#, .NET
Symbian	C,C++
Samsung Bada	C++, HTML, JavaScript

With the number of languages listed, developers must be very versatile and businesses need to expend significant resources to have their software available on more than one platform. Comscore, a leader in measuring digital market share, estimates that in the United States, Android has over 52% market share with the rest being split between Apple, RIM, Microsoft and Symbian [23]. Figure 2.1 shows the

level of developer interests in mobile platforms. This figure includes mention of both tablet and smartphone editions of each platform.

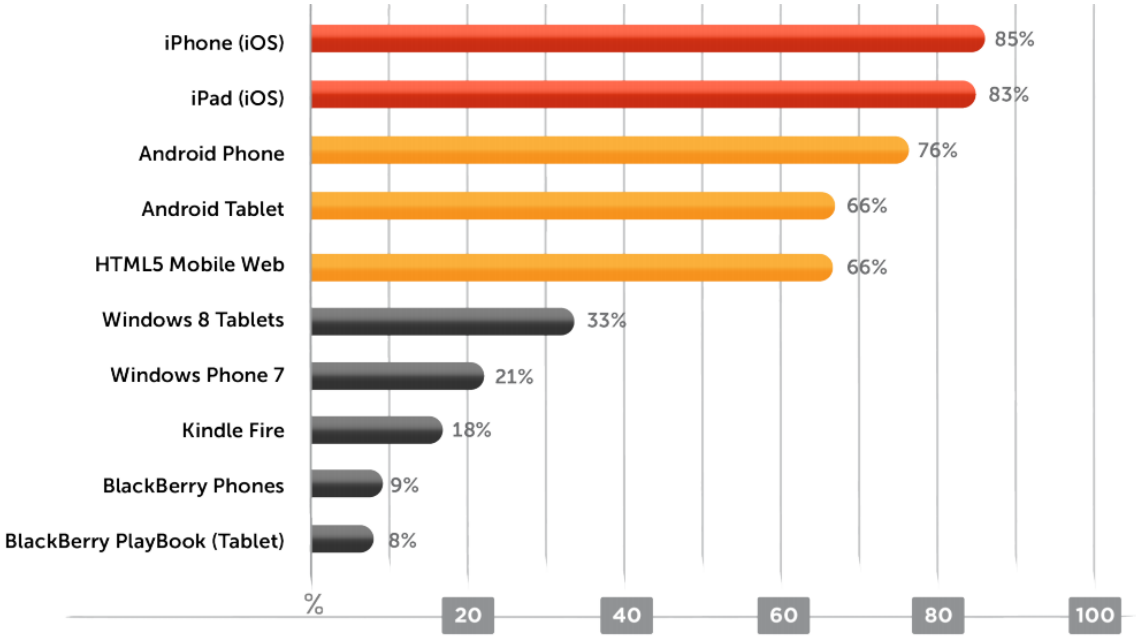

Figure 2.1: Developer Interests in Mobile Platforms [7]

With so much interest in availability for many platforms, there is an apparent problem facing developers in requiring the ability to have their application available to the widest possible audience, but lack resources to build natively for each platform. In order to further the reach of applications, without expending the considerable time needed to master each platform; using a CPDT may be the answer.

2.2 History of Cross-Platform Development

Java was believed to be marvelous for PC development for allowing a program to be written once and translated into an intermediary language before being run on a Java Virtual Machine (JVM) on a user's system. This allowed a single program to be run on Windows, Mac and Linux with no platform dependency. Java Micro Edition (Java ME) came to mobile devices but never went mainstream often thought to be due to its limited capabilities, perceived performance issues, and device fragmentation [16] [24]. Java was soon abandoned by some handset makers or forked into customized

versions with added functionality. Cross-platform tools that extended the Java Mobile architecture had gained some ground with Bedrock, Celsius, NeoMAD and alcheMo entering the market [54]. These tools allowed applications to run on a significant number of devices that supported Java, but the tools were outpaced by significant leaps in technology for native platform SDKs. These tools seemed to lose relevance with the advent of the new era of highly versatile smartphones.

Since 2011, a new set of CPDTs has been coming to market with new features being added rapidly over time. This latest generation of tools, will be discussed in the next section.

2.3 CPDTs for Mobile Development

There are several tools for cross-platform mobile application development available on the market today with various levels of functionality and compatibility. This new generation of CPDT allows more control, functionality and diversity than the previous generation of Java based tools [24] [37] [54]. The Cabana tool in [27], helps bridge the gap between the older and newer generation in terms of features. We have researched five tools as shown in Table 2.2. These tools were chosen due to their flexibility, feature support and popularity amongst developers. The chosen tools provide samples of different approaches to cross-platform development with cross-compiled, runtime and web wrapper styles. They were also required to support only a minimum of two distinct platforms allowing for a wider array of analysis of emerging CPDTs. Mobile web browser based applications are included for comparative purposes.

Table 2.2: Tools for Cross-Platform Application Development

CPDT	BlackBerry OS	BlackBerry 10	iOS	Android	Windows Phone 7	Bada
Mobile Web	✓	✓	✓	✓	✓	✓
Adobe PhoneGap	✓		✓	✓	✓	✓
Appcelerator Titanium		Beta	✓	✓		
Rhomobile Rhodes	✓		3.0+	1.6+	✓	
Adobe Air		✓	✓	✓		
MoSync			✓	✓	✓	

The CPDTs described in Table 2.2 vary greatly in capabilities, language and platform support. They can be split into categories based on the method used to compile and run the applications built using the tools. While some CPDTs such as MoSync can cross-compile the application into native code, others use a web wrapper, essentially running the application in an HTML5 compatible browser object. Traditional runtimes and extensions such as those used in Adobe Flash are widespread on desktops and notebooks but have limited support on mobile platforms and are being phased out. The alternative is using a similar approach to bridge the divide which is found in Adobe Air [72]. Each of these methods has distinct advantages over purely web-based applications adding possible performance, user-interface and notification enhancements not available to web developers. These added capabilities can quickly become an interesting proposition for a developer wishing to have more fully featured applications.

Some of these tools are already in use by the likes of NBC Universal, eBay and The New York State Senate allowing them to provide the application to as many customers as possible at a low cost [8].

Table 2.3: CPDT Features

CPDT	Development Language	Compilation Type	Native UI Elements	Access to Sensors *	File System Access	User & System Notifications	App store Support
Mobile Web	HTML5 + JavaScript	Runs in Browser		Limited	Limited		
Adobe PhoneGap	HTML5 + JavaScript	Native Web App		✓	✓	✓	✓
Appcelerator Titanium	HTML5 + JavaScript / Python / Ruby / PHP	Native Code and Runtime	✓	✓	✓	✓	✓
Rhomobile Rhodes	Ruby	Runtime	✓	✓	✓	✓	✓
Adobe Air	ActionScript/ HTML/AJAX	Runtime	✓	✓	✓	✓	✓
MoSync	C / C++ / HTML5 + JavaScript	Native Code	✓	✓	✓	✓	✓

*Camera, GPS, Accelerometer

The tools presented in Table 2.2 use different scripting languages and provide various features that are not compatible across all mobile platforms. In Table 2.3 we show many features provided by each tool, including the development language of choice.

2.3.1 Mobile Web

The promise of the mobile web as a standard based architecture is that it will display applications similarly despite being on different platforms and devices. Rich web applications can be written using HTML5 and JavaScript. These provide a high degree of functionality like in Google's GMail web application. This idea is not a new approach and although it does solve many issues for developers it does add a few new concerns. Mobile web applications that run through a phone's browser often do not match the phones native UI which can be quite confusing. However, these applications can leverage JavaScript and CSS (Cascading Style Sheets) frameworks such as JQuery Mobile [65], Zepto [30] and Sencha Touch [60] to provide much needed UI enhancement with little work on the developer's part. Many such frameworks exist to enhance the web development experience and these are only three of them that have been optimized for mobile handsets.

Browser security historically has not allowed storage of local data or access to device sensors such as an accelerometer. The webinos framework proposed in [42], provides an alternative to allow more functionality but does not have widespread adoption. Additionally, the browsers themselves have been fragmented with different implementations of JavaScript and rendering engines which can make functionality and interface behave inconsistently on different devices [59]. The largest concern is often offline access where there cannot be any access to the application when there is no connectivity.

Are these problems a deal breaker in using this as a development environment? The answer is quite subjective but the downsides are growing to be less and less as

the technology progresses. HTML5 has allowed for deeper integration with the handset and with WebKit being nearly ubiquitous across handsets, the rendering issues of the past are becoming less of a major consideration [20]. Unfortunately, living in the browser still blocks these programs from the application markets which are useful in helping end users find the developers work.

While mobile widgets add some features, it is still largely limited to the browser [40]. An approach taken by many CPDT developers is taking the concept and ideals of widgets and the mobile web and enhancing it with the features that were missing. The next sections will discuss tools that take an integrated approach to cross-platform development.

2.3.2 Adobe PhoneGap

PhoneGap [1] is an implementation of the recently renamed Apache Cordova open source project. Cordova and PhoneGap are tightly linked and will be discussed as one tool. PhoneGap is one of the most popular mobile web based approaches to cross-platform development. The tool takes advantage of the standardized web technologies used for mobile web development and brings them to native web applications. Much like the W3C's Widget standard; PhoneGap uses HTML, CSS and JavaScript to write applications that are then encapsulated in a browser object to look like applications built using native code. With this approach, the developer can submit the application to the application store to enable purchase and high visibility. Along with deep integration with the phone operating system and hardware, applications can make use of local storage and are fully available when there is no network connectivity. Many of these advantages are not present in standard mobile web applications.

In Table 2.2 and Table 2.3 we see the wide support for devices and features with PhoneGap. Through an API, developers can make use of various JavaScript functions use underlying native device capabilities. Most notably missing is native

UI elements, however this trade-off was made in order for the ability of having it available on so many platforms.

Many 3rd party tools are available to be used with PhoneGap, along with the already mentioned JavaScript libraries; the appMobi XDK for PhoneGap [12] provides an IDE and testing environment to aid developers significantly. This comes with tools to emulate the look of the mobile application on a variety of devices. appMobi additionally provides an analytics platform and their own build service similar to that provided by PhoneGap directly [11]. This service allows compilation of applications without the need to install an SDK for each mobile platform. With the addition of this XDK, the PhoneGap development lifecycle begins to resemble that of native applications much more closely.

Another tool that enhances PhoneGap is IBM Worklight [36]. It provides additional APIs and server side integration frameworks for enterprise level applications. The architecture of Worklight, seen in Figure 2.2, makes heavy use of PhoneGap to bridge the JavaScript code to the native device. It adds the Worklight API to enhance security, add analytics and access to their middleware, Worklight Server. This server provides the connection to the enterprise backend systems for the mobile application [37].

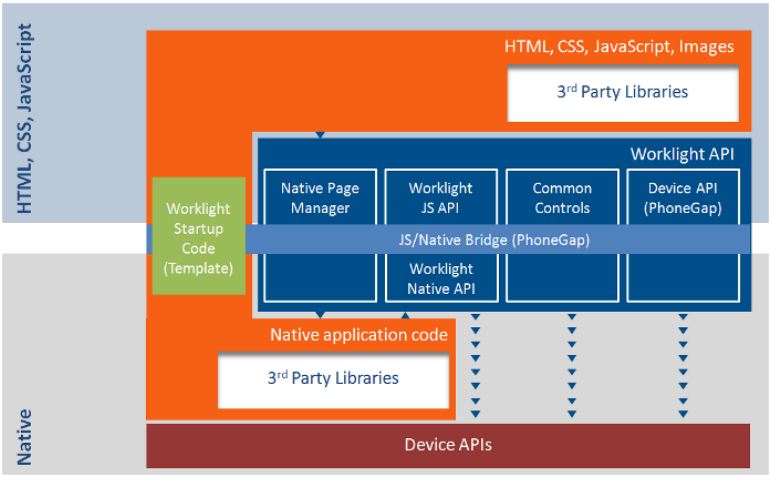

Figure 2.2: IBM Worklight Architecture [35]

Rather than using the Chrome browser frame based approach as in the appMobi XDK, Worklight has an IDE called Worklight Studio that is based on Eclipse. Once applications are deployed, all analytics and push notifications are handled through the Worklight console [37].

Whether using the PhoneGap directly, appMobi XDK or the suite of Worklight tools, PhoneGap provides an incredibly flexible approach for mobile application development. For small scale applications that need to be completed quickly to large enterprise level applications, there is a variant of this tool suitable for many use cases.

2.3.3 Appcelerator Titanium

Appcelerator Titanium [8] shares many traits with PhoneGap and the mobile web. Development for all three use standard web development languages, however Titanium differs in some important areas.

Running on an open source core, Appcelerator products allow for applications to run natively on devices without embedding a browser object as we have seen in PhoneGap. Instead, their approach uses a runtime object and compilation method to optimize and compile code. Options are provided to compile and run using a runtime; build into a web application, or a hybrid application similar to PhoneGap. The claim is that performance levels are increased to match those of native applications when using the runtime based approach [10] [68]. This assertion by the vendor requires independent analysis through a standard benchmarking architecture to determine accuracy. The UI approach also differs greatly with the use of native UI elements rather than focusing on a purely web-like interface. This is seen as a major advantage by many people as the application will not cause confusion by having a different look and feel than those built using native tools. Others, especially those transitioning mobile web applications, may find this as a negative and find this handcuffs the customizability of the interface [6].

Like the IBM Worklight extension to the PhoneGap core, Titanium provides a host of services for analytics and server side hosting. Similar to Worklight, server side features such as data storage and push notifications are available. The Appcelerator team does not provide the application the same level of end to end security guarantees found with Worklight and is more of a consumer, rather than enterprise focused architecture [10] [37]. Titanium includes Titanium Studio, an Eclipse based IDE and a set of testing tools and emulators.

Appcelerator Titanium lacks in platform support with only iOS and Android being fully supported with BlackBerry OS support being put into beta release one year ago [9]. This beta was discontinued and replaced with an upcoming BlackBerry 10 OS beta. Supporting only two of the major mobile platforms provides only the minimum required for being called cross-platform and large segments of the market are left out when developing applications using this tool.

2.3.4 Rhomobile Rhodes

Rather than using web standards, a different approach has been taken by Motorola Solutions in Rhodes [45]. Rhodes is developed by Rhomobile which has been acquired by Motorola Solutions, the enterprise and government focused division of Motorola. This part of the company was not acquired by Google and is independent of the Google Android project [68].

This method is based on the Ruby programming language which allows current Ruby developers to have an easy transition. Not all Ruby gems are translated to work inside Rhodes; however, more can be added as needed [50]. The tool provides strong UI tools including access to native UI elements for Android and iOS. Unlike many other tools, Rhodes has full Model View Controller (MVC) support, negating the need to have business logic in the view as JavaScript may have in other CPDTs

[63]. This separation can have major positive impact within the development process for various screen dimensions.

Rhodes uses a runtime based approach that has a runtime built using the C++ NDK for Android, which interprets the Ruby bytecode for the platform. The Rhodes vendor considers this to be more efficient than using Java on Android [68]. Rhodes offers the RhoStudio IDE with build tools, debuggers and simulators. Application hosting and enterprise data connection is a strong push for Rhodes with RhoConnect and RhoSync server. This positions RhoSync to compete for the enterprise market and Worklight. The final piece, RhoHub puts the services together into an interface with Git-powered source control and an online build service similar to PhoneGap Build. While open sourced under an MIT Licence, much of the functionality is only available through paid subscription for services from RhoMobile [68].

2.3.5 Adobe Air

Adobe Air is aimed at building rich internet applications that can be run on many platforms. It uses Adobe Flash, Adobe Flex, HTML, ActionScript and Ajax for development scripting. Flash skills are easily found in the industry and that is perceived as a major advantage for Adobe Air. Existing Flash applications can often be translated to run as native applications using Air instead of them being run through a browser extension. Air applications, unlike Flash, require installation of the runtime as well as each application. Mobile support on Android and iOS is present with an independent runtime available on Android and necessary elements being packaged with the application in iOS. As of version 2.6, most features are on par for the Android and iOS versions of the platform. It is suggested that runtimes like these can be an effective way of battling the cross-platform question however; there has been a mixed reception from device manufacturers [18] [20].

Air developer tools are some of the most robust and supported in the industry. Since the tool stems from the popular Flash tool, the same tools used like Flash

Builder and Dreamweaver can be used for development. When combined with Adobe Flex technologies, enterprise applications are possible with rich UI capabilities that may surpass those of other platforms. Adobe does not provide the integrated experience and the end to end functionality, tools and services provided by some CPDTs previously discussed.

These runtime based approaches found in Appcelerator Titanium, Rhomobile Rhodes and Adobe Air do have significant drawbacks in some areas. The main one is the dependence on the runtime environment itself. The mobile OS makers may at some point block these from their stores and the included runtime can cause larger file size. The features may also lag behind what is released natively as it is integrated into the runtime. As execution often uses just in time compilation, performance may be impacted when using any of these runtimes [43].

2.3.6 MoSync

MoSync is much like Air, however, the major differentiator with MoSync to other CPDTs is that it uses cross-compilation where the code is translated into native Objective C and Java code depending on the platform. The C++ compiler used in MoSync outputs an intermediary language that is then optimized and outputted as a binary for the desired platform. This is a different approach that may lead to increased performance of applications. MoSync also offers MoSync Wormhole a PhoneGap like CPDT that uses a browser object to display the application. C and C++ code can be translated using their wormhole technology to be used in these web based applications.

MoSync offers compatibility with OpenGL ES, which allows 3D graphics for game development therefore unlike many other CPDTs, this is suitable for cross-platform gaming [68]. It additionally is quite extensible allowing the addition of C and C++ libraries to your applications. Cross-compilation can be a very useful

technique allowing the ability to deal with compiled native code in the end for increased optimization.

2.3.7 Tool Analysis and Comparison

Choosing one CPDT over another can be a daunting task, available skills and project requirements will often dictate which to use when comparing the capabilities of the CPDTs. The choices are significantly narrowed when special features such as native UI's and the ability for 3D graphics are needed in a CPDT.

The runtime based CPDTs, must include the runtime with the application and in many instances create larger file sizes [34]. Cross compilation may be an effective way of completing the task; however, these approaches discussed still have limited APIs. Web based CPDTs like Appcelerator Titanium and PhoneGap offer many features however have limited platform support in some cases. The lack of native UI elements can be problematic but also provides freedom for custom designed layouts. With all of the tools studied, compatibility to use device functions such as the accelerometer, camera and notifications are present. These provide a much more integrated experience as opposed to mobile web apps where there is still little in the way of integration, particularly with notifications to users.

These CPDTs come with a variety of costs and licenses involved that are rapidly being adjusted. Some such as PhoneGap are free and open source and others like Appcelerator have free and paid tiers. The tiered model with support options in higher tiers is typical and also used by PhoneGap. As the tools have evolved, so have these pricing models which are a moving target.

With each of the cross-platform methods, adequate debugging functionality and documentation was still seen to be lacking or outdated [49]. Currently, many resources are restricted to documentation provided by the tool makers themselves so

full objective comparison is difficult to establish in these early days and requires personal experience with each CPDT.

2.4 Related Work

Little research is available comparing CPDTs. Some comparison of features has been done in [32] and [68] although they lack some depth. The CPDT evaluators used a 13 item chart in [32] to allow comparison of tool features. Many important items like storage and camera access are covered in the survey. However, neither of these reports includes performance evaluation or discussion of development practices and detailed costs.

In [47], the authors discuss many CPDTs and provide partial comparison. Their work includes discussion of native versus web-based user interface elements and the importance of well performing applications. However, the authors state that they are not concerned with the internal workings of the tools and only if the applications will be approved for the application stores as the main development need. The article discusses the lack of debugging tools in many current CPDTs and provides an 8 point feature comparison. The authors develop a simple application that creates a screen with a text label and measured the RAM usage and start time for nine CPDTs. The results are found in Figure 2.3.

The results show the quick launch time for the application built using the native Android SDK but a much slower start for other tools such as Titanium and PhoneGap. Runtime based CPDTs seem to fare the worst and have large RAM footprints. This information provides useful understanding of the possible performance differences with developing applications on the Android platform using CPDTs but does not include other platforms or more extensive tests beyond initialization. It is possible with further testing that an application may perform poorly for initialization but run remarkably well afterwards but this cannot be determined by this test.

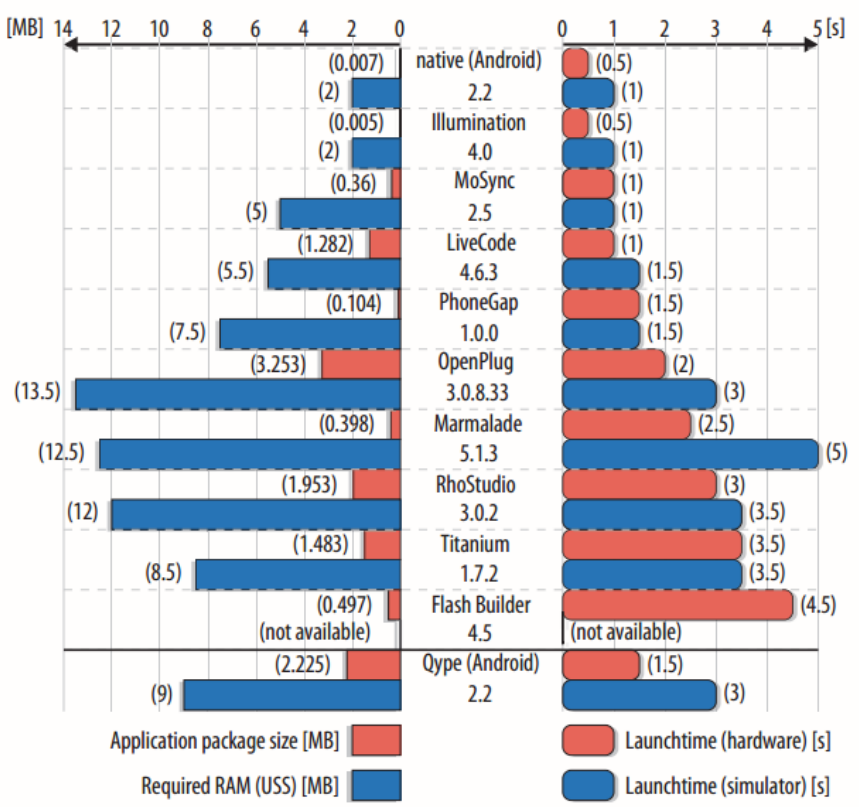

Figure 2.3: Performance Characteristics of Various CPDTs [47]

Mobile devices have been benchmarked before using a variety of approaches. Benchmarking has been around for a long time and can be applied to many forms of computational tasks where the overall capability of a processor or system is calculated and compared using complex benchmarking tools. These consist of a set of intensive tasks that measure the time it takes to complete. Currently, the PCMark suite is the most prominent desktop PC benchmarking software and uses several open source and commercial applications [61]. Using the contained test suites allow CPU, memory, graphics and hard disk performance analysis. There is no equivalent gold standard of these test suites in the mobile arena [66], although work is being completed on native benchmarks for various platforms. Some currently available are Quadrant [13] and AnTuTu [5]. Current published work in mobile benchmarking is extremely limited. This is non-existent in comparing these mobile CPDTs to their native counterparts outside of a simple test on Android.

Rather than running through pre-selected algorithms as with the aforementioned tools, the TMAPP project running on a Meebo OS based netbook used applications such as Firefox and Open Office to run through a script of tasks. The authors suggest that this method is superior to preselected algorithms as these are real world use cases [38]. However, this may be difficult to replicate with the security models on some mobile platforms. Many benchmarks available for PCs cannot be translated over to mobile devices due to restrictions from the operating system, such as running a script of opening and closing multiple applications as done in TMAPP [38].

The Rodinia Benchmark Suite [21], outlines the creation of a benchmark suite for heterogeneous systems. Included in their testing are neural networks and breadth-first search in trees. Rodinia makes use of GPU accelerators using OpenMP, OpenCL and Cuda as design elements. While not applicable to mobile uses, this is an interesting use of an application of a comparable benchmark across many devices that was shown to be successful.

A similar benchmark approach is used to compare algorithms written using various programming languages in [17], [25], [46], and [51]. Each builds a benchmark using standard algorithms that are implemented independently in each language. The resulting tests show language and compiler efficiencies much in the same way that we seek to find CPDT application performance. In [17] the author uses the Linpack benchmark suite as a guideline for the algorithms due to their long study over decades. Various sorting and complex scientific calculations are used in each of these papers to test each language and provide a common line of comparison.

A positive UX is important for all applications. Creating a consistent UX when using cross-platform development can be difficult. In [67], metrics for strong cross-platform UX are discussed. The authors find that the common themes of usability, performance, social integration and context-aware services were important for users when they are using a similar service on both mobile and desktop terminals. A CPDT

that wishes to work across device categories should ensure they have this functionality built in.

The Swerve Studio and X-forge CPDTs are discussed in [73]. These tools are focussed on cross-platform game development which is shown to have its own set of requirements that differ from those discussed in previous papers.

Current research in this area has particular limitations. When comparing CPDTs, research has focussed on narrow scales that only incorporate very few features and will not provide the required breadth necessary for decision making. Furthermore, performance benchmarks are limited to opening an application and do not provide further processing. These evaluation criteria are inadequate to draw conclusions on if there are differences in performance depending on which tool is used. More advanced testing and a larger picture of development tools is necessary to answer the greater questions of if these tools can replace native development for certain tasks.

2.5 Summary

In this chapter we have discussed the many mobile platforms that currently exist in the marketplace and the confusion and difficulties this may bring. A new generation of CPDTs have been outlined with some comparison provided.

Benchmarking procedures for mobile devices and applications are still in their infancy, however, some work has been done in [38] as well as some generally available applications. Investigation of several programming languages and how to adequately compare them are found in literature in [17] and [33].

It can be seen from an overview of previous work in this area that only surface evaluation of comparing CPDTs is available. As these tools are fairly new, benchmarks comparing them are not available at this time; however, comparable

benchmarking procedures used for other purposes have similarity and can be adapted for this purpose.

In the next chapter we will present an evaluation framework for CPDTs. The phases of evaluation and components of the framework will be outlined at a high level.

Chapter 3

An Evaluation Framework for CPDTs

The emergence of the many CPDTs in the market leave an important question unanswered; does using a CPDT to build and deploy your application impact the quality of the application? Furthermore, do the costs for development decrease with usage of a CPDT instead of building an application natively multiple times? The ability for a CPDT to have all of the features and performance, as well as decreasing costs in both monetary amounts and time for development when compared to native development are currently still open questions. This chapter presents a framework for evaluating CPDTs in order to provide recommendations to developers and determine if any trade-offs must be made when using a CPDT.

3.1 Framework

Evaluating a CPDT must be done using a variety of methods. Since there is currently little research available comparing the capabilities of CPDTs to one another and against native platforms, this framework uses standard practices adapted from other frameworks in similar domains to solve this problem.

The aim of this framework is to ensure a scientific and equitable basis can be used to determine the current state of performance for CPDTs. The scope of the framework is limited to applications that are not graphically intensive therefore it should not be applied to game development. Game development is a special case that requires different tools that are dissimilar to those used in standard application development and therefore would require different evaluation methods. Gaming uses different engines and 3D graphics where frame rates are an important factor rather than the items discussed in this framework. Similarly, this research focuses on the smartphone but is extensible to tablets in future work.

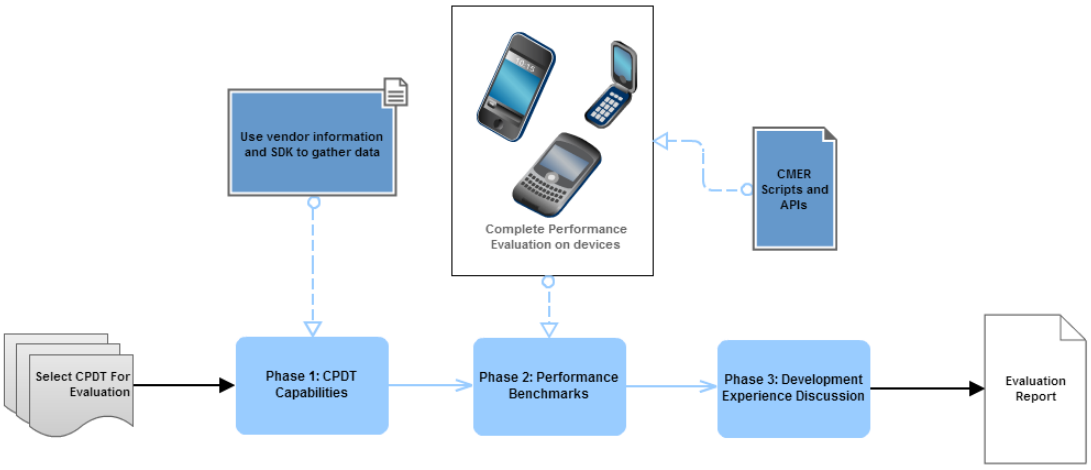

Figure 3.1: CPDT Evaluation Framework

The evaluation framework consists of three phases as shown in Figure 3.1. The first phase focuses on determining the capabilities of the framework using a large set of features some tools offer. The second phase provides a set of benchmarks that must be implemented using the CPDT and tested for completion time. This is followed by the final phase where more subjective development experience items are discussed in order to provide context and additional information to understand the result. Completing each of these phases will allow for a full view of the capabilities of the studied tool and the results should be summarized in an evaluation report.

3.2 Phase I: CPDT Capabilities

Phase I, provides an evaluation of CPDT capabilities and features, when tested against a large set of features that development platforms can potentially allow. The evaluation is checklist-based and will require the evaluator to go through the documentation and SDK of the CPDT to see if the features are supported and create a chart showing compatibility. The columns in this chart are grouped into various overarching categories of development such as access to sensors and security. Future iterations of native platforms and CPDTs may have new features included that are not currently listed. The framework is designed such that overarching categories will remain consistent while the sub-elements may need to be updated.

Some features can be considered as supported, not supported or partially supported. Within an evaluation, an overview of the features that comprise this should be included to summarize the result.

3.2.1 CPDT Basic Elements

All CPDT evaluation will bring some common basic questions such as which mobile platforms are supported and what license CPDT is under. This first section should outline these basics that must be known to understand a CPDT and include features found in current native SDKs that simply cannot be otherwise categorized. This section also includes initial and ongoing costs for using the CPDT which may be substantial in some cases.

3.2.2 Development Environment

Learning about the development environment is important when choosing which CPDT best suits one's needs. This category will include the robustness of the IDE and the type, development languages available to be used with the CPDT and debug environment. These make it easier to gauge the time required to learn new tools. Also important is finding the type of compilation used by the CPDT, many varieties exist and may yield different performance.

From a development environment, the developers need the tools to build applications that are of the same quality of native development tools. They should allow for quick debugging and may provide simple integrated methods of compiling applications. The basic characteristics of this environment must be listed and made available and any integration with specialized APIs documented.

3.2.3 User Experience

In order to create a compelling application, an appealing user interface must be created. Within this section, we will discuss the ability to have user interfaces that

meets interface guidelines provided by each platform vendor. Since these guidelines often change, the framework will be focused on confirming the available flexibility of the CPDT to be able to meet such guidelines.

3.2.4 Device Access

Devices have many internal OS information storage locations that contain device files and application data or user information. In order for some applications to best function, access to information repositories must be an available option in CPDTs. Additionally, having hooks into the lower level device functions and hardware features may enhance performance in specific networking and graphically intense applications.

3.2.5 Sensors

Mobile devices have a multitude of sensors that can gather data to feed into applications for an unlimited number of purposes. New sensors are quickly being added which native SDKs quickly adopt, however CPDTs may not have access to each of these. This section should outline what sensors are available.

3.2.6 Geolocation

Some specific sensors can be used in combination with the mobile OS to enable geolocation of the device. Even though GPS may seem like the only feature needed for adequate positioning capabilities, there are many other possibilities for finding locations. Some tools may not allow usage of each of these methods or may not have the ability to choose which method is used, this should be outlined here.

3.2.7 Notifications

Important information often must be sent to the device from external services in order to notify the application or the user of some information or to perform an action. The availability of a method to perform these actions with reliability is necessary for many

applications. Platform vendors may provide their own method of sending device notifications and integration with these or the ability to use a 3rd party solution should be specified.

3.2.8 Monetization

In order to enable monetization of an application, the CPDT must support a variety of features to give developers the freedom to make use of the pricing scheme they choose. Many free applications may want to use an advertisement platform, which can be provided by the platform vendors or a 3rd party, where paid application developers may be most interested in the ability for their users to make an in-app purchase with ease. Other monetization models may become common and could be further included. The ability to get the application in users' hands with high visibility through various outlets such as application stores should be included.

3.2.9 Security

Security is of concern for most service providers to ensure the user data is not mishandled and the application source code is not made available to those who should not have it. Methods to securely store information and retrieve it through network connections must be included in CPDTs. Additionally, the CPDT should ensure that when deployed, users do not have the ability to read application source code.

3.3 Phase II: Performance Benchmarks

Performance metrics are an important part of testing the ability of these CPDTs to perform tasks as well as natively built applications. For this phase, the benchmark tests will be developed as a suite and deployed for each CPDT by the evaluator and compared to similarly developed native tests. The difference in the result on the same device from one implementation to another is the important factor being investigated. When a new CPDT becomes available, the framework would require development of

the testing suite using the new tool, while adhering to the requirements set out in the framework.

The performance of the underlying technology and compilation optimizations of each CPDT will be tested by benchmarking an application developed using that tool. The results of this benchmark will show if the CPDT is able to create high performing applications.

As much as possible, tests will follow well-evaluated algorithms that have previously been included in other benchmarking suites for mobile devices or PCs mentioned in 2.4. Some tests may not be able to be implemented on every CPDT or on every platform due to restrictions within the mobile operating system or in the tool itself.

3.3.1 Processor Intensive Benchmarks

First we will evaluate processor intensive applications and look to the SunSpider JavaScript Benchmark version 0.9.1 for AES encryption and input validation algorithms [71]. These are two often used items on mobile devices and will be completed constantly by mobile applications. Using the SunSpider JavaScript code as a guideline for the algorithm, both the AES and input validation benchmarks will produce an output in milliseconds. The tag cloud test can also be used from SunSpider in order to process a large amount of text and determine the frequency of word usage. This can be useful when adding indexed search functionality to an application.

Another testing suite to be used for benchmarks is the Linpack for Android application based on Linpack Java [55]. This is an open source Java application that provides many tests studied over the past 40 years and used in comparison in [17]. Linpack benchmarks measure how fast the system can solve linear equations with Gaussian elimination. This will provide a score rated in Millions of Floating-point Operations Per Second (MFLOPS). This test has been shown to improve in later

versions of Java on the same hardware due to new efficiencies in the virtual machine and will provide useful insight into how the other tools perform [55].

The aim of using any of these well-studied tests is to stress test the CPU to see if code from cross-platform tools is as efficient as lower level native code.

3.3.2 Data Driven Benchmarks

Mobile applications must make use of many data sources, and often combine them into a digestible form for users. In order to simulate testing for this, local and remote databases using the CMER APIs will be accessed, data extracted and sorting algorithms used to sort large arrays of data. Potential tests can be found in [17] and [46] using the heap sort algorithm. Some tests should also use the internal sorting functions provided by the CPDT API.

The CMER APIs are provided to allow for a set of data to be pulled from our CMER server using a local connection. This is important for any remote testing to remove the irregularities based on communication through the cellular network. The implementation of these APIs can vary based on the tests that are implemented in the specific evaluation. This can be very limited or broad based on the framework implementation.

3.3.3 Device Access Benchmarks

Mobile applications have the ability to capture information from multiple sensors and save the information or use it within the application. Tests will be conducted polling the microphone and recording audio clips of random length. The time taken for recording initialization will be documented. A similar test can be conducted for the initialization and shot to shot time for the camera. Since the tests using both native SDKs and CPDTs are conducted on the same device, the characteristics of the device will not affect the result.

3.3.4 User Experience Benchmarks

One of the most important aspects for user experience with an application is responsiveness. These tests will add and remove UI components in quick succession and render new application screens and items. This will be done to perform evaluation of the ability of the CPDT to render interface components and transition quickly from one screen to the next. If a CPDT suffers in UI rendering, it will provide a poor experience for users of applications built using it.

3.3.5 Test Procedure

When ready to move into the testing phase, a set of actions should be completed. Before testing on any device, certain actions should be taken to ensure the tests are as accurate as possible. First update the device system software to the latest version including any bug fixes and patches that are available. Then a factory wipe should be performed to remove any user applications and customizations that may take CPU cycles away from the test.

After installing the benchmark application, perform a full device reset to clear any programs in memory. Furthermore, close any running background tasks that are able to be closed. Run the benchmark with the full 3 external iterations with averaged results showing on the screen and sent to the logging server. A visual description of the test procedure can be found in Figure 3.2.

When testing with multiple CPDTs, each application should be run separately and the results compiled for Phase II of the evaluation report. Once testing is complete, the evaluation report can be compiled using the development experience from developing the benchmarks, to answer the questions posed in Phase III.

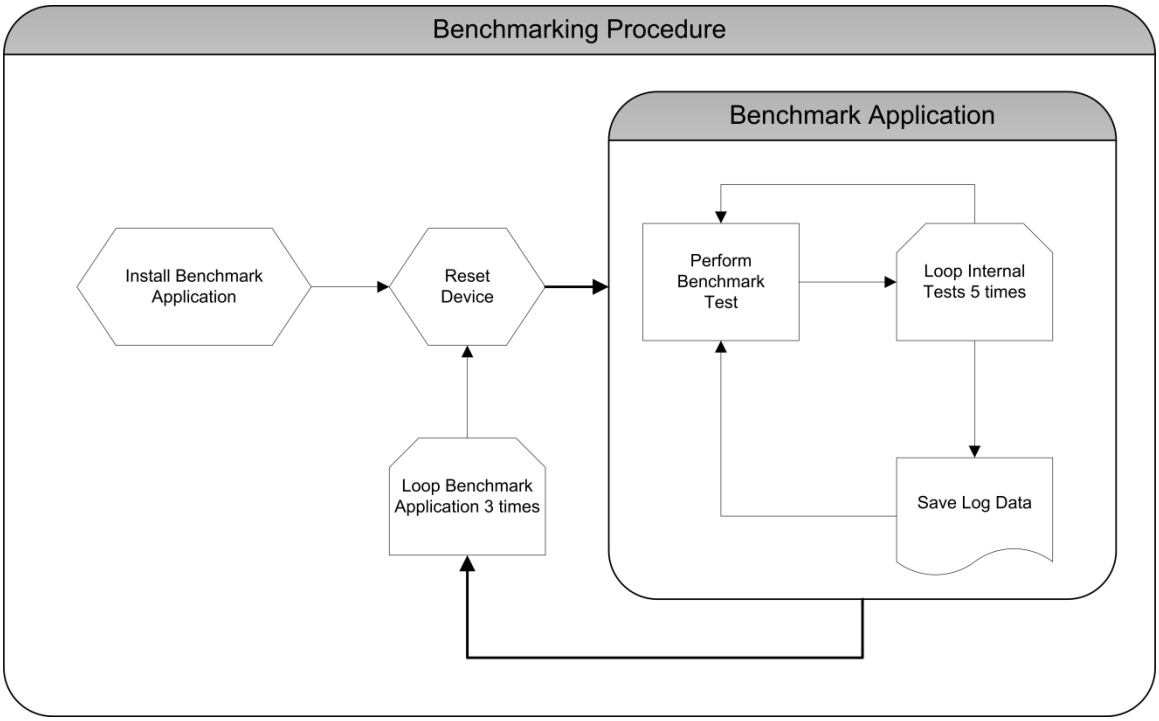

Figure 3.2: Benchmarking Procedure

Following these test procedures should ensure outliers are removed from testing for unforeseen delays when completing the test. It is important in all benchmarking to complete all tests on an equitable basis and to repeat them to find average results [21].

3.4 Phase III: Development Experience Discussion

In Phase III, the evaluator will investigate criteria that you are unable to put a measurement on in any simple way. Due to the nature of these CPDTs, not everything can be put into a measure or checklist, they instead need to use the experience gathered while completing Phase I and II to discuss parts of the tool in more depth. This section will lead discussion of some features to enable the reader to make meaningful judgments of the level of functionality provided. The discussion will be led in such a way to limit possible bias from the evaluator and should be used to provide the context to understand the results in Phase I and Phase II.

3.4.1 Tool Related Discussion

Some characteristics of CPDTs can be difficult to speak of in a chart form and require lengthy explanation. Characteristics of the tools discussing how well maintained the CPDT is for updates, new features and bug fixes should be discussed here. Additionally, further discussion of costs associated with using the tool and any ongoing costs can be included. This would include costs associated and features available for any cloud or analytics services offered.

The discussion should also revolve around the development IDE and features supported. Answer if the features and debugging methods included provide the expected level of functionality.

3.4.2 Development Experience Discussion

This section aims to use the development experience gained by completing the other phases of this evaluation framework to discuss items such as relative application size and the strength of the UI toolkit. Flexibility of interface construction can be important and should be noted.

The evaluator should attempt to discuss whether the tool allows for write once, run anywhere development or the level of customization required for different platforms. Additionally discuss the overall robustness of the development experience and provide specific examples of where using the tool is helpful and any drawbacks. Lastly, speak about the learning curve for using this CPDT in this section of the report.

3.5 Summary

Choice of which development tools to use for mobile application creation has never been greater. While building natively for each platform may provide the quickest access to new features, CPDTs have their place in the market. Through

implementation of each phase of this framework, CPDTs available today or new tools released in the future can be evaluated based on the criteria discussed in this chapter. This will provide a balanced look at the features, performance and development experience for any CPDT. Armed with this information, developers are provided with a detailed and unbiased view of tool capabilities and will be able to best decide which meets their criteria for their applications.

This framework requires one to take the overarching themes and apply them in more detail to create adequate tests for not only today's CPDTs, but future tools. The capabilities, performance metrics and experience discussion in this chapter provide the subject matter applicable to further testing.

In order to test the framework for its suitability it was applied to various CPDTs. This will not only show the strength and usefulness of the framework, it allowed for evaluation of current top tier tools in order to provide immediate guidance on the suitability of each for different applications.

In the next chapter, we will implement this framework to create specific tests that are relevant to the CPDTs on the market today. The experiments that were conducted will be outlined and discussed.

Chapter 4

Implementation and Experiments

Using the framework developed in Chapter 3, we will discuss how this framework has been implemented, in order to evaluate today's CPDTs. As the framework is designed to outline only the overarching principles, the first step is to break down each phase in detail in order to create a fair set of tests that are relevant to the development tools being chosen for evaluation.

This chapter outlines the tools and devices used for this evaluation. This provides an overview to readers of the evaluation results of the scope of this study. Following this, a detailed outline and specifications for each phase of evaluation is provided.

The implementation presented in this chapter, should satisfy the criteria outlined in Chapter 3, while implementing it and focussing on today's environment of CPDTs. The scope of the evaluation need not encompass all aspects but should provide a thorough basis of comparison for the CPDTs included in the study.

4.1 Experiment Parameters

In order to provide structure for the experiment, the following subsections will outline the tools included as part of the study, the devices used for testing and any assumptions factored into the results.

4.1.1 CPDTs and Native Development Kits

This study has been conducted using many development tools. Both cross-platform and native development kits will be included to provide a basis for comparison. The tools included and platforms supported can be found in Table 4.1. Entries marked with a dash are incompatible.

Table 4.1: Compatibility Matrix for Tools Included in Study

	Android	iOS	BlackBerry 10	BlackBerry 7
WebWorks	-	-	Compatible	Compatible
BB10 Native SDK	-	-	Compatible	-
Android Java SDK	Compatible	-	-	-
iOS Native	-	Compatible	-	-
PhoneGap	Compatible	Compatible	-	Compatible
Appcelerator Titanium	Compatible	Compatible	-	-
Adobe Air	Compatible	Compatible	Compatible	-
MoSync	Compatible	Compatible	-	-

While Adobe Air is compatible with BlackBerry 10, it is excluded from our testing due to many issues with the beta software. This can be included in later testing once the platform has stabilized. Appcelerator Titanium's BlackBerry 7 Beta software was unavailable for testing as its development has now been discontinued. The performance benchmarks in Phase II will be developed for each of the compatible tools and platform combinations. This selection of tools, allows for a wide variety of comparisons to take place with multiple tools running on each platform.

Some CPDTs offer a variety of methods for development. The Titanium development focused on application built and run using their runtime technology while MoSync applications focus on cross-compilation only. Adobe Air applications use the ActionScript, cross-platform development approach.

4.1.2 Devices

Each platform has all tests conducted on a single device using both native and cross-platform tools. The devices to be used are found in Table 4.2.

Table 4.2: Devices Used for Testing

Platform	Device Model	Software Version
Android	Samsung Galaxy S II (i9100)	4.0.3
iOS	Apple iPhone 4s	5.1.1
BlackBerry 7	BlackBerry Bold 9900	7.0.0.579
Blackberry 10	BlackBerry 10 Dev Alpha	10.0.4.197

4.1.3 Assumptions

In order to test the CPDTs, certain assumptions must be taken into account. It has been deemed adequate to test on a single device for each CPDT. The comparison completed in this framework is not on device performance, but the difference in native to CPDT performance. This factor will be little affected by the device the tests are run on as long as all testing is done on the same device. There is a small chance of a device specific bug that may affect the results of one CPDT and not the others, however, we have assumed that this is highly unlikely and completing all tests on a single device will provide the necessary results. The testing has been conducted on only the devices shown in Section 4.1.2 after completing the procedures outlined in Section 3.3.5. These procedures are sufficient to close necessary background applications and clear enough memory to complete the test fairly with minimal disturbance by other processes.

It is expected that there may be some variability to tests based on the version of the CPDT used. The version used for testing should be shown during Phase I of the evaluation and stay consistent throughout each phase. Furthermore, as different developers may implement the algorithms in slightly different ways, they should be written in a way to conform to the norms and best practices laid out for that particular CPDT. Each test implementation has the same result and steps regardless of the tool or language used.

4.2 Phase I: CPDT Capabilities

Phase I of testing evaluates the CPDT based on its feature set when compared to a set of features often found in mobile development tools. The overarching categories in Section 3.2 have been broken down into the current set of features available for each category.

In Table A.1 you will find the list of criteria that this framework will seek for a CPDT. Currently, 53 elements available in various native development tools are included to evaluate the CPDTs against. The features were selected through discussions with our industry partner and surveying the available features in each of the native tools. When evaluating a CPDT, the elements would be presented in a chart form and supported or not supported would be written in each column for that particular CPDT. Often support will only be available on certain versions of the device operating system or not fully supported. Where more information is required, context can be provided and therefore the values for the evaluation do not necessarily need to be binary.

Of the 53 elements in Table A.1, 46 can be considered as supported, not supported or partially supported. These features are indicated as belonging to this quantifiable group in the table to be classified in the results.

4.3 Phase II: Performance Benchmarks

Performance evaluation is a key aspect of evaluating CPDTs. At this point, it is unknown how well web based CPDTs perform compared to native with only anecdotal evidence available. Furthermore, similar tools using the same compilation methods may have performance enhancements from the vendor that are often touted as a reason to choose one tool over another. Not everything can be included in the performance benchmarks but tests are developed to simulate common tasks and actions. Each test is designed to allow a wide variety of tools to be able to implement

the test with an overarching skeleton program to handle the administrative tasks of running a benchmark.

The benchmark algorithms and specifications will be outlined in the following subsections. Implementation of each of these tests will occur using the standard development language for each of the CPDTs. In all cases, plugins or 3rd party additions to the tools are avoided and some code is provided.

4.3.1 Application Skeleton

This application should provide the basic features needed in order for performance evaluation tests to be completed. The skeleton refers to a simple GUI that allows a selection of a set of tests and a button to begin testing. Once all tests are completed, a summary screen is displayed with average results for the iterations of the test. All tests should be completed once and then the cycle begins again until the specified number of iterations has been reached. The detailed results are contained in a log file. These tests should be modular and can be added and deleted at will. This skeleton and the following tests can be implemented on each CPDT with the number of iterations specified by the framework in Section 4.3.1.4. Debug functionality should be provided where only one iteration is completed of each selected test to speed up development.

4.3.1.1 Interface

The interface is a simple list of tests with a run button as seen in Figure 4.1. Some changes may be necessary depending on the tool capabilities such as moving the buttons to the left on the Titanium version. During test completion, a progress indicator is displayed. This progress indicator should be for total testing progress and only updated between tests to ensure it does not interfere with results. As some tests include usage of user interface components, the user sees those before being returned to the progress screen during those tests.

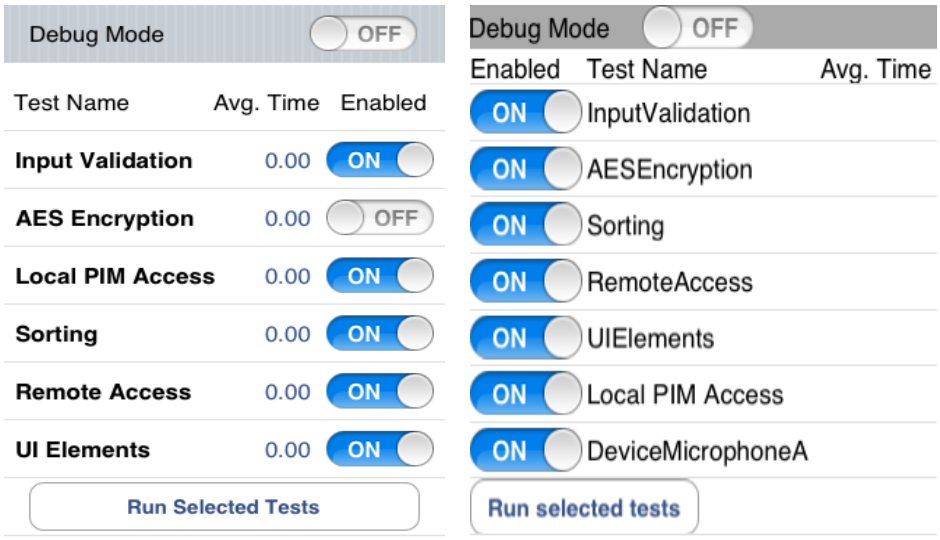

Figure 4.1: Native Implementation on iOS (left), Titanium implementation on iOS (right)

4.3.1.2 Test Modules

Each module should be self-contained with a start function called “test” that is called by the skeleton application. At the end of a test run, a set of values is passed to a function called addResult in the skeleton that handles the data. It takes three parameters: a String for the testname, a long for the starttime, and a long for the endtime. This function will save the information to be sent to the server once all tests have completed. When all iterations of all selected test cases have finished, another function is called to organize the saved data, calculate averages, add the organized and complete data to a string, and send that string to the log server. The string is sent via HTTP POST, with a parameter called “data” which equals the generated string.

4.3.1.3 Log

Two files are saved during each evaluation. These files should include a date and time prefix to not delete any previous results. These files should be saved as a Comma-Separated Value (CSV) file.

The first log file includes the raw results for each test. If a test is run with 5 iterations, the ‘testname’ will have 5 identical entries with different duration values.

The filename should be CPDTTYPE-DEVICE-DATE-TIME.csv with CPDTTYPE, DEVICE, DATE and TIME replaced at the time of creation by the server script. If the developer of the benchmark would like to save the CSV files locally, that would not affect test results and may be done. Due to device restrictions, this may not always be possible and logging on a server could be necessary. The data structure is [testname, starttime, endtime, duration]. These column names should appear in the first line followed by entries for each test. All durations are in milliseconds with one decimal place.

The second file uses the name CPDTTYPE-DEVICE-DATETIME.csv. This should only contain the header showing the structure and the average duration result for each of the tests. The structure is [testname,average]. Therefore, regardless of the number of iterations a test has in the skeleton, the final result will be the mean of those tests saved in this file.

Data is sent to the server in the following format:

DEVICE,CPDTTYPE,OSVERSION,DATE,TIME;
testname,AVERAGE;
testname,AVERAGE; ….;
testname,starttime,endtime,value;
testname,starttime,endtime, value; …;

The server will parse this information and create the log files as follows.

In folder “Raw logs”, Filename “CPDTTYPE-DEVICE-DATE-TIME.csv”

DEVICE,CPDTTYPE,OSVERSION,DATE,TIME	(test information header)
testname,starttime,endtime,value	(category header)
testname,starttime,endtime,value	(data)
testname,starttime,endtime,value	(data)

In folder “Averages/CPDTYPE”, Filename “CPDTTYPE-DEVICE-OSVERSION.csv” append the following:

DEVICE,CPDTTYPE,OSVERSION	(test info header)
testname,datetime,AVERAGE	(category header)
testname,datetime,AVERAGE	(data)
testname,datetime,AVERAGE	(data)

Implementation of the application skeleton can be found in Appendix B for the WebWorks implementation. A similar implementation is used on all other platforms. For WebWorks, all logic is in the JavaScript files and for each plugin usage is avoided.

4.3.1.4 Iterations

The tests are completed sequentially and then iterated through until the total number of iterations has been reached. For the CPDT evaluation, each test was completed 5 times with the results averaged to be displayed to the user and saved in the averages log file. Some tests may have sections that are internally repeated thus creating a large number of total iterations when the benchmark application is externally run 3 times for final results. All tests return a single value and therefore, internal iterations are either averaged, or a total duration value is taken before being returned to the skeleton.

4.3.2 Processor Intensive Benchmark Tests

In order to complete testing described in Section 3.3, processor, data intensive, device access and user experience testing will be conducted. This section will detail the test algorithms.

4.3.2.1 AES Encryption and Decryption

The objective of this test is to encrypt and decrypt a passage of text. The duration between start and finish will be provided to the test skeleton. The algorithm to be

used is the Rijndael algorithm which is the same standard AES technique used in the SunSpider JavaScript benchmark version 0.9.

To enable the quick implementation of this algorithm, the code in the SunSpider Crypto-AES test [71] can be used as a guideline to port to other platforms. The algorithm has also been implemented in Java, C and C++ with much documentation provided on the National Institute of Science and Technology website [58]. The text, encryption keys and algorithm can be found in Appendix D.

This test returns a result of the total time taken for one encryption and decryption cycle.

4.3.2.2 Input Validation

The objective of this test is to attempt to simulate testing for valid user inputs for items such as email addresses and US zip codes. The algorithm documented in the string-validate-input test which be used as a guideline from the SunSpider JavaScript benchmark version 0.9 [71].

In order to complete this test, 4000 email addresses will be generated and verified followed by 4000 zip codes. The test will look for length, illegal characters and correct structure. Completing it this many times will provide insight on if any CPDTs have difficulty comparing strings.

This test returns the total duration to complete the 4000 email verifications and 4000 zip verifications. The process for this test is outlined in Figure 4.2.

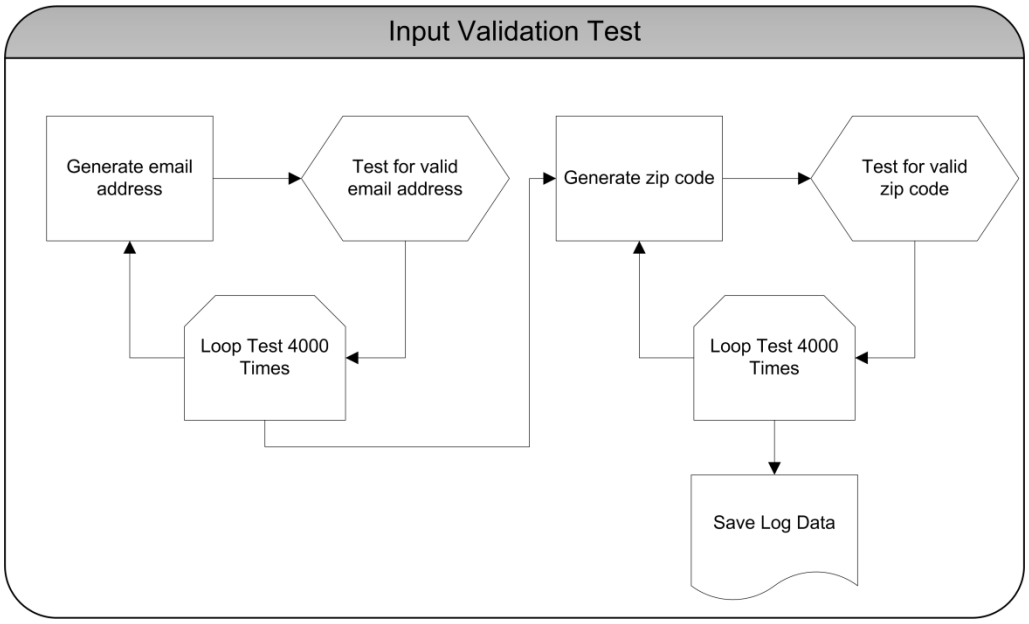

Figure 4.2: Input Validation Test Procedure

4.3.3 Data Intensive Benchmark Tests

The data intensive benchmarks are focussed on manipulation of large amounts of data in structured formats and access to the device databases. The next sections will outline these tests.

4.3.3.1 Local PIM Access

This benchmark test aims to read locally stored user data and write to the PIM (Personal Information Manager) address book. It will comprise of operations with the device contacts database. A series of steps should be taken to test the efficiency of writing data to the address book and retrieving it.

A CSV file containing 100 entries of first name, last name, phone number and email will be used that is randomly generated using the tools found in [29] and included as Appendix E.

Procedure:

1. Read local CSV file with stored contact information into memory,
2. Enter all 100 contacts to the empty PIM database,
3. Retrieve 100 contacts from the PIM database including first name, last name, phone number and email,
4. Delete all contacts from the PIM database.

This test returns the total duration for completing the above algorithm.

4.3.3.2 Remote Service Access

The aim of this test is to query a remote web service on a server and retrieve information that can be used within an application. This information will be sent to the device and response time will be measured. All testing is done via local network to reduce any network latency irregularities.

This test allows for creation of a CMER API in the form of a script that will be housed on a server running Ubuntu Server 11.04 on a machine connected to a 100 Mbps network with 802.11g Wi-Fi connecting the devices. The machine consists of an Intel Core 2 Duo E6550 2.33GHz with 2GB of RAM. A line of PHP will be used to return the text via HTTP. The text is a hardcoded string and the PHP code is found in Appendix F. The code has been simplified to an echo response but in other implementations of the evaluation, an advanced API may be required.

The benchmark test will consist of establishing a connection to this script, retrieve the string and repeating the process 100 times. The total duration for all 100 internal iterations is the returned value. This process can be seen in Figure 4.3.

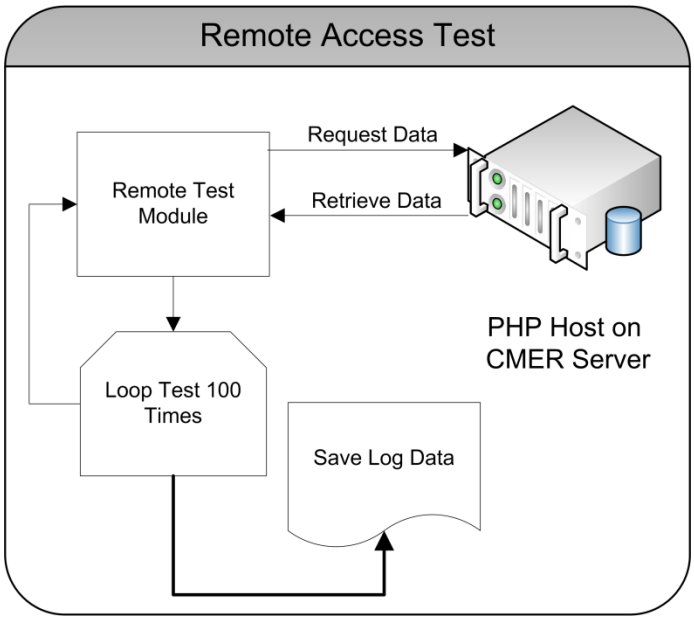

Figure 4.3: Remote Access Test Procedure

4.3.3.3 Sorting

This test comprises of generation of 2500 random integers between 0 – 2500, allowing duplicates and sorting them. A simple bubble sort will sort these numbers from lowest to highest. This is conducted using loops to compare neighbouring nodes. While not the most efficient sorting method, the high number of comparative operations makes it a good candidate to see how well different CPDTs perform. This process will then be repeated 5 times internally. The test duration should include number generation and sorting. This test is loosely designed to follow sorting benchmarks in [17] and [46]. The total duration for all 5 internal iterations is the returned value.

Excerpts of the sorting algorithm can be found in Appendix G. They show the same algorithm implemented on two CPDTs. The procedure for the test is described in Figure 4.4.

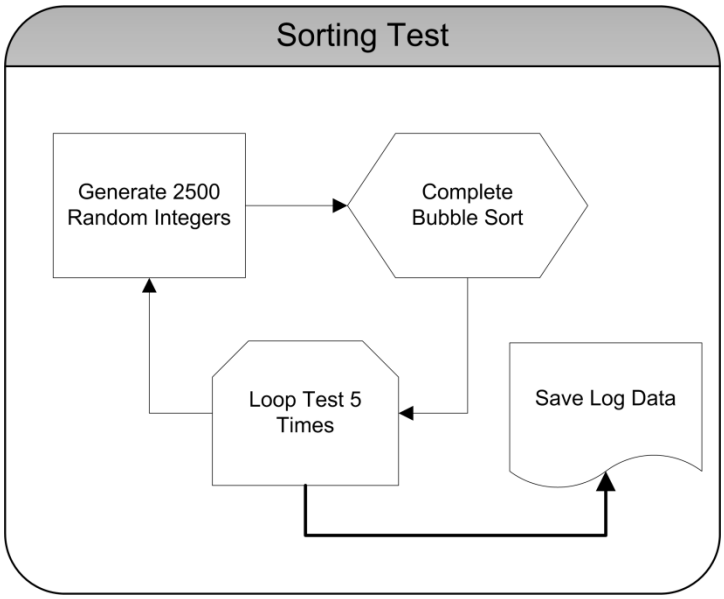

Figure 4.4: Sorting Test Procedure

4.3.4 Device Access Benchmark Test

Device access tests are focused on making use of device features through the API of the CPDT. The following test will use the microphone sensor to test the API.

4.3.4.1 Microphone Usage

This test will focus on gathering information using the microphone and any lag between the time it takes from request to result using different CPDTs.

The protocol will initialize the microphone, record 0.5 seconds of audio, which does not need to be saved and record the time taken for this process. This will be repeated and an average returned.

The procedure is as follows:

1. Record time,
2. Initialize microphone,
3. Record 0.5 seconds of audio,

4. Record end time for a result ,
5. Repeat steps 1-4 a total of 25 times,
6. Find the average of 25 internal iterations and return it as the final result.

The logged result in the test is an average of the 25 initialization durations. A start and end time are not required and can be left as 0.

4.3.5 User Experience Benchmark Tests

The user experience tests aim to provide manipulations for front facing features and UI to ensure highly performing applications. The two tests in this section provide manipulation and transition benchmarks.

4.3.5.1 UI Elements

This test will try to manage the modification of the user screen elements quickly in a sequential test and record completion time. The elements to be added include images, text boxes, labels, buttons and radio buttons. A simple interface will be created and then modified in specific steps.

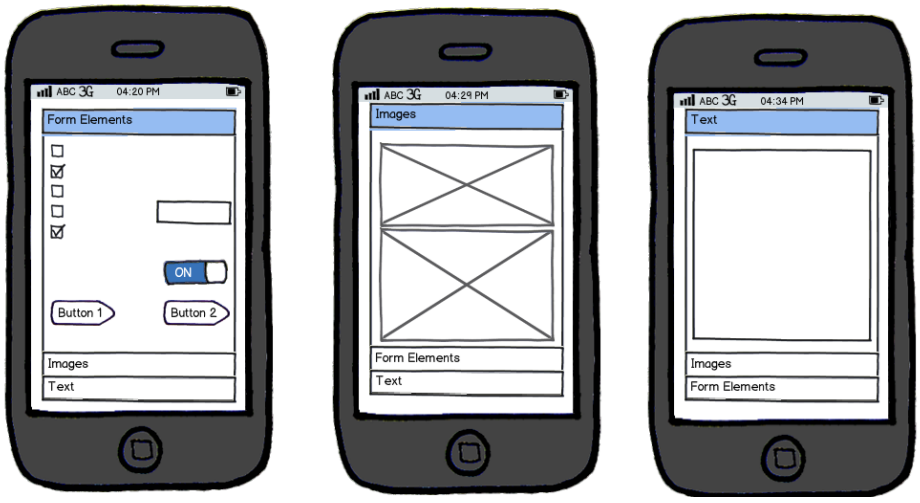

Figure 4.5: Screens for UI Elements Test

An accordion style interface layout will be used with 3 sections. The first will consist of form elements including 5 checkboxes, a text field, switch and 2 buttons. The second will have two images, one at the native resolution of the image and one that has been shrunken to fit a 20% x 20% of the screen size. These images are included with this document in Appendix G. The third screen will consist of text found in the Appendix D and is the same passage from Romeo and Juliet used for AES encryption testing.

After rendering the initial accordion style screen, we flip through the sections and add and remove elements as follows:

1. Record time,
2. Start in form elements section,
3. Switch to images section,
4. Switch to text section,
5. Delete all text and add it back, loop 20 times,
6. Switch to Form elements section,
7. Randomly select and deselect checkbox elements 500 times,
8. Delete Button 2 and re-add it with the top left pixel position being randomly selected, repeat 50 times,
9. Switch to the images section. Remove both images and add them back switching which is at its normal size and which is scaled to 20% of the screen, repeat 20 times,
10. Record completion time.

Completing these steps will be considered a single run of the test. Duration will be from before the initial screen render to the final change of the image. An implementation of this algorithm for WebWorks can be found in Appendix C.

4.3.5.2 Screen Transition

This test will flip through a series of application screens and measure the time it takes to complete the render and transition to the next screen.

The screens will look like the 3 shown in Figure 4.6. The transitions will be as follows:

A→B→A→C→A→C→B→A→B→A→B→C→C→A→B→B→A→A→A→A→B
→A→B→B→B→A→B→C→C→A→C→B→C→A→A→B→C→A→B→C→A

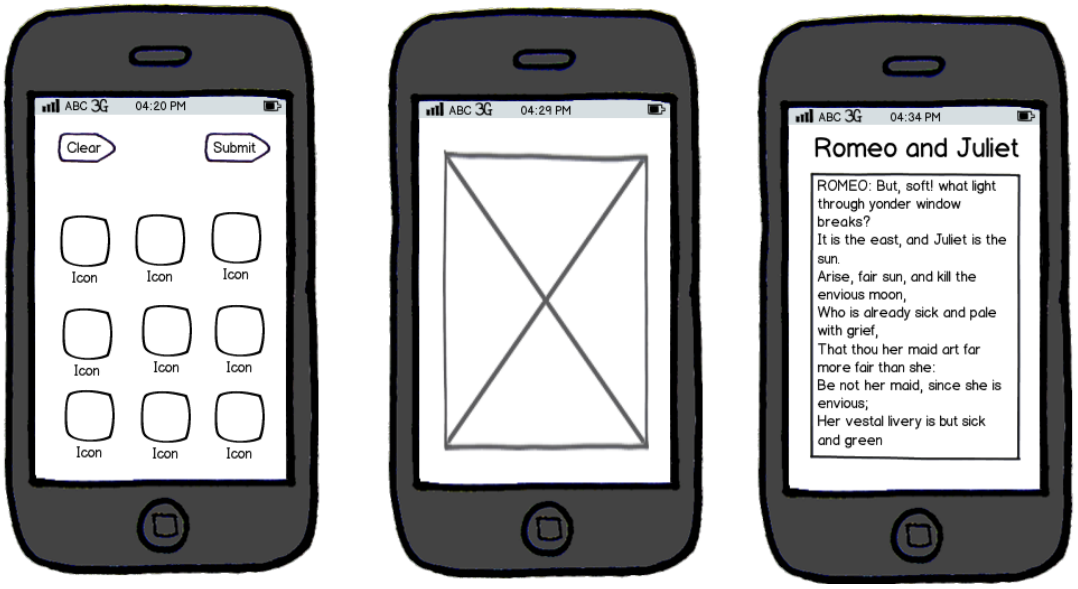

Figure 4.6: Screen A, Screen B and Screen C for Transition Test

Screen A consists of two buttons and 9 icons as seen in most application menus. The icons are enclosed in Appendix G as is the image for Screen B. Screen B contains a single image stretched to the entire screen. The final screen, Screen C, has a title and text from the AES Encryption test found in Appendix D.

If possible, the CPDT should cache the screen so it does not incur extra loading. If there is no option for this, the default is used. A controller object will control the screen transitions according to the diagram in Figure 4.7. This object will also measure the time.

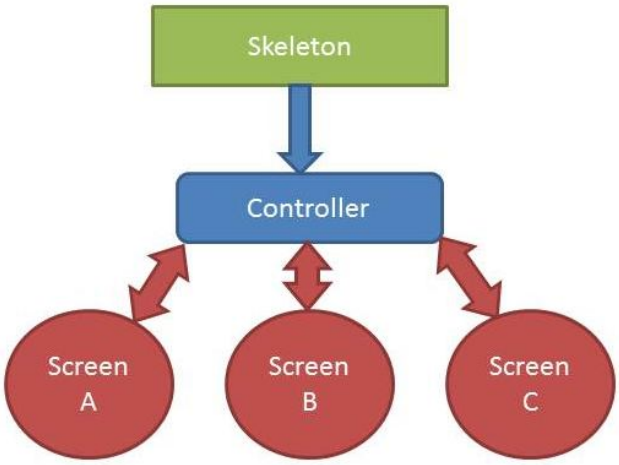

Figure 4.7: Control Structure for Transition Test

For a full completion of the test, all 40 transitions must be completed and duration recorded.

4.3.6 Test Implementation

Due to a variety of factors, for the purposes of this evaluation, not every test was implemented using each CPDT. The tests that are included in this evaluation are shown in Table 4.3. Enough tests are included in the evaluation to show the strength of the framework to provide a basis of comparison of CPDTs and draw conclusions about the framework and strength of certain tools. Larger studies can be completed and are discussed as part of future work in Section 6.2.

Table 4.3: Tests Implemented in the Study

	Android	iOS	BlackBerry 10	BlackBerry 7
WebWorks	-	-	Input Validation, Sorting	AES, Input Validation, Remote, Sorting, UI Elements,
BB10 Native SDK	-	-	Input Validation, Sorting	-
Android Java SDK	Input Validation, PIM, Remote, Sorting, Microphone, Transition	-	-	-
iOS Native	-	Input Validation, PIM, Remote, Sorting, UI Elements,	-	-
PhoneGap	AES, Input Validation, PIM, Remote, Sorting, UI Elements, Transition	AES, Input Validation, PIM, Remote, Sorting, UI Elements, Transition	-	AES, Input Validation, Remote, Sorting, UI Elements,
Appcelerator Titanium	AES, Input Validation, Sorting, Transition	AES, Input Validation, PIM, Sorting, UI Elements, Transition	-	-
Adobe Air	Input Validation, Remote, Sorting, Microphone, UI Elements, Transition	Input Validation, Remote, Sorting, UI Elements, Transition	-	-
MoSync	Sorting, UI Elements, Transition	Sorting, UI Elements, Transition	-	-

4.4 Phase III: Development Experience Discussion

Phase III consists of discussion over key features and how well implemented they are in accordance to the framework in Section 3.4.

4.4.1 Tool Related Discussion

Each CPDT has its own unique characteristics that may be difficult to speak of in short. It has been thought in the past that using CPDTs over native development slows the adoption of new features into applications. This can be a problem for developers when choosing a tool. An evaluator must discuss in this section the lag time between major SDK enhancements for any of the mobile platforms and when these new features were incorporated into the CPDT API.

Furthermore, current tools vary wildly in their monetization models. Some tools are free to use but charge for support, some do not offer any forms of support and others charge for development, support and continuous deployment. Phase I required information to be gathered about the costs of the tools and the ongoing cost for using the CPDT. In this section, the evaluator is to expand on one-time or subscription costs and current support options. Questions that should be answered are if there are free options, if push service or analytics is provided and at what cost, build service costs and miscellaneous costs such as developer accounts with platform vendors.

Lastly, using the information gathered about the IDE and languages the tool uses for development, the evaluator will discuss the flexibility and popularity of these development languages and tools. The focus should be on if they are full featured and flexible as well as if the skills for development using them are readily available amongst current developers. New tools can be extremely useful, however, there may be trade-offs given the amount of time it takes to first learn them which will be discussed as part of the development experience.

4.4.2 Development Experience Discussion

The aim of this section is to use the developers personal experience implementing this framework in Phases I and II to further discuss the development experience one would face while using the CPDT selected for study. First, discuss the relative application size using the CPDT; does this tool package a large runtime with the

application? An item that can vary greatly between platforms is UI construction. The UI tools provided for use of the CPDT should provide a level of flexibility where devices with very low and very high resolution displays should all display the application in a usable form. Having the interface resemble and perform as well as the native UI for the platform and follow design guidelines will not only help the user have a more intuitive experience, but may be a requirement for submission to application stores. The UI tools should allow for some form of simple language customization for easy deployment in foreign languages and have the ability to look compelling and well designed. Some devices do not have a touch interface, also discuss if the CPDT allows for a cursor or touchpad rather than touchscreen implementation.

Furthering the discussion from above, many CPDTs simply do not have a write once, run anywhere structure and are more akin to write once, customize everywhere. What level of customization is required to have applications run well on all supported platforms? This is a key question as it may take an overhaul of the UI for each platform and without MVC design ability; this may cause other problems with the application. Outlining this is a vital element to know for any developer looking to go cross platform.

After developing the benchmarks and looking extensively into the features of the studied CPDT, the evaluator should now have the experience to discuss the robustness of the development tools provided for debugging and testing applications as well as the overall ease and speed of development when compared to directly using native SDKs. This piece should summarize experiences and use specific examples of where each style of development excels. The evaluator should refrain from drawing conclusions on other CPDTs and instead provide examples relating only to the tool studied to allow other studies to draw conclusions when looking at the spectrum of tools using the reports developed with this framework.

4.5 Summary

This chapter has described in detail the tests that will be conducted to evaluate the CPDTs. It has outlined which CPDTs and native environments will be tested and the exact test procedure. The structure of the skeleton application to produce the log files has been discussed so it can be replicated.

While the framework allows the tests to change over time according to developments in the marketplace, the test parameters in this chapter are used to conduct the testing of the outlined tools, of which the results will be found in the following chapter.

Chapter 5

Results and Evaluation

By implementing the tests using various CPDTs, each phase of evaluation has provided significant information regarding the tools strengths and weaknesses. These results are presented in this chapter. The results will show the tool and platform used for the test and the result. Following this, analysis is provided for each test and overall analysis for the full evaluation in Section 5.4.

5.1 Phase I: CPDT Capabilities

As explained in Section 4.2, the first phase of evaluation is careful analysis of a CPDTs documentation and APIs to discover its feature set. Table I.1 in Appendix I, provides a summary of this feature evaluation for all native and cross-platform tools in the study.

Much of the information contained in Table I.1 is binary and allows for simple categorization. For example, does the tool provide a build service? Other features have more complex explanations provided. Table 5.1 describes the number of supported features for each CPDT out of the possible 46 that can be considered in such a way.

Table 5.1: Number of Supported Features

	Supported	Partially Supported	Not Supported	Indeterminable
PhoneGap	16	5	21	4
Appcelerator Titanium	31	0	12	3
Adobe Air	27	5	9	5
MoSync	15	9	7	15

Table 5.1 shows the feature set is diverse with only some tools covering the majority of features. Appcelerator Titanium has the most support in our feature evaluation with Adobe Air being a very close second with many partially implemented features. In these respects, PhoneGap and MoSync lag behind the other two CPDTs. It should be noted that no native development tools have all 46 features; however each have a subset of them.

5.2 Phase II: Performance Benchmarks

Features do not tell the full story when it comes to evaluating these tools so while we now know what each is capable of, it is important to test how well some of these features work. Each of the performance tests in Section 4.3 have been implemented on some or all platforms. Not every test was completed on each platform due to security, feature or other restrictions. This is a barrier faced by cross-platform tools that can be problematic when not choosing the tool fit to purpose explicitly.

Each of the following sections will provide results and some comment on each of the tests. These tests consist of the specified test in Section 4.3 repeated 5 times and averaged for a single run. Each test run is then repeated 3 times, for a final average that is listed in the results. This means that at a minimum, tests are run 15 times for each platform in order to find a baseline average result. All test results are shown in result matrices for tools and platforms. Since not every tool supports all platforms, some places will be intentionally left blank.

It is important to note that comparisons should only be made for different CPDTs using the same platform and therefore, comparing PhoneGap on Android versus Titanium on iOS is not a fair comparison. This is due to dissimilar hardware which makes the comparison invalid.

5.2.1 Processor Intensive: AES Encryption

In order to test this with an exact version of the same algorithm, this test was only completed using WebWorks, Titanium and PhoneGap to test their individual optimization techniques. Testing was done on BlackBerry 7, iOS and Android.

The result matrix in Table 5.2 will show the average milliseconds taken to complete the test.

Table 5.2: AES Encryption Test Result Matrix (milliseconds)

	Android	iOS	BlackBerry 7
WebWorks	-	-	108.5
PhoneGap	82.9	275.9	145.1
Appcelerator Titanium	39.3	248.3	-

We can see from the results that Appcelerator is significantly faster on Android and somewhat on iOS. In all tests, PhoneGap lags behind the others even with identical code. WebWorks, being from the platform vendor itself, may have performance enhancements but this is speculation. Appcelerator Titanium claims to have a compilation engine that boosts JavaScript performance and from these results, it may be true. It can be seen that the purported optimization techniques may hold water in this case as Titanium leads the pack by a significant margin.

5.2.2 Processor Intensive: Input Validation

This test was developed using each CPDT included in the study with the exception of MoSync because of its lack of regular expression comparators. The test has been completed on BlackBerry 7, iOS, BlackBerry 10 and Android using both cross-platform and native development tools. Table 5.3 shows the results of the input validation benchmark with up to 4 comparisons for each platform.

Table 5.3: Input Validation Test Result Matrix (milliseconds)

	Android	iOS	BlackBerry 10	BlackBerry 7
WebWorks	-	-	104.2	172.5
BB10 Native SDK	-	-	141.8	-
Android Java SDK	1463.5	-	-	-
iOS Native SDK	-	311.9	-	-
PhoneGap	128.2	257.9	-	339.8
Appcelerator Titanium	55.3	213.6	-	-
Adobe Air	549.3	1001	-	-

The results show quite clearly that there is a correlation with performance and using certain tools. The BlackBerry 10 results appear quite close but BlackBerry 7, iOS and Android have an extremely large range. Adobe Air in this test seems to perform fairly poorly. It is commonly expected that native tests perform best, however it can be seen that Android's Java SDK which is used for most development on the platform, actually fairs worse than all CPDTs.

Along with the results in Table 5.3, we can look at the individual tests that make up the averages in that chart. Figure 5.1 shows the 4 tools on the Android platform graphed for each of the 15 test runs. Each produces fairly consistent results with some variability.

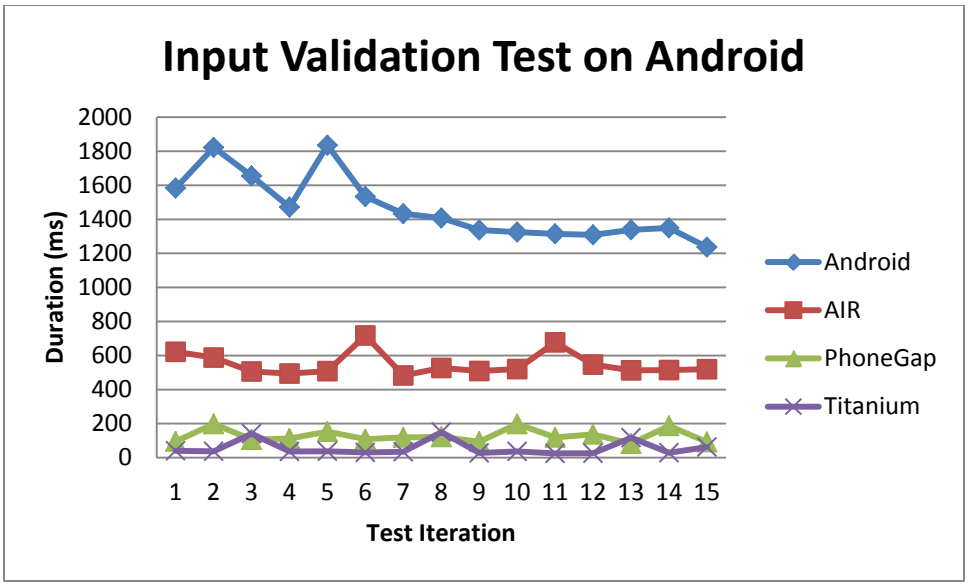

Figure 5.1: Input Validation Test Results on Android Platform

The median values and standard deviation are found in Table 5.4, confirming the relatively low variability between tests. The low variability confirms that the consistency of the test, and that we are using an adequate number of averaged iterations.

Table 5.4: Test Variability Statistics for Input Validation on Android

	Mean	Median	Standard Deviation
Android Java SDK	1463.5	1408	186.7
PhoneGap	128.2	119	38.7
Appcelerator Titanium	55.3	36	43.2
Adobe Air	549.3	519	70.2

It is important to know the significance of the values and see if the difference in the means is statistically significant. To do this, the T-test will be used where the raw data for the 15 test runs are compared to see if it is possible that the difference in the mean value is simply by chance.

In order to test this, we must choose our null hypothesis. For this we have selected: $H_0: \mu_{TOOL1} = \mu_{Tool2,}$ i.e., that the two means are equal. Our alternate hypothesis would be stated as $H_A: \mu_{TOOL1} \neq \mu_{Tool2.}$ A high level of confidence in the answer is required to prove the difference in mean is significant so a level of 99% is used, therefore with a α of 0.01.

Table 5.5: T-test Calculations for Android Native and PhoneGap Means on Android

	Android Native	PhoneGap
Mean	1463.47	128.2
Variance	34843.56	1495.89
Observations	15	15
Pooled Variance	18169.72	
Hypothesized Mean Difference	0	
df	28	
t Stat	27.13	
P(T<=t) two-tail	1.20043E-21	
t Critical two-tail	2.76	

When observing the difference in the Android Native test data to the PhoneGap tests, we can find a P value of 1.2×10^{-21} as seen in Table 5.5. This means that as this number is less than 0.01, we are able to reject the null hypothesis and adopt the alternative hypothesis. With a high degree of confidence we can see that the two tests show a statistically verifiable difference in the time for completion. Similar tests are shown in Table 5.6 to Table 5.9.

Table 5.6: T-test Calculations for iOS Native and Titanium Means on iOS

	iOS Native	Titanium
Mean	311.85	213.6
Variance	3764.73	14.54
Observations	15	15
Pooled Variance	1889.64	
Hypothesized Mean Difference	0	
df	28	
t Stat	6.19	
P(T<=t) two-tail	1.09978E-06	
t Critical two-tail	2.76	

Table 5.7: T-test Calculations for BlackBerry 7 WebWorks and PhoneGap Means on BB7

	WebWorks	PhoneGap
Mean	172.5	339.8
Variance	118.37	1969.03
Observations	15	15
Pooled Variance	1011.79	
Hypothesized Mean Difference	0	
df	29	
t Stat	-14.63	
P(T<=t) two-tail	6.33709E-15	
t Critical two-tail	2.76	

Table 5.8: T-test Calculations for BB10 Native and WebWorks Means on BB10

	Native BB10	WebWorks
Mean	141.8	104.23
Variance	134.31	404.87
Observations	15	15
Pooled Variance	269.59	
Hypothesized Mean Difference	0	
df	28	
t Stat	6.27	
P(T<=t) two-tail	8.96616E-07	
t Critical two-tail	2.76	

Table 5.9: T-test Calculations for PhoneGap and Titanium Means on Android

	PhoneGap	Titanium
Mean	128.2	55.33
Variance	1495.89	1863.24
Observations	15	15
Pooled Variance	1679.56	
Hypothesized Mean Difference	0	
df	28	
t Stat	4.87	
P(T<=t) two-tail	3.9653E-05	
t Critical two-tail	2.76	

Each of these tests shows the significant difference between the two tested values of the native platform tools and the CPDTs. In Table 5.9 we additionally see that there is a statistically significant difference in the PhoneGap and Titanium results on Android as well. This shows that the values on various CPDTs are also very different.

Similar testing was conducted for the other combinations and showed similar significance.

5.2.3 Data Driven: Local PIM Access

The test of access to local contact information posed some difficulties with some APIs not being available or they did not allow interaction as we would like. On Android, Appcelerator requires user interaction to be able to access any contact data and therefore was unable to complete the test. BlackBerry WebWorks seemed to produce errors when testing this feature and was subsequently dropped from the test.

Once again we are seeing the native Java on Android tests perform worse than the CPDT, which is quite promising for the prospects of using cross-platform development. This however, is only for Android; on iOS, the Objective C native version continues to provide much quicker access to contact information by a very large margin.

Table 5.10: Local PIM Access Test Result Matrix (milliseconds)

	Android	iOS
WebWorks	-	-
Android Java SDK	33176	-
iOS Native SDK	-	1154.6
PhoneGap	23396	9274.6
Appcelerator Titanium	-	4574.3

5.2.4 Data Driven: Remote Service Access

With the remote access test, we aim to see if there is a lag in establishing connections to our local server to download data. It does seem by the results in Table 5.11 that there is variability based on platform for connecting to this server. All testing was

done via a local network on a server with zero utilization therefore removing those factors.

As we've seen in other tests, Adobe Air seems to lag behind the competition and iOS native tests continue to outperform all others. Similarly, WebWorks on BlackBerry 7 has outperformed PhoneGap, once again using nearly identical code. The delay between PhoneGap and native applications is less apparent in this test than some others, showing that the network access lag may not be a large problem for a developer who chooses to use PhoneGap.

Table 5.11: Remote Service Access Test Result Matrix (milliseconds)

	Android	iOS	BlackBerry 7
WebWorks	-	-	685.5
Android Java SDK	1476.8	-	-
iOS Native	-	1156.3	-
PhoneGap	1141.6	1890.3	986
Adobe Air	4204	3485.3	-

5.2.5 Data Driven: Sorting

The sort test is another that can provide a wide view of many tools due to the simplicity of the approach. With that said, this test introduced the largest range we have seen in testing especially on iOS. This version had PhoneGap and Appcelerator running upwards of 38 times slower than native. Both of these tools rely on a JavaScript engine to complete computation, explaining why they both had similar results in this test.

Testing on Android produced an interesting result with MoSync. While it performed very quickly on iOS, it was by far the slowest on Android. This type of variability and leaves the developer with a lack of confidence for the performance of

their application across all platforms. Even with the cross-compiled code it provides, MoSync's Android performance is dissimilar to code from Android's Java SDK. This may suggest that cross-compilation may not be the best method of cross-platform tool.

Table 5.12: Sorting Test Result Matrix (milliseconds)

	Android	iOS	BlackBerry 10	BlackBerry 7
WebWorks	-	-	536.8	629.7
BB10 Native SDK	-	-	850	-
Android Java SDK	802.3	-	-	-
iOS Native	-	624.3	-	-
PhoneGap	1290.3	23048.8	-	3365.7
Appcelerator Titanium	804.1	20550.2	-	-
Adobe Air	3755.3	8991.7	-	-
MoSync	12132.5	516.1	-	-

As seen in the Input Validation Test in Section 5.2.2, statistical analysis can be conducted on the results to test if they are significantly different. In addition to the T-test, the ANOVA can be used to compare the results of each CPDT for each platform. The advantage of using this test is the ability to compare multiple means rather than just two as in the T-test. As in the case from Section 5.2.2, we will be using an α of 0.01. The same null and alternative hypothesis applies. The calculations for the ANOVA function are summarized for iOS and Android in Table 5.13 to Table 5.16.

These tables show the summary statistics for the raw data and the variance based on the 15 test runs. The second chart for each platform shows the calculations needed for the ANOVA function. These are the sum of squares (SS), degrees of freedom (df),

mean square (MS), F value, P-value, F critical. The important value to look at at the end of the calculation is the P-value.

Table 5.13: Summary Statistics for ANOVA Calculations on Android

Groups	Count	Sum	Average	Variance
Titanium	15	12062	804.13	1043.27
PhoneGap	15	19354	1290.27	77909.5
Native	15	12035	802.33	12863.81
Air	15	56330	3755.33	31042.81
MoSync	15	181987	12132.47	94741.12

Table 5.14: ANOVA Calculations for Android using data from Table 5.13

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	1405240419	4	351310104.8	8072.36	1.9221E-92	3.60
Within Groups	3046407.07	70	43520.10095			
Total	1408286826	74				

We can see in Table 5.14 a P-value of 1.922×10^{-92}, far smaller than our alpha of 0.01. This allows a rejection of the null hypothesis and strong support of the alternative hypothesis. This means that the average difference in the Android time taken for test completion using each tool is significantly different. Also of note is the enormous difference of the F value and the critical similar to the P-Value and alpha showing the strong support for the alternative hypothesis.

We can also see a P-value of 7.59×10^{-152} in Table 5.15 when looking at the iOS results supporting the same alternative hypothesis. Similar results are expected for all other tests and additional statistical analysis can be completed in future work.

Table 5.15: Summary Statistics for ANOVA Calculations on iOS

Groups	Count	Sum	Average	Variance
Titanium	15	308253	20550.2	6439.46
PhoneGap	15	345732	23048.8	8767.03
Native	15	9364	624.3	641.37
Air	15	134875	8991.7	5433.95
MoSync	15	7742	516.1	17.27

Table 5.16: ANOVA Calculations for iOS using data from Table 5.15

Source of Variation	SS	df	MS	F	P-value	F crit
Between Groups	6864873598	4	1716218400	402885.58	7.5962E-152	3.60
Within Groups	298187.12	70	4259.82			
Total	6865171785	74				

5.2.6 Device Access: Microphone Usage

The microphone usage test was unable to be completed with most tools due to the requirement for user interaction or lack of relevant APIs. The test was completed natively for Android and using Adobe Air and showed once again Android's Java SDK being outperformed by a cross-platform rival.

Table 5.17: Microphone Usage Test Result Matrix (milliseconds)

	Android
Android Java SDK	1193.6
Adobe Air	520.3

Using the Android version, in order to complete the test, the MediaRecorder function was used providing us the controls for recording with parameters seen here.

```
MediaRecorder recorder = new MediaRecorder();
recorder.setAudioSource(MediaRecorder.AudioSource.MIC);
recorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
recorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
recorder.setMaxDuration(500);
```

Unlike many other platforms, Android Native allowed the max duration to be set so the recording can begin and stop programmatically, allowing the test to be completed.

5.2.7 User Experience: UI Elements

The user interface tests are interesting because it can be difficult to ensure the actions are being completed before moving on to the next. Table 5.18 shows the MoSync tests with an asterisk because after approximately 250 checking and unchecking cycles of the checkboxes, the application will throw a MoSync panic. It does not provide any information on what the panic situation is and the test could only be completed with the checkbox portion removed. Android's Java version also ran into issues with some UI elements forcing it to be dropped from testing.

All other tools that were able to complete the test had varied results again having an enormous range. Adobe Air provides the most interesting data and some concern. The test is so fast that it seems it may not have completed all steps however, it can be programmatically paused at any point, and it shows the correct state. Air has a different UI model than other platforms and it seems in manipulating that UI, Air shines. PhoneGap once again fared the worst on all platforms.

Table 5.18: UI Elements Test Result Matrix (milliseconds)

	Android	iOS	BlackBerry 7
WebWorks	-	-	11388.7
iOS Native	-	748	-
PhoneGap	9350.9	12161.8	17582.9
Appcelerator Titanium	-	1571.7	-
Adobe Air	75.7	54	-
MoSync	5618.1*	1624.9*	-

With the major difference between the Titanium and PhoneGap CPDTs being their UI approach, it seems from this test that using native UI components has allowed for better performance. While still behind native and Adobe Air, on iOS, Titanium took approximately half as much time to complete the actions as PhoneGap. This begins to show that this method of blending cross-platform JavaScript with native tools may be a strong solution to the cross-platform question.

5.2.8 User Experience: Screen Transition

The final test requires 40 screen transitions. Similar to what was seen in Section 5.2.7; Adobe Air was orders of magnitude faster than the others. Careful considerations were taken to ensure the result was accurate and it seems to be so. The result is so quick that nothing is seen on screen but again can be stopped midpoint to see the completed actions.

On Android, PhoneGap with its HTML based UI seemed to provide the quickest transitions giving the widely compatible UI strong indicator of performance. Titanium uses much of the same components as PhoneGap but uses native UI tools; seems to have been slowed quite a bit.

Table 5.19: Screen Transition Test Result Matrix (milliseconds)

	Android	iOS
WebWorks	-	-
Android Java SDK	3814.2	-
PhoneGap	722.9	1395.5
Appcelerator Titanium	5717.1	1767.8
Adobe Air	31	11.3
MoSync	735.5	922.9

5.2.9 Overall Result

The testing showed quite a varied result with many CPDTs being best at certain actions. Table 5.20 shows in how many tests the development tool performed best or second best. Since not all tools were available in the same number of tests, a percentage is shown based on the total number of tests it competed in.

Table 5.20: Overall Results of Performance Evaluation

	Total Tests	1st	2nd	Percent First
WebWorks	7	6	0	86%
BB10 Native SDK	2	0	2	0%
Android Java SDK	6	1	3	17%
iOS Native	5	3	1	60%
PhoneGap	19	2	12	11%
Appcelerator Titanium	10	4	3	40%
Adobe Air	11	5	0	45%
MoSync	6	0	3	0%

5.3 Phase III: Development Experience Discussion

During the evaluation process, much experience has been gathered for developing using these CPDTs. Many bugs, irregularities and promising features have been uncovered in each tool that makes it useful for developers to use. Since we are evaluating 4 such tools, the discussion will be broken into the respective subsections and discuss topics mentioned in the framework.

5.3.1 PhoneGap

PhoneGap is a relatively useful development tool, however with no IDE with debugging; it became difficult to test the applications. PhoneGap provides much core functionality that more or less worked as expected for each of the platforms. The Ripple emulator works to test some functionality, but as it is simply a chrome frame; the UI of an application looks very different than what comes on the device. The emulator is also only compatible with version 1.0 which is behind the latest 2.0 release currently.

PhoneGap Build was the major differentiator with this tool. However, using it means that there are certain assumptions implied on applications and certain limitations as well. PhoneGap Build currently has no support for plugins, meaning that any missing functionality in PhoneGap cannot be implemented by developers. Support for plugins is on PhoneGap Build's roadmap, so developers may see this available in the future.

The open source nature of PhoneGap allows others to make use of its core functionality and build the missing components to make it a more robust tool. This is beginning to happen with IBM Worklight and is promising for this CPDT.

PhoneGap is quickly adapted to the latest changes in mobile platforms through the community support and its rapid development cycle. It was seen that all input types

were available and interface construction was limited to web technologies that were not the best choice for mobile applications. Usage of the tools is free with charges for support giving it an edge in that respect.

5.3.2 Appcelerator Titanium

Titanium shares many similarities with PhoneGap but from the start we noticed that you cannot have one set of code for Android and iOS. Segments of code and UI elements must be developed independently with only a central section of the code being cross-platform. It is an interesting approach and seemed to work well even with the extra work. The trade off to allow native UI components appears to be worth the extra effort needed for a consistent UI with all of the features of native tools.

The Titanium SDK encountered quite a few bugs and issues on Windows and Mac. It does not seem to be currently compatible with the latest release of Xcode but when running, worked well. Some tests were not possible such as the Android version of the PIM test from Section 4.3.3.1. The APIs do not provide direct access to contacts without user interaction, limiting what can be done with them. Further, we found they did not allow standard JavaScript HTTP requests and instead had their own implementation that our server had some issues with.

Titanium is a good development tool with many developer debugging and other tools available but many bugs and limitations in this early release of the product, shows it lacks maturity. The Titanium developers have stated that they will update the platform within 30 days of new features being released, however that cannot be verified. Additionally, the costs of using Titanium can be substantial if you require any of the enterprise level features. Costs are not generally available and are specific to each application and the agreement between the developer and the sales team.

5.3.3 Adobe Air

Air is a fairly easy tool to learn for the many Flash developers in the industry. It is a simple and natural progression of the Flash platform which allows for cross-platform applications. It seems that some of the native Flex components performed very slowly, but alternative Flash libraries were available to solve this. Similarly, the drag and drop design view does not perform very well or consistently across platforms so creating components programmatically may be the best option in many cases.

Using ActionScript to develop the UI provides very quick responsiveness for the visual components and can be compiled into a SWC file, allowing them to be rendered quickly. The UI tests showed that this was effective and Air showed a commanding lead. The UI allows for multiple input methods and quite a bit of flexibility.

Air is not updated as frequently as other platforms and it seems many of the functions we required were missing from the ActionScript libraries. Development for Adobe Air is free; however, some of the advanced tools come at a price. These tools are well matured and assist with the development process.

The overall experience using Air was painless. Most APIs were available with only a few tests not being possible. ActionScript is a powerful language and provides very descriptive warnings to prevent issues. While the ActionScript component was somewhat limited when compared to using Air with Flex, it provided a satisfactory experience for cross-platform development. Air while adequate, relies heavily on using platform specific code when not using ActionScript. This is somewhat limiting in our aim for creating cross-platform tests and not everything was possible.

5.3.4 MoSync

MoSync while seeming great in the feature comparison did not live up to expectations. The tool had many bugs with very little information provided to

describe what exactly had gone wrong so while it may be a developer issue, it is more difficult to discover this than on other platforms. Issues with the lack of regular expressions, using checkboxes and asynchronous methods prevented MoSync from being included in many benchmark tests.

While a development environment is provided, installing on to the emulator seemed to be much slower than directly using a device both with slow compilation. The IDE provided is not very good for C and C++ coding and while its interface is somewhat robust, code completion runs fairly inaccurately. The setup instructions were often unclear and it seemed that the iOS compilation would fail arbitrarily where if you try compiling the same code again, it may work.

Overall, MoSync development environment is lacking ripeness and it seems that the cross-compilation methods it provides did not offer any performance boost. The interface components were robust and free tiers are available, but the stability of the environment is main issue. MoSync offers many forms of application development where we focussed solely on cross-compilation using C code. Other methods may fare better.

5.3.5 Native Tools

When building the benchmarks some experience with native development tools was also collected. Generally, native tools provided a better experience for development with some CPDTs coming close like Adobe Air. Native tools on iOS and BlackBerry also seemed to perform better in testing while being easier to develop for.

With Android, we had issues with it believing the application was unresponsive while completing certain tasks that required long processing times and restricted updating of the UI. The auto-building feature, debugging and integration of the Android SDK saved quite a bit of time when compared to the CPDTs used. This made it simple to find underlying errors and resolve them quickly.

With iOS, the native development environment uses Objective C, a fairly complex language with unusual syntax. Performance and the speed of testing and debugging were admirable with native tools however it had difficulty using regular expressions. The interfaces that can be built are quite friendly, however; using the storyboard to make them has a steep learning curve.

BlackBerry 10 development tools are still in their early stages which were shown by it missing important things like a contacts API. During compilation, it often mentioned unresolved inclusions when they were added according to the specifications and the compiled file worked correctly. The IDE is useful but is not as good at code completion and error location as Android's IDE. Errors do not get underlined until compile time and do not show if they have been resolved until a new compile which can be misleading. The code completion was slow, but faster than MoSync.

Developing with WebWorks was very similar to PhoneGap. For many tests, the identical code was compiled with both tools but WebWorks performed better. The same issues are apparent with the lack of an adequate IDE and emulator. The APIs are limited but allow for additional customization through Java and community extensions. Overall application performance was fairly good for this native-like tool.

For each of the native tools, they receive updates as soon as they are released, since they come from the vendor themselves. This can help provide the latest features to users and also allow for stability when new features and enhancements are added. The costs of development very with most offering a free tier with the only exception being iOS application publication.

5.4 Analysis

The results of the evaluation of these 4 CPDTs and corresponding native tools provided an assortment of winners and losers. In some instances, the cross-platform

tools performed much better in the controlled experiments than the native, but in others much worse. PhoneGap performed the worst on average and iOS Native was the best on its platform.

Adobe Air and Appcelerator Titanium were shown to be the most feature rich and best performing tools in testing, however, both still have some downsides discussed previously. Adobe tools worked well but required more platform specific code to complete certain action than the others. Titanium also required modification as the UI must be independently developed for both Android and iOS. This however, was seen to provide better performance in comparison to PhoneGap which provided the most portable code of the group.

A clear winner is not apparent; however the tests show that depending on the development required, different tools may be best used. UI intensive tasks would favour Titanium or Air where simple applications would be best on PhoneGap to reach the widest audience. MoSync was seen as a disappointment in features, performance and development experience although this may change as the tool matures. It will be vitally important for any developer considering using a CPDT to look to evaluations such as this to find the matching features for their application before beginning development.

The results show that CPDTs can rival their native counterparts in many respects and should be strongly considered for development. The feature sets are becoming more robust and time is saved by building the applications using the tools rather than multiple native versions. The results show that native tools do not necessarily mean the application will have the best performance but you must instead match your tool to your application.

The framework itself lent quite well to providing a full view of the CPDT capabilities and performance. The high level feature structure was able to be broken into its various components and applied to evaluate these CPDTs providing the best

overview available. The performance benchmarks have shown the first testing of apples to apples comparison of algorithm completion times using code compiled with multiple CPDTs. This information has provided the necessary information for developers to choose the correct CPDT for their application.

The results of this evaluation have also shown that the current set of CPDTs often show weaknesses when compared to their native counterparts. To eliminate the issue, optimization techniques must be developed to enable cross-platform developers to be on a more level playing field to those using native tools.

5.5 Summary

In this chapter we saw the implementation of the evaluation framework applied to PhoneGap, Appcelerator Titanium, Adobe Air, MoSync and a collection of native tools. The tests were implemented for a variety of areas of study in the framework to show the viability of each of the pieces. Each tool fared very differently in the feature comparison. The number of features supported varied greatly and showed that it is important to complete a feature audit before choosing a tool for development.

The results of this evaluation showed that the frameworks, even with nearly identical code, perform very differently in performance testing. These differences are sometimes very significant and may deter developers from using a tool such as MoSync or PhoneGap that performed poorly in comparison. The native tools performed quite well, especially on the iOS platform where it provided the best performance in the majority of tests.

Aspects of the various tools showed where some have difficulties and others do not. The development environment was seen to be lacking compared to most of the native tools. We saw that the CPDTs did not score as highly on the evaluation as some native tools. In the next chapter we will draw conclusions and discuss methods for extending this research.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Developing applications for today's diverse mobile platform market is a time consuming and difficult task. We have seen that developers have an open question about whether CPDTs can be used to save development time, while keeping a certain level of performance and functionality. Through developing a framework for evaluating CPDTs in this thesis, we have produced a method of answering this and applied it to numerous tools. This has shown that the framework provides the details needed to answer this question through its benchmarking and evaluation.

Our testing has shown that the clear differences in the tools chosen can have a great impact both negatively and positively on development of a mobile application. Some CPDTs were shown to have performance issues while others provided too little functionality. The most attractive part of this framework is that it can be extended to new functionality brought by future releases of CPDTs with the core concepts remaining. We have seen that it is important for developers to choose the right tool for their particular application and this evaluation framework provides enough detail on the CPDT tested, to allow developers to choose which tool is fit to purpose for them.

The contributions of this thesis are:

- A high level, extensible framework for evaluation of any CPDT,
- An implementation of the framework with specific test criteria that can be used for in-market CPDTs,
- Specific testing procedures for mobile device benchmarks,

- A wide ranging evaluation of several popular cross-platform tools using standard benchmarks and new specifically designed tests,
- Creation of reusable tests and scripts that can be translated into an API.

Through our own development and evaluation, we have seen that the usage of CPDTs still has many downsides. Performance issues and a somewhat steep learning curve remain while documentation is being improved. UI considerations must be made for each platform and the evaluation has shown that write once, run anywhere is currently not possible.

As mobile OS platform makers make their browsers more standards compliant, web-based tools like PhoneGap will improve with HTML5 integration. Each of these tools is still in their infancy and more APIs will be added and limitations removed as time goes on. The development process is very fluid for mobile applications and it seems that will continue to be true over the next few years. Using this evaluation framework and consistently updating the results will help developers sort through the difficult task of choosing a CPDT. With the number of platforms available, developers must be very versatile and businesses need to expend significant resources to have their applications available on more than one device so interest in the usage of these tools is only set to grow.

6.2 Future Work

This thesis has been successful in providing a robust framework to evaluate cross-platform tools from many angles, but there is room for evolution. Testing and the number of CPDTs included were limited by the immense time and effort it takes to complete one of these evaluations. Further testing can occur to provide guidance with new CPDTs that was not currently necessary to prove the validity of this framework. The framework can be extended to test other uses such as game development that were out of scope.

The scripts used for testing can in the future be turned into advanced APIs when additional tests are implemented. Additionally, the same tests and concepts can be extended to be tested with tablet devices. The larger screen and higher resolution of tablets will allow for expanded UI testing. The same tests may incorporate other aspects including memory usage and application size as well as further statistical analysis.

It is expected that the APIs used for remote database testing and logging can be expanded for use by outside parties to complete their tests and report their results into a larger system. These results can be verified and displayed in some usable form for a developer to easily review evaluations for multiple CPDTs at a glance to choose the features they require and fine the correct tool for their purpose.

Bibliography

[1] Adobe Systems Inc. (2012) PhoneGap. [Online]. http://www.phonegap.com/.

[2] Adobe Systems Inc. (2012, April) PhoneGap. [Online]. https://build.phonegap.com/docs/config-xml.

[3] V. Agarwal, S. Goyal, S. Mittal, and S. Mukherjea, "MobiVine: A Middleware Layer to Gandle Fragmentation of Platform Interfaces for Mobile Applications," in *Middleware '09: Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, 2009, pp. 24:1-24:10.

[4] S. Allen, V. Graupera, and L. Lundrigan, *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile and Android Development and Distribution*, 1st ed. Berkely, CA, USA: Apress, 2010.

[5] AnTuTu Hong Kong. (2012) AnTuTu. [Online]. http://www.antutu.com.

[6] Appcelerator Inc. / IDC, "Q1 2011 Mobile Developer Report," Tech. Rep. 2011. [Online]. http://assets.appcelerator.com.s3.amazonaws.com/docs/Appcelerator-IDC-Q1-2011-Mobile-Developer-Report.pdf.

[7] Appcelerator Inc. / IDC, "Q3 2012 Mobile Developers Report," Tech. Rep. 2012. [Online]. http://www.appcelerator.com.s3.amazonaws.com/pdf/Appcelerator-Report-Q3-2012-final.pdf.

[8] Appcelerator Inc. (2012) Appcelerator. [Online]. http://www.appcelerator.com/.

[9] Appcelerator Inc., "Appcelerator Announces Beta Support For Blackberry," Press Release 2010. [Online]. http://www.appcelerator.com/2010/04/appcelerator-announces-beta-support-for-blackberry/.

[10] Appcelerator Inc. (2012) Appcelerator Titanium Mobile (SDK). [Online]. http://docs.appcelerator.com/titanium/2.0/#!/api.

[11] appMobi. (2012, February) HTML5 App School Webinar: Track your app's usage using appMobi's statMobi analytics. Webinar. [Online]. http://www.youtube.com/watch?v=lU7NJY6RYQE.

[12] appMobi. (2012) The appMobi PhoneGap XDK turbocharges PhoneGap. [Online]. http://dev.appmobi.com/?q=node/153.

[13] Aurora Softworks. (2012, May) Aurora Softworks Home. [Online]. http://www.aurorasoftworks.com/.

[14] J. Babb et al., "The RAW benchmark suite: computation structures for general purpose computing," in *IEEE Symposium on Field-Programmable Custom Computing Machines*, apr 1997, pp. 134-143.

[15] H. Behrens, "Cross-Platform App Development for iPhone, Android and Co. — A Comparison," in *MobileTechCon*, Mainz, September 2010.

[16] M.D. Bloice, F. Wotawa, and A. Holzinger, "Java's alternatives and the limitations of Java when writing cross-platform applications for mobile devices in the medical domain," in *ITI '09. Proceedings of the ITI 2009 31st International Conference on Information Technology Interfaces, 2009.*, june 2009, pp. 47-54.

[17] J. M. Bull, L. A. Smith, L. Pottage, and R. Freeman, "Benchmarking Java against C and Fortran for scientific applications," in *Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande*, 2001, pp. 97-105.

[18] C. Cantrell. (2011, March) iOS Features in Adobe Air. [Online]. http://www.adobe.com/devnet/air/articles/ios_features_in_air26.html.

[19] Centre for Mobile Education and Research. (2012, May) CMER Applications. [Online]. http://cmer.uoguelph.ca/apps.html.

[20] A. Charland and B. LeRoux, "Mobile Application Development: Web vs. Native," *Queue*, vol. 9, pp. 20:20--20:28, April 2011.

[21] S. Che et al., "Rodinia: A benchmark suite for heterogeneous computing," in *Proceedings of the 2009 IEEE International Symposium on Workload Characterization (IISWC)*, Washington, oct. 2009, pp. 44-54.

[22] S. Che et al., "A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads," in *IISWC '10 Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'10)*, Washington, dec. 2010, pp. 1-11.

[23] comScore. (2012, October) comScore Reports August 2012 U.S. Mobile Subscriber Market Share. Press Release. [Online]. http://www.comscore.com/Press_Events/Press_Releases/2012/10/comScore_Reports_August_2012_U.S._Mobile_Subscriber_Market_Share.

[24] B. Cornelius, *Understanding Java*. Harlow, England: Addison-Wesley, 2001.

[25] C. Cowell-Shah. (2004, January) Nine Language Performance Round-up: Benchmarking Math & File I/O. [Online]. http://www.osnews.com/story/5602.

[26] U. Dave and R. Samant, "Overview of smartphone application development using cross-platform framework," in *ICWET '11: Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, 2011, pp. 1379-1379.

[27] P. Dickson, "Cabana: a cross-platform mobile development system," in *SIGCSE '12: Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 2012, pp. 529-534.

[28] P. Fleming and J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results," *Communications of the ACM*, vol. 29, no. 3, pp. 218-221, March 1986.

[29] Form Tools. (2012) Generate Data. [Online]. http://www.generatedata.com/.

[30] T. Fuchs. (2012, April) Zepto.js. [Online]. http://zeptojs.com/.

[31] M. Gualtieri. (2012, June) Mike Gualtieri's Blog. [Online]. http://blogs.forrester.com/mike_gualtieri/12-06-02-hey_developers_make_your_mobile_apps_blazing_fast.

[32] G. Hartmann, G. Stead, and A. DeGani, "Cross-platform mobile development," Tribal, Lincoln House, The Paddocks, Tech. Rep. March 2011. [Online]. http://www.mole-

project.net/images/documents/deliverables/WP4_crossplatform_mobile_development_March2011.pdf.

[33] K. Hoste et al., "Performance prediction based on inherent program similarity," in *PACT '06: Proceedings of the 15th international conference on Parallel architectures and compilation techniques*, 2006, pp. 114-122.

[34] G. Hu and A. Gadapa, "Compiling C++ programs to Java bytecode," in *SNPD-SAWN '05: Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05)*, #May# 2005, pp. 56-61.

[35] IBM. (2012) IBM - Client Architecture - PhoneGap. [Online]. http://www-01.ibm.com/software/mobile-solutions/worklight/features/phonegap/.

[36] IBM. (2012) IBM Worklight. [Online]. http://www-01.ibm.com/software/mobile-solutions/worklight/.

[37] IBM, "Worklight and PhoneGap Comparison," Worklight Inc., Tech Brief 2012. [Online]. http://www.worklight.com/assets/pdf/Worklight%20vs%20PhoneGap%20-%20Comparison.pdf.

[38] J. Issa et al., "TMAPP – Typical Mobile Applications Benchmark," in *MoBS7 '11: Seventh Annual Workshop on Modeling, Benchmarking and Simulation*, 2011.

[39] S. Kaltofen, M. Milrad, and A. Kurti, "A cross-platform software system to create and deploy mobile mashups," in *ICWE'10: Proceedings of the 10th international conference on Web engineering*, Berlin, Heidelberg, 2010, pp. 518-521.

[40] Y. Kao, C. Lin, K. Yang, and S. Yuan, "A Cross-Platform Runtime Environment for Mobile Widget-Based Application," in *CYBERC '11: Proceedings of the 2011 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, oct. 2011, pp. 68-71.

[41] U. Krishnaswamy and I. Scherson, "A framework for computer performance evaluation using benchmark sets," *IEEE Transactions on Computers*, vol. 49, no. 12, pp. 1325-1338, dec 2000.

[42] J. Lyle, S. Monteleone, S. Faily, D. Patti, and F. Ricciato, "Cross-Platform Access Control for Mobile Web Applications," in *POLICY '12: Proceedings of the 2012 IEEE International Symposium on Policies for Distributed Systems and Networks*, july 2012, pp. 37-44.

[43] Macadamian, "Choosing A Cross-Platform Mobile Framework," Tech. rep. 2011. [Online]. http://www.macadamian.com/images/uploads/whitepapers/Macadamian_Mobile_Cross_Platform.pdf.

[44] MoSync. (2010, September) MoSync. [Online]. http://www.mosync.com/documentation/manualpages/optimizing-mobile-applications.

[45] Motorola Solutions Inc. (2012) Rhodes. [Online]. http://www.motorola.com/Business/US-EN/RhoMobile+Suite/Rhodes.

[46] G. Nikishkov, Y. Nikishkov, and V. Savchenko, "Comparison of C and Java performance in finite element computations," *Computers & Structures*, vol. 81, no. 24–25, pp. 2401-2408, 2003.

[47] J. Ohrt and V. Turau, "Cross-Platform Development Tools for Smartphone Applications," *Computer*, vol. 45, no. 9, pp. 72-79, September 2012.

[48] D. Olanoff. (2012, September) Tech Crunch. [Online]. http://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5/.

[49] T. Paananen, "Smartphone Cross-Platform Frameworks," Jamk University of Applied Sciences, Thesis 2011. [Online]. https://publications.theseus.fi/bitstream/handle/10024/30221/110510_Thesis_Timo_Paananen.pdf.

[50] L. Paulson, "Materials Breakthrough Could Eliminate Bootups," *Computer*,

vol. 42, no. 6, pp. 23-25, June 2009.

[51] L. Prechelt, "An Empirical Comparison of Seven Programming Languages," *Computer*, vol. 33, no. 10, pp. 23-29, October 2000.

[52] A. Puder, "Cross-compiling Android applications to the iPhone," in *PPPJ '10: Proceedings of the 8th International Conference on the Principles and Practice of Programming in Java*, 2010, pp. 69-77.

[53] A. Puder and I. Yoon, "Smartphone Cross-Compilation Framework for Multiplayer Online Games," in *ELML '10: Proceedings of the 2010 Second International Conference on Mobile, Hybrid, and On-Line Learning*, #feb.# 2010, pp. 87-92.

[54] D. Rajapakse, "Techniques for Defragmenting Mobile Applications: A Taxonomy," in *SEKE, 2008*, 2008, pp. 923-928.

[55] J. Reed and P. McMahan. (2000, June) Linpack Benchmark -- Java Version. [Online]. http://www.netlib.org/benchmark/linpackjava/.

[56] Research in Motion, "BlackBerry WebWorks SDK: Development Guide," Tech. rep. ISBN: SWD-1214876-1129021640-001, 2010. [Online]. http://docs.blackberry.com/en/developers/deliverables/20772/BlackBerry_Widget_SDK-Development_Guide--1214876-0920094453-001-1.5-US.pdf.

[57] Research in Motion Inc. (2011, October) RIM Unveils BlackBerry BBX - Combines the Best of BlackBerry and QNX to Provide a Next Generation Platform for BlackBerry Smartphones and Tablets. Press Release. [Online]. http://press.rim.com/release.jsp?id=5230.

[58] V. Rijmen and J. Daemen. (2001, February) AES Algorithm (Rijndael) Information. [Online]. http://csrc.nist.gov/archive/aes/rijndael/wsdindex.html.

[59] R. Rogers, "Developing Portable Mobile Web Applications," *Linux J.*, vol. 2010, September 2010.

[60] Sencha Inc. (2012) Sencha Touch. [Online]. http://www.sencha.com/products/touch.

[61] F. Sibai, "Evaluating the Performance of Single and Multiple Core Processors

with PCMARK 05 and Benchmark Analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 35, pp. 62-71, March 2008.

[62] B. Siegfried, "Enhanced student technology support with cross-platform mobile apps," in *SIGUCCS '11: Proceeding of the 39th ACM annual conference on SIGUCCS*, 2011, pp. 31-34.

[63] I. Singh and M. Palmieri, "Comparison of Cross-Platform Mobile Development Tools," in *2012 16th International Conference on Intelligence in Next Generation Networks*, Berlin, 2012.

[64] P. Smutny, "Mobile development tools and cross-platform solutions," in *2012 13th International Carpathian Control Conference (ICCC)*, may 2012, pp. 653-656.

[65] The jQuery Foundation. (2012) jQuery Mobile. [Online]. http://jquerymobile.com/.

[66] N. Uti and R. Fox, "Testing the Computational Capabilities of Mobile Device Processors: Some Interesting Benchmark Results," in *2010 IEEE/ACIS 9th International Conference on Computer and Information Science (ICIS)*, Washington, DC, USA, 2010, pp. 477-481.

[67] K. Vaananen-Vainio-Mattila and M. Waljas, "Developing an Expert Evaluation Method for User Experience of Cross-Platform Web Services," in *MindTrek '09 Proceedings of the 13th International MindTrek Conference: Everyday Life in the Ubiquitous Era*, 2009, pp. 162-169.

[68] Vision Mobile, "Cross-Platform Developer Tools 2012," London, Tech. rep. 2012. [Online]. http://www.visionmobile.com/product/cross-platform-developer-tools-2012/.

[69] W3C. (2010, December) Mobile Web Application Best Practices. [Online]. http://www.w3.org/TR/mwabp/.

[70] A. Wasserman, "Software Engineering Issues for Mobile Application Development," in *FoSER '10: Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 397-400.

[71] WebKit Open Source Project. (2010, April) SunSpider JavaScript Benchmark. [Online]. http://www.webkit.org/perf/sunspider/sunspider.html.

[72] D. Winokur. (2011, November) Flash to Focus on PC Browsing and Mobile Apps; Adobe to More Aggressively Contribute to HTML5. [Online]. http://blogs.adobe.com/conversations/2011/11/flash-focus.html.

[73] C. Xin, "Cross-Platform Mobile Phone Game Development Environment," in *IIS '09: Proceedings of the 2009 International Conference on Industrial and Information Systems*, april 2009, pp. 182-184.

Appendix A: CPDT Features to be Evaluated

Table A.1 shows the features that are included in this evaluation of the CPDTs chosen for study. These specific features each fall under the categories mentioned in the framework discussed in Chapter 3. The features marked with an asterisk are able to be categorized as supported, not supported and partially supported.

Table A.1: CPDT Features in Evaluation

CPDT Basic Elements	Development Environment	User Experience	Device Access	Sensors	Geolocation	Notifications	Monetization	Security
Version Studied	IDE type	Access Native User Interface Elements*	File System*	Camera (Video, still, front and rear) *	Wi-Fi Positioning*	User Notifications*	Application Store Support*	Access to Secure Storage*
Platforms Supported	Development Language	Screen Rotation*	SMS*	Microphone*	Cellular Positioning*	System Notifications*	In App Purchases (Native or 3rd party) *	Code Obfuscation*
Ability to use Background processes*	Compilation Type	Swipe*	Call log*	Noise Cancellation Microphone*	GPS*	Push Notifications* (Native or 3rd Party)	Mobile Ad Platform support (Native or 3rd Party) *	
Costs for development (Free tier Availability)*	Debug Environment	Pinch*	Contacts*	Sensor Data Capture*	Native Map Support*		Analytics Platform Provided/ Compatibility*	
License	MVC Support*	Accessibility Features*	Calendar*	Proximity*				
	Social APIs*	Ability to playback media*	Low level Networking*	NFC*				
	Cloud APIs*		Ability to Choose Data Path*	Accelerometer*				
	Build Service Availability*		GPU Acceleration*	Gyroscope*				
			Bluetooth*	Barometer*				
			Voice Activation*	Compass*				

Appendix B: Benchmark Application Skeleton

Below is the code for the benchmark application skeleton for BlackBerry WebWorks. The code will include the files index.html, Main.js and Skeleton.js. Index.html is the initial file that is loaded for the benchmark application. It handles the UI elements and the starting of the tests. Main.js handles the progress and display of the final results while Skeleton.js handles the running and logging of test data.

index.html

```
<html>

    <head>
        <link rel="stylesheet" type="text/css" href="CPDT.css" />
        <!-- Add each test and script that is used within skeleton !-->
        <script type="text/javascript" src="./phonegap.js">
        </script>
        <script type="text/javascript" src="./scripts/zepto.min.js">
        </script>
        <script type="text/javascript" src="./scripts/UIElements.js">
        </script>
        <script type="text/javascript" src="./scripts/CryptoAES.js">
        </script>
        <script type="text/javascript" src="./scripts/RemoteAccess.js">
        </script>
        <script type="text/javascript" src="./scripts/Sorting.js">
        </script>
        <script type="text/javascript" src="./scripts/test.js">
        </script>
        <script type="text/javascript" src="./scripts/test2.js">
        </script>
        <script type="text/javascript"
        src="./scripts/InputValidation.js">
        </script>
        <script type="text/javascript"
        src="./scripts/LocalPIMAccess.js">
        </script>
        <script type="text/javascript" src="./scripts/Skeleton.js">
        </script>
        <script type="text/javascript" src="./scripts/Main.js">
        </script>
    </head>

    <!-- Inside the body of the document, create the table listing all
         tests and the run button !-->
    <body onload="drawTable();">
```

```
        <div id="main">
            <div id="debug-div">
             <span>Debug Mode</span>
                <input type="checkbox" value=false id="debug-switch"
                />
            </div>
            <div id="table"></div>
            <div id="run-div">
                <input type="button" value="Run Selected Tests"
                onclick="runTests();"
                />
            </div>
            <div id="progress-div" style="display:none;">
            Running Tests...
                <div id="progress-bar">
                    <div id="progress"> </div>
                </div>
            </div>
        </div>
        <div id="UITest" style="text-align:center;"></div>
    </body>

</html>
```

Main.js

```
var skeleton = new Skeleton();

/* Create the table to display the output */
function drawTable() {

$("#table").html("<table><tr><th>Enabled</th><th>TestName</th><th>Avg.
Time</th></tr></table>");
    for (var i = 0; i < skeleton.tests.length; i++) {
        $("#table table").append("<tr id=\"row" + i + "\"></tr>");
         $("#row" + i).append("<td><input type=\"checkbox\"
       id=\"check"       + i + "\" checked=\"true\"/></td>");
         $("#row" + i).append("<td>" + skeleton.tests[i].getName()
        + "</td>");
         $("#row" + i).append("<td class=\"time\" id=\"time-" +
       skeleton.tests[i].getName() + "\">" + "" + "</td>");
    }
}

/* update the progress window */
function updateProgress(outOfOne) {
    $("#progress").css("width", "" + (outOfOne * 100) + "%");
}

/* when tests are finished, run this function and calculate the
averages and show to the user */
function testsFinished() {
    var averages = skeleton.getAverages();
    $(".time").html("");
    for (var test in averages) {
        $("#time-" + test).html("" + averages[test]);
```

```
    }
    $("#debug-div").css("display", "block");
    $("#table").css("display", "block");
    $("#run-div").css("display", "block");
    $("#progress-div").css("display", "none");
    skeleton.logTestRemote();
}

/*run selected tests within the skeleton object*/
function runTests() {
    $("#debug-div").css("display", "none");
    $("#table").css("display", "none");
    $("#run-div").css("display", "none");
    $("#progress-div").css("display", "block");
    var debug = ($("#debug-switch").is(":checked") ? true : false);
    var enabled = new Object;
    for (var i = 0; i < skeleton.tests.length; i++) {
        enabled[skeleton.tests[i].getName()] =
       ($("#check" + i).is(":checked") ? true : false);
    }
    skeleton.runTests(enabled, debug, testsFinished);

}
```

Skeleton.js

```
var DEBUG_ITERATIONS = 1;
var ITERATIONS = 5;

/* location of logging server */
var LOG_URL = "http://apps.socs.uoguelph.ca/CPDTEvaluation/index.php";

function Skeleton() {
    /*Member Variables*/
    this.tests = new Array();
    this.results = new Array();
    this.debug = false;
    this.testFinished = false;
    this.callbackStack = new callbackStack();

   /* Create tests */
    this.tests.push(new InputValidation());
    this.tests.push(new AESEncryption());
    this.tests.push(new LocalPIMAccess());
    this.tests.push(new Sorting());
    this.tests.push(new RemoteAccess());
    this.tests.push(new UIElements());


    /*Methods*/
    this.addResults = function (testName, startTime, endTime) {
        this.results.push({
            'testName': testName,
            'startTime': startTime,
            'endTime': endTime
        });
```

```
        this.callbackStack.callback();
    };

    this.markTestFinished = function () {
        this.testFinished = true;
    };
    /* run tests with desired paramaters */
    this.runTests = function (enabled, debug, callback) {
        var obj = this;
        this.callbackStack.stackEmpty = callback;
        this.debug = debug;
        this.results = new Array();
        var iterations = (this.debug ? DEBUG_ITERATIONS : ITERATIONS);
        var priority = 0;
        for (var j = 0; j < iterations; j++) {
            for (var i = 0; i < this.tests.length; i++) {
                if (enabled[this.tests[i].getName()] == true) {
                    this.testFinished = false;
                    var currentTest = eval("new " +
                    this.tests[i].getName() + "(" + priority + ");");
                    priority++;
                    this.callbackStack.put(
                    currentTest.test,
                    doNothing, [this],
                    currentTest, this, priority);

                }
            }
        }


        this.callbackStack.callFirst();

    };

    /* retrieve the averages from the tests */
    this.getAverages = function () {
        var averages = new Object();
        for (var i = 0; i < this.results.length; i++) {
            if (averages[this.results[i].testName] == undefined) {
                averages[this.results[i].testName] = 0;
            }
            averages[this.results[i].testName] +=
          (this.results[i].endTime - this.results[i].startTime);
        }
        for (var index in averages) {
            averages[index] = averages[index] /
          (this.debug ? DEBUG_ITERATIONS : ITERATIONS);
        }
        return averages;
    };

    /* create the log string in long form */
    this.generateLongLog = function () {
        var delim = ';';
        var logString = ""; /*"TestName,StartTime,EndTime,Duration" +
```

```
                                delim;*/
            for (var i = 0; i < this.results.length; i++) {
                logString += this.results[i].testName + ',' +
              this.results[i].startTime + ',' + this.results[i].endTime +
              ',' + (this.results[i].endTime - this.results[i].startTime)
              + delim;
            }
            return logString;
        };

        /* Create the short log string with averages only */
        this.generateShortLog = function () {
            var delim = ';';
            var logString = ""; //"TestName,AverageTime" + delim;
            var averages = this.getAverages();
            for (var index in averages) {
                logString += index + ',' + averages[index] + delim;
            }
            return logString;
        };

        /* Send information to logging server */
        this.logTestRemote = function () {
            var date = new Date();
            var data = "";
            data += device.platform + " " + device.name + ",";
            data += "WebWorks,";
            data += device.version + ",";
            data += date.getFullYear() + "-" +
            (date.getMonth() + 1) + "-" + date.getDate() + ",";
            data += date.getHours() + ":" +
            date.getMinutes() + ":" + date.getSeconds() + ";";
            data += this.generateShortLog();
            data += this.generateLongLog();
            $.post(
            LOG_URL, {
                'data': data
            },

            function (data, status, xhr) {
                //Do Nothing
            });
        };

    }

    function doNothing() {
        //DO NOTHING
    }

    /* Stack to hold commands for each function */
    function callbackStack() {

        this.stackEmpty = null;
        this.executeStack = new Array();
        this.parameterStack = new Array();
```

```
    this.callbackStack = new Array();
    this.functionContextStack = new Array();
    this.callbackContextStack = new Array();
    this.priorityStack = new Array();
    this.put = function (func, callback, param,
   functionContext, callbackContext, priority) {
        //console.log("\ncallbackStack.put\n");
        var i = 0;
        while (this.priorityStack[i] < priority) {
            i++;
        }


        this.executeStack.splice(i, 0, func);
        this.callbackStack.splice(i, 0, callback);
        this.parameterStack.splice(i, 0, param);
        this.functionContextStack.splice(i, 0, functionContext);
        this.callbackContextStack.splice(i, 0, callbackContext);
        this.priorityStack.splice(i, 0, priority);
    }
    this.callFirst = function () {

        var func = this.executeStack.pop();
        var param = this.parameterStack.pop();
        var context = this.functionContextStack.pop();
        func.apply(context, param);
    }
    this.callback = function (data) {
        var callbackContext = this.callbackContextStack.pop();

        //execute the spesific callback
        var callback = this.callbackStack.pop();
        callback.call(callbackContext, data);
        //console.log(callback);
        //If there are more commands on the stack, we are going to call
          them too
        if (this.executeStack.length <= 0) {
            this.stackEmpty.call(null, null);
            return;
        }
        var functionContext = this.functionContextStack.pop();
        var func = this.executeStack.pop();
        var param = this.parameterStack.pop();
        this.priorityStack.pop();
        try {
            setTimeout(function () {
                func.apply(functionContext, param)
            }, 0);
        } catch (e) {
            console.log(e);
        }
        //console.log(func);
    }
}

/* functions for reading the file system */
```

```
function logTest() {
    window.requestFileSystem(LocalFileSystem.PERSISTENT, 0, gotFS,
fail);
}

/* create the file on the server */
function gotFS(fileSystem) {
    var date = new Date().getUTCDate();
    alert(fileSystem.root.fullPath);
    fileSystem.root.getFile("Documents/CPDT-Log-Short-" + date, {
        create: true,
        exclusive: false
    }, gotFileEntry_Short, fail);
    fileSystem.root.getFile("Documents/CPDT-Log-Long-" + date, {
        create: true,
        exclusive: false
    }, gotFileEntry_Long, fail);
}
/*
 * The functions below are for generating and
 * writing the log to the server
 */
function gotFileEntry_Short(fileEntry) {
    alert(fileEntry.fullPath);
    fileEntry.createWriter(gotFileWriter_Short, fail);
}

function gotFileEntry_Long(fileEntry) {
    alert(fileEntry.fullPath);
    fileEntry.createWriter(gotFileWriter_Long, fail);
}

function gotFileWriter_Short(writer) {
    alert("Writing Log");
    writer.write(skeleton.generateShortLog());
}

function gotFileWriter_Long(writer) {
    alert("Writing Log");
    writer.write(skeleton.generateLongLog());
}

function fail(error) {
    alert(error.code);
}
```

Appendix C: Benchmark Application UI Elements Test

A sample of the UI Elements test using the WebWorks tool for BlackBerry 7 devices is shown here. This JavaScript file completes the test and returns the length of time taken to the application skeleton.

```
/* UI Elements Test Module */
function UIElements(priority) {
    this.priority = priority;
    this.name = "UIElements";
    this.getName = function () {
        return this.name;
    }
    this.skeleton = null;
    this.startTime;
   /*  Text used for data field */
this.testText = "ROMEO: But, soft! what light through yonder window
breaks? \n\
                It is the east, and Juliet is the sun. \n\
                Arise, fair sun, and kill the envious moon, \n\
                Who is already sick and pale with grief, \n\
                That thou her maid art far more fair than she: \n\
                Be not her maid, since she is envious; \n\
                Her vestal livery is but sick and green \n\
                And none but fools do wear it; cast it off. \n\
                It is my lady, O, it is my love! \n\
                O, that she knew she were! \n\
                She speaks yet she says nothing: what of that? \n\
                Her eye discourses; I will answer it. \n\
                I am too bold, 'tis not to me she speaks: \n\
                Two of the fairest stars in all the heaven, \n\
                Having some business, do entreat her eyes \n\
                To twinkle in their spheres till they return. \n\
                What if her eyes were there, they in her head? \n\
                The brightness of her cheek would shame those stars,
                \n\
                As daylight doth a lamp; her eyes in heaven \n\
                Would through the airy region stream so bright \n\
                That birds would sing and think it were not night. \n\
                See, how she leans her cheek upon her hand! \n\
                O, that I were a glove upon that hand, \n\
                That I might touch that cheek! \n\
                JULIET: Ay me! \n\
                ROMEO: She speaks: \n\
                O, speak again, bright angel! for thou art \n\
                As glorious to this night, being o'er my head \n\
                As is a winged messenger of heaven \n\
                Unto the white-upturned wondering eyes \n\
                Of mortals that fall back to gaze on him \n\
                When he bestrides the lazy-pacing clouds \n\
```

```
                And sails upon the bosom of the air.\n";

    /* The checkboxes and dropdown box for the first partition are
created in this statement  */
     this.partition1 = "<div id=\"elements-header\"
    onclick=\"accordianSwap(1);\" style=\"border-style:solid;\">Form
    Elements</div>\
       <div id=\"elements\">\
          <table>\
          \
             <tr>\
                <td>\
                <span display=\"block\" id=\"check-group-1\">\
                <input type=\"checkbox\"/></span><br/>\
                <span display=\"block\" id=\"check-group-2\">\
                <input type=\"checkbox\"/></span><br/>\
                <span display=\"block\" id=\"check-group-3\">\
                <input type=\"checkbox\"/></span><br/>\
                <span display=\"block\" id=\"check-group-4\">\
                <input type=\"checkbox\"/></span><br/>\
                <span display=\"block\" id=\"check-group-5\">\
                <input type=\"checkbox\"/></span><br/>\
                </td>\
                <td>\
                   <input type=\"text\" id=\"text-field\">\
                <td/>\
             </tr>\
             <tr>\
                <td>\
                   <!--<select>\
                   <!-- Code truncated to remove
                   select for all 50 US states !-->\
                   </select>!-->\
                </td>\
                <td>\
                   <span class=\"switch\" id=\"switch\">\
                   <input type=\"button\" class=\"switch-option \
                   option-1\" value=\"ON\" onclick=\"toggleSwitch();\
                   \" disabled=\"disabled\"/><input type=\"button\" \
                   class=\"switch-option option-2\" value=\"OFF\" \
                   onclick=\"toggleSwitch();\"/>\
                   </span>\
                </td>\
             </tr>\
             <tr id=\"button-cell\">\
                <td><input type=\"button\" value=\"Button 1\" \
                id=\"button-1\"/></td>\
                <td><input type=\"button\" value=\"Button 2\" \
                id=\"button-2\"/></td>\
             </tr>\
          </table>\
       </div>";
    /* Partition 2 contains images that are displayed at various sizes*/
     this.partition2 = "<div id=\"images-header\"
    onclick=\"accordianSwap(2);\" \
    style=\"border-style:solid;\">Images</div>\
```

```
   <div id=\"images\">\
      <img src=\"image1.png\" alt=\"Image 1\"/><br/>\
<img src=\"image2.png\" alt=\"Image 2\" style=\"width:20%; \
height:20%\"/>\
   </div>";

/* The third partition includes a text text area
containing a verse from Romeo and Juliette  */
 this.partition3 = "<div id=\"text-header\" onclick=\
"accordianSwap(3);\" style=\"border-style:solid;\">Text</div>\
<div id=\"text\"><textarea rows=\"5\" style=\
"width:100%;\" id=\"textarea\">"
+ this.testText + "\
</textarea></div>";

/* The function below starts the test process  */
 this.test = function (skeleton) {
     var obj = this;

     this.startTime = new Date();
     this.skeleton = skeleton;
     $("#UITest").html(this.partition1 + this.partition2
   + this.partition3);
     accordianSwap(1);

   /* Set call back stack for all elements of the test */
     this.skeleton.callbackStack.put(function () {
         $("#main").hide();
         obj.skeleton.callbackStack.callback();
     }, doNothing, [], null, null, this.priority);


     this.skeleton.callbackStack.put(function () {
         $("#UITest").show();
         obj.skeleton.callbackStack.callback();
     }, doNothing, [], null, null, this.priority);

     this.skeleton.callbackStack.put(function () {
         accordianSwap(2);
         obj.skeleton.callbackStack.callback();
     }, doNothing, [], null, null, this.priority);

     this.skeleton.callbackStack.put(function () {
         accordianS
         wap(3);
         obj.skeleton.callbackStack.callback();
     }, doNothing, [], null, null, this.priority);

     for (var i = 0; i < 20; i++) {

         this.skeleton.callbackStack.put(function () {
             $("#textarea").html("");
             obj.skeleton.callbackStack.callback();
         }, doNothing, [], null, null, this.priority);
```

```
    this.skeleton.callbackStack.put(function () {
        $("#textarea").html(obj.testText);
        obj.skeleton.callbackStack.callback();
    }, doNothing, [], null, null, this.priority);

}

this.skeleton.callbackStack.put(function () {
    accordianSwap(1);
    obj.skeleton.callbackStack.callback();
}, doNothing, [], null, null, this.priority);

for (var i = 0; i < 500; i++) {
    this.skeleton.callbackStack.put(

    function () {
        var index = Math.floor((Math.random() * 5) + 1);
        if ($("#check-group-" + index +
     " input").attr("checked") == 'checked')
     $("#check-group-" +
     index + " input").removeAttr("checked");
        else $("#check-group-" + index +
     " input").attr("checked", "checked");
        obj.skeleton.callbackStack.callback();
    }, doNothing, [], null, null, this.priority);


}
for (var i = 0; i < 50; i++) {
    this.skeleton.callbackStack.put(

    function () {
         var x = Math.floor((Math.random() *
         $(document).width()) + 1);
         var y = Math.floor((Math.random() *
         $(document).height()) + 1);
          $("#button-2").remove();
          $("#button-cell").append("<input type=\"button\"
         value=\"Button 2\" id=\"button-2\"
         style=\"position:absolute; top: "
         + y + "; left: " + x + ";\"/>");
        obj.skeleton.callbackStack.callback();
    }, doNothing, [], null, null, this.priority);

}

this.skeleton.callbackStack.put(function () {
    accordianSwap(2);
    obj.skeleton.callbackStack.callback();
}, doNothing, [], null, null, this.priority);
for (var i = 0; i < 20; i++) {
    this.skeleton.callbackStack.put(

    function () {
        $("#images").html("");
        if (whichIs20 == 2) {
```

```
                    $("#images").html("<img src=\"image1.png\"
                alt=\"Image 1\" style=\"width:20%; height:20%\"/><br/>
                <img src=\"image2.png\" alt=\"Image 2\"/>");
                    whichIs20 = 1;
                } else if (whichIs20 == 1) {
                    $("#images").html("<img src=\"image1.png\"
                    alt=\"Image 1\"/><br/><img src=\"image2.png\"
                    alt=\"Image 2\"
                    style=\"width:20%; height:20%\"/>");
                    whichIs20 = 2
                }
                obj.skeleton.callbackStack.callback();
            }, doNothing, [], null, null, this.priority);
        }


        this.skeleton.callbackStack.put(function () {
            $("#main").show();
            obj.skeleton.callbackStack.callback();
        }, doNothing, [], null, null, this.priority);
        this.skeleton.callbackStack.put(function () {
            $("#UITest").hide();
            obj.skeleton.callbackStack.callback();
        }, doNothing, [], null, null, this.priority);

        this.skeleton.callbackStack.put(

        function () {
            var endTime = new Date();
            this.skeleton.callbackStack.put(this.skeleton.addResults,
          doNothing, [obj.name, obj.startTime, endTime],
          obj.skeleton, obj.skeleton, obj.priority);
            obj.skeleton.callbackStack.callback();
        }, doNothing, [], null, null, this.priority);
        this.skeleton.callbackStack.callback();

    }

}

var whichIs20 = 2;
/*  Swaps the visible section in the accordian pane */
function accordianSwap(clicked) {
    if (clicked == 1) {
        $("#elements").show();
        $("#images").hide();
        $("#text").hide();
    } else if (clicked == 2) {
        $("#elements").hide();
        $("#images").show();
        $("#text").hide();
    } else if (clicked == 3) {
        $("#elements").hide();
        $("#images").hide();
        $("#text").show();
    }
```

```
}
/* changes the value of the option boxes  */
function toggleSwitch() {
    if ($("#switch .option-1").attr("disabled") == "disabled") {
        $("#switch .option-1").removeAttr("disabled");
        $("#switch .option-2").attr("disabled", "disabled");
    } else {
        $("#switch .option-1").attr("disabled", "disabled");
        $("#switch .option-2").removeAttr("disabled");
    }
}
```

Appendix D: AES Encryption Data

For the AES encryption, decryption test a password and text to be encrypted must be used. This data was first used in the SunSpider test that we intend to emulate [71]. The passage and password are passages from Shakespeare's Romeo and Juliet. They provide the length and complexity to allow for a measurable test.

Password: O Romeo, Romeo! wherefore art thou Romeo?

Text for encryption:

ROMEO: But, soft! what light through yonder window breaks?
It is the east, and Juliet is the sun.
Arise, fair sun, and kill the envious moon,
Who is already sick and pale with grief,
That thou her maid art far more fair than she:
Be not her maid, since she is envious;
Her vestal livery is but sick and green
And none but fools do wear it; cast it off.
It is my lady, O, it is my love!
O, that she knew she were!
She speaks yet she says nothing: what of that?
Her eye discourses; I will answer it.
I am too bold, 'tis not to me she speaks:
Two of the fairest stars in all the heaven,
Having some business, do entreat her eyes
To twinkle in their spheres till they return.
What if her eyes were there, they in her head?
The brightness of her cheek would shame those stars,
As daylight doth a lamp; her eyes in heaven
Would through the airy region stream so bright
That birds would sing and think it were not night.
See, how she leans her cheek upon her hand!
O, that I were a glove upon that hand,
That I might touch that cheek!
JULIET: Ay me!
ROMEO: She speaks:
O, speak again, bright angel! for thou art
As glorious to this night, being o'er my head
As is a winged messenger of heaven
Unto the white-upturned wondering eyes
Of mortals that fall back to gaze on him
When he bestrides the lazy-pacing clouds
And sails upon the bosom of the air.

Appendix E: Contact Data

The information contained below is from the file contactData.csv that is included in the suite of resources used to create the benchmark. This data generated randomly using the tools from [29], is used in the PIM testing to enter data into the device address book and remove it.

contactData.csv

First Name,Last Name,Phone,Email
Maxwell,Norman,1-610-680-6136,lectus.convallis.est@eget.ca
Laith,Hull,1-994-274-6251,nibh.Phasellus@NuncmaurisMorbi.com
Zachary,Hall,1-283-707-3724,purus.gravida.sagittis@mauriseuelit.edu
Xantha,Morin,1-371-996-0558,lacus.Ut.nec@doloregestasrhoncus.edu
Pascale,Mercado,1-983-356-6376,Cras.interdum@idante.edu
Griffin,Ford,1-239-295-0759,Cum@iaculisquispede.org
John,Ball,1-806-534-1938,facilisis@Nullam.org
Kessie,Goodman,1-948-123-9206,turpis.nec.mauris@vulputate.com
Chadwick,Mckay,1-850-144-8500,Phasellus.fermentum.convallis@sitamet.org
Aurora,Carver,1-735-655-2404,Donec.felis@tinciduntpede.com
Cally,Blevins,1-365-215-2138,tincidunt.vehicula@etipsumcursus.org
Ella,Boone,1-486-737-1817,non.luctus.sit@CurabiturmassaVestibulum.ca
Jillian,Hoffman,1-969-373-1131,ipsum.non.arcu@egestasSed.org
Duncan,Horn,1-261-343-0279,Quisque@eutellus.org
Amethyst,Meadows,1-789-892-9733,commodo.tincidunt@posuereenimnisl.edu
Florence,Carpenter,1-474-502-3277,ipsum.primis.in@duilectusrutrum.ca
Lynn,Hinton,1-867-830-2899,eu.erat.semper@Cras.com
Alma,Jimenez,1-916-334-0268,amet.nulla.Donec@gravidasagittis.com
Alana,Spears,1-439-628-5197,ligula@metusurnaconvallis.com
Uma,Floyd,1-527-553-3297,Suspendisse.eleifend@risusDonecegestas.org
Breanna,Daniels,1-431-840-6644,velit@nibhvulputate.edu
Melvin,Delacruz,1-715-145-7895,ante@acrisusMorbi.edu
Alfreda,Jennings,1-855-519-9363,facilisis.Suspendisse.commodo@eu.org
Colt,Odonnell,1-661-550-0676,dolor.Nulla.semper@tempusscelerisquelorem.org
Cora,Acevedo,1-239-266-4555,egestas.a.dui@volutpatNulladignissim.ca
Hayfa,Marshall,1-396-354-9516,a@purussapiengravida.ca
Cullen,Wise,1-196-541-0128,sit.amet@nibh.edu
Bell,Rush,1-562-414-0229,Phasellus@semper.edu
Boris,Rush,1-869-881-6318,ridiculus.mus@ipsumDonecsollicitudin.ca
Logan,Lowe,1-379-391-2234,leo.in@mauriseu.com
Chelsea,Burks,1-392-933-8867,ornare@sagittisNullam.org

Cathleen,Pena,1-118-332-7783,ad.litora.torquent@magnaSed.org
Keegan,Graves,1-597-373-0084,interdum.ligula.eu@ametorci.org
Remedios,Haley,1-916-694-0975,at@risus.ca
Xena,Cantrell,1-175-274-5592,justo@Maecenas.org
Chloe,Austin,1-215-212-7157,ac.libero@dolorNullasemper.com
Kasimir,Fitzpatrick,1-526-286-3834,fringilla@molestieSed.ca
Petra,Rios,1-280-806-6694,metus.Vivamus@Donecatarcu.ca
Cooper,Hines,1-101-484-8124,facilisi.Sed@cursus.com
Amber,Conley,1-308-977-3112,Ut.nec.urna@etlacinia.edu
Kaseem,Hyde,1-548-351-7939,Ut@mollisvitaeposuere.ca
Audrey,Barton,1-887-209-4998,imperdiet@tristiquealiquet.com
Sacha,Murphy,1-837-621-5384,ligula.tortor@estac.org
Hiram,Hartman,1-842-342-1715,lacus.Etiam@Namtempor.com
Palmer,Turner,1-190-538-0865,augue.Sed@etnetuset.edu
Iola,Carlson,1-383-555-0014,ridiculus@et.org
Cally,Bowen,1-249-335-8778,ultrices.Duis@Suspendisse.edu
Azalia,Brown,1-666-768-0842,sem.vitae.aliquam@etlibero.com
Signe,Tanner,1-614-492-2504,orci@tinciduntneque.ca
Chaney,Barr,1-688-763-9543,Aliquam.fringilla@Maecenas.com
Herrod,Schroeder,1-830-181-5140,dapibus.ligula@amet.ca
Bertha,Jenkins,1-778-723-0713,Donec@sitametconsectetuer.com
Alisa,Becker,1-592-106-1604,sollicitudin@velquam.org
Donna,Leon,1-458-530-7564,aliquet.magna@anteNunc.org
Inga,Roman,1-754-814-2347,neque@tincidunt.org
Lee,Hunter,1-294-940-4919,sed.consequat@atvelit.com
Tasha,Rodgers,1-302-547-2592,mauris@cursusinhendrerit.org
Ramona,Barton,1-621-122-0704,dictum@Morbi.org
Gillian,Cantu,1-278-428-6335,mollis.non.cursus@erat.ca
Kerry,Cannon,1-873-854-2686,et.magnis@anteblanditviverra.org
Odysseus,Lee,1-193-737-3386,consectetuer.cursus@natoque.com
Teegan,Vance,1-343-501-1561,vitae@vulputateduinec.com
Russell,Bowman,1-328-556-1923,et.magnis@nisi.ca
Madaline,Pace,1-568-625-7495,Nulla@adipiscingMauris.org
Kylie,Britt,1-217-309-6568,egestas.nunc.sed@nonhendrerit.edu
Baxter,Hopkins,1-639-379-8922,libero@ultriciesdignissimlacus.edu
Thor,Brock,1-490-947-7321,Mauris.molestie.pharetra@egestas.ca
Lilah,Guerra,1-232-135-3540,ac.mattis.semper@sollicitudinamalesuada.ca
Wynter,Bullock,1-848-681-7740,volutpat@acmattis.org
Zenia,Reynolds,1-633-987-0720,metus.In.nec@dictummiac.ca
Holmes,Robbins,1-429-293-2068,cursus.Nunc@montesnasceturridiculus.ca
Candace,Moreno,1-703-786-7676,urna.convallis@lacinia.edu
Evangeline,Duke,1-325-532-3941,pede@Phasellus.ca
Glenna,Everett,1-619-486-0321,odio.tristique.pharetra@auctorvelit.org
Noble,Mayo,1-108-492-5148,rhoncus@et.ca
Hillary,Ellis,1-329-461-5359,magna.Sed@ornarelectusante.edu
Nomlanga,Potts,1-569-183-9971,rhoncus@ligulaNullam.edu

Reuben,Hunter,1-446-971-8135,sagittis@libero.edu
Thor,Puckett,1-171-168-2957,dignissim.tempor@arcuSed.ca
Zeus,Michael,1-878-781-2964,id.libero.Donec@maurisSuspendisse.org
Burton,Mcmillan,1-884-980-7353,nisl@commodoat.com
Brendan,Payne,1-108-394-7701,Cras@Duis.org
Brent,Duke,1-692-428-7530,ultrices.posuere@cursus.ca
Ciara,Watkins,1-213-984-0023,libero.lacus.varius@odioAliquam.com
Harper,Guzman,1-862-615-6528,nec.enim@Duis.edu
Chandler,White,1-204-217-5969,non@vitaesodales.ca
Xavier,Nielsen,1-502-255-9269,pede.sagittis.augue@vitae.ca
Colette,Clarke,1-239-635-6369,suscipit.est.ac@arcuSedeu.edu
Alexandra,Reese,1-362-388-2573,eu.neque@nibhenim.ca
Vance,Hewitt,1-529-165-0410,condimentum@elit.org
Melyssa,Beck,1-849-106-4430,Nulla.tempor.augue@metusVivamuseuismod.org
Coby,Franco,1-788-830-8349,purus@euelit.ca
Audra,Cash,1-918-451-3533,orci.lobortis.augue@felisNulla.ca
Idona,Boyd,1-771-515-4250,sed.leo@bibendum.com
Echo,Jordan,1-686-140-4209,ut.dolor@hendreritDonec.org
Cassady,Randolph,1-233-659-6037,Cras.dolor.dolor@primis.com
Tiger,Terry,1-439-760-7091,turpis.nec@sed.org
Madeline,Quinn,1-326-939-6066,ipsum@Maecenaslibero.edu
Theodore,Buckley,1-151-761-8262,ac.risus.Morbi@Nunclaoreetlectus.org
Angela,Bond,1-530-236-8654,a.feugiat.tellus@euismodac.ca

Appendix F: Remote Data Script

When activated, the PHP script below will respond to the request by sending a large string that is a passage from Shakespeare's Romeo and Juliet [71]. This is used in the remote access test in order to establish a server connection, receive data and close the connection multiple times.

```
<?php
     echo "ROMEO: But, soft! what light through yonder window breaks?
    It is the east, and Juliet is the sun.
    Arise, fair sun, and kill the envious moon,
    Who is already sick and pale with grief,
    That thou her maid art far more fair than she:
    Be not her maid, since she is envious;
    Her vestal livery is but sick and green
    And none but fools do wear it; cast it off.
    It is my lady, O, it is my love!
    O, that she knew she were!
    She speaks yet she says nothing: what of that?
    Her eye discourses; I will answer it.
    I am too bold, 'tis not to me she speaks:
    Two of the fairest stars in all the heaven,
    Having some business, do entreat her eyes
    To twinkle in their spheres till they return.
    What if her eyes were there, they in her head?
    The brightness of her cheek would shame those stars,
    As daylight doth a lamp; her eyes in heaven
    Would through the airy region stream so bright
    That birds would sing and think it were not night.
    See, how she leans her cheek upon her hand!
    O, that I were a glove upon that hand,
    That I might touch that cheek!
    JULIET: Ay me!
    ROMEO: She speaks:
    O, speak again, bright angel! for thou art
    As glorious to this night, being o'er my head
    As is a winged messenger of heaven
    Unto the white-upturned wondering eyes
    Of mortals that fall back to gaze on him
    When he bestrides the lazy-pacing clouds
    And sails upon the bosom of the air.0"
 ?>
```

Appendix G: Benchmark Application Sorting Algorithm

The sorting function was implemented on various platforms. Below you will find code for the sort functions for WebWorks, PhoneGap, Android Native, iOS Native, MoSync, Titanium and BB10 Native. This code is written using numerous platforms but the algorithm is the same. The code is shown on each platform to demonstrate this. These sort functions complete the test outlined in Section 4.3.3.3 with 2500 random integers. This size is set as the internalIterations or testSize variable.

Sort (WebWorks and PhoneGap)

```
do{
          swapped = false;
          for(var i = 1; i < this.internalIterations; i++){
             if(sortArray[i-1] > sortArray[i]){
                var temp = sortArray[i-1];
                sortArray[i-1] = sortArray[i];
                sortArray[i] = temp;
                swapped = true;
             }
          }
}while(swapped == true);
```

Sort (Android Native)

```
do {
       swapped = false;
       for(int i=1; i<testSize-1; i++){
          if(test[i-1] > test[i]){
             temp = test[i];
             test[i] = test[i-1];
             test[i-1] = temp;
             swapped = true;
          }
       }
} while(swapped);
```

Sort (iOS Native)

```
do{
    swapped = false;
    for(int i = 1; i < internalIteratons; i++){
        if(sortArray[i-1] > sortArray[i]){
            int temp = sortArray[i-1];
            sortArray[i-1] = sortArray[i];
            sortArray[i] = temp;
```

```
                swapped = true;
            }
        }
}while(swapped == true);
```

Sort (MoSync)

```
do {
        swapped = false;
        for(int i=1; i<testSize-1; i++){
            if(test[i-1] > test[i]){
                temp = test[i];
                test[i] = test[i-1];
                test[i-1] = temp;
                swapped = true;
            }
        }
} while(swapped == true);
```

Sort (Titanium)

```
do{
        swapped = false;
        for(var i = 1; i < this.internalIterations; i++){
            if(sortArray[i-1] > sortArray[i]){
                var temp = sortArray[i-1];
                sortArray[i-1] = sortArray[i];
                sortArray[i] = temp;
                swapped = true;
            }
        }
}while(swapped == true);
```

Sort(BB10)

```
do {
        swapped = false;
        for(int i=1; i<testSize; i++){
            if(test[i-1] > test[i]){
                temp = test[i];
                test[i] = test[i-1];
                test[i-1] = temp;
                swapped = true;
            }
        }
} while(swapped);
```

Appendix H: Images for UI Testing

The images contained in these figures are used in the UI testing section of the performance benchmarks. The images in Figure H.1 and Figure H.2 were created by Creative House for CMER. The icons in Figure H.3 were developed by Justin Carvalho, an undergraduate student working for CMER. These images are used in their current form to test the ability for the UI engine of each CPDT to render and resize images quickly. In order to replicate these tests, the same images should be used.

Figure H.1: image1.png

Figure H.2: image2.png

Figure H.3: Icons for Menu

Appendix I: Results of Phase I Evaluation

Table I.1 contains the data found as part of the Phase I evaluation of the CPDTs included in this study in Section 5.1. This table provides the details of which features are supported by the various CPDTs and native tools.

Table I.1: Phase I Evaluation of CPDT Features

Tool	CPDT Basic Elements					Development Environment						
	Version Studied	Platforms Supported	Background Processes (Options for running in background)	Costs for development (initial and ongoing) Is there a free tier?	License	IDE type	Development Language	Compilation Type	Debug Environment	MVC Support	Social API's	Cloud API's
Mobile Web	HTML5 Working Draft	All	No	Free	Standard	Large variety of options	HTML 5, CSS, Javascript	Interpreted	Web Browser Developer tools	No	3rd Party	No
BlackBerry WebWorks	2.3.1	Blackberry Tablet OS, BlackBerry OS, BlackBerry 10 OS	Yes	Free	Proprietary with Open Source Components	Large variety of options	HTML 5, CSS, Javascript	Hybrid	Ripple Emulator, BlackBerry Simulator	No	No	No
Blackberry Cascades Native SDK	Beta 1	BlackBerry 10 OS	Yes	Free	Proprietary with Open Source Components	QNX Momentics	Qt or QML and JavaScript, or both	Native	QNX Momentics	You can separate UI from logic using QML	Scoreloop gaming APIs	No
Android Java	1.0 - 4.0	Android	Yes	Free *Other Google services may have a cost if you exceed the free quota	*OS: Apache License *SDK: http://developer.android.com/sdk/terms.html *Google APIs have additional licensing	Eclipse plugin, JBuilder plugin, or IDE-less development	Android's custom version of Java, and XML	Native	Eclipse with Dalvik Debug Monitor Server (DDMS), can be run from Eclipse, JBuilder, or command line.	It can be done depending on how you want to define the separation between View and Controller, but it has been said that MVP is a better fit for Android	Native "Contacts" API since Android 4.0; others (Facebook, Twitter, etc.) accessible through their respective 3rd party web APIs	Data backup
iOS Native	5.1	iOS	No	$99/year	Proprietary	Complete (Xcode)	Objective C	Native	Xcode	Yes	No	Yes
Adobe PhoneGap	1.8.1	iOS, Android, Windows Phone, Blackberry OS, WebOS, Symbian, Bada	No	Free	Apache	Large variety of options	HTML 5, CSS, Javascript	Hybrid	Web Browser Developer tools and device specific tools	No	Via Plugin	No
Appcelerator Titanium	Titanium Mobile SDK 2.0.1.GA2	iOS, Android, Mobile Web, Blackberry OS (beta)	Yes (non-guaranteed)	Free + Support costs	Apache, Proprietary	Titanium Studio	Javascript	Native with interpreter for dynamic segments	Titanium Studio	Yes	Yes	Yes
Adobe Air	FLEX 4.6	Playbook, BB10, iOS, Android	iOS: no. Android: yes, at a reduced framerate.	SDK is free, Flash Builder costs $699	Proprietary	Flash Builder	Actionscript 3.0, XML	Runtime	Air Debug Launcher	Information not readily available	Accessible through imported ActionScript API's	Accessible through imported ActionScript API's
MoSync	MoSync 3.0+	Android, iOS, Windows7 Phone, JavaME, Moblin, Symbian, Windows Mobile, BlackBerry	At least for Android	*Free - Community version *Free - Free version (requires registration annually renewed) *199EUR/developer/year - Basic Pro *2999EUR/developer/year - Gold Pro *Custom price - Platinum Pro	Dual Licensed: GPL2 and various commercial licenses	MoSync Eclipse IDE	C/C++, HTML5/JavaScript/CSS, or a mix of both (a technology called Wormhole connects the underlying C code to the JavaScript)	Depends on the platform. iOS is native code. WP7 is bundled with a recompiler. Android likely has a VM interpreting the MoSync bytecode	MoSync Eclipse IDE, with native simulators or with MoRE (MoSync Reference Environment), which creates a general emulator that bases its properties off of device data. *Testify (replaced MATest) is the test framework, utilizing hooks and functions.	Information not readily available	Facebook (using FacebookManager). Wikipedia, Twitter, etc. accessible through their respective 3rd party web APIs	None currently apparent

Tool	Build service availability	User Experience						Device Access				
		Access Native UI Elements	Screen Rotation	Swipe	Pinch	Accessibility Features	Ability to playback media	File System Access	SMS Access	Call log access	Contacts	Calendar
Mobile Web	N/A	No	No	Yes	Yes	No	Partial Compliance on most devices	Draft. Partial implementation in Android, Blackberry and Windows Phone	Draft, unimplemented	No	Draft, unimplemented	No
BlackBerry WebWorks	No	No	Yes	Yes	Enable/Disable pinch to zoom only	Information not readily available	Yes	Yes	Yes	Yes	Yes	Yes
Blackberry Cascades Native SDK	No	Cascades elements	API added in Beta 2	Can be manually handled	Can be manually handled. API added in Beta 2	Information not readily available	Yes	Yes	Information not readily available	Information not readily available	No	No
Android Java	No	Yes	Yes	GestureDetector	ScaleGesture Detector	TextToSpeech (Android 1.6+)	Yes	Yes	Yes (using SmsManager, since Android 1.6)	Yes	Yes using ContactsProvider	Android 4.0+, prior to this there was only the Google Calendar APIs.
iOS Native	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Adobe PhoneGap	Yes, PhoneGap Build	No	No	No	No	No	Yes	Yes	No	No	Yes	No
Appcelerator Titanium	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes	Android only
Adobe Air	No	Yes, libraries are included in the SDK's	Yes	Yes	Information not readily available	No	Yes	Yes, on Playbook, BB10, and Android.	Using SMS: URI scheme, or possibly imported ActionScript APIs.	No	Yes	No
MoSync	Information not readily available	Platform Dependent	Information not readily available	Information not readily available	Information not readily available	Information not readily available	HTML5: (video) Devices that support HTML5 video tag C++: iOS, Android. C: (audio) For Sound class (play single file), all platforms but Windows7Phone	HTML5: iOS Android. C++: All. C: All	Yes	No	For all supported platforms except for Windows Mobile, Windows 7 Phone, Moblin.	Information not readily available

Tool	Device Access						Sensor Access					
	Low level Network Access	Ability to choose data transport method (Cellular or Wi-Fi only)	GPU Acceleration	Bluetooth	Voice Activation	Notification Light Activation	Camera	Microphone	Noise Cancellation Microphone	Sensor Data Capture	Proximity	NFC
Mobile Web	No	Information not readily available	No	No	Information not readily available	Information not readily available	Draft	Draft, unimplemented	No	Draft, unimplemented	No	No
BlackBerry WebWorks	No	Information not readily available	No	No	Information not readily available	Information not readily available	Yes	Yes	No	Yes	No	Via RIM extension
Blackberry Cascades Native SDK	Information not readily available	Information not readily available	Information not readily available	No APIs, but planned for future release	3rd party libraries	Yes. Supports Red, Green, Blue, Yellow, Cyan, Magenta, and White	Yes	Information not readily available	Information not readily available	Information not readily available	Information not readily available	Added in Beta 2
Android Java	Yes	Yes	Hardware acceleration for 2D drawing added in Android 3.0	Android 2.0+	Android 2.2+	Yes. Colors are device-dependent (though hardware will try to estimate closest color)	Single camera pre-Android 2.3, multiple camera support in Android 2.3+	Yes	No	Yes	Since Android 1.5 (on devices with the proper hardware)	Since Android 1.5 (on devices with the proper hardware)
iOS Native	Yes	Information not readily available	Yes	Bluetooth 4.0 LE	Information not readily available	Information not readily available	Yes	Yes	No	Yes	Yes	No
Adobe PhoneGap	No	No	No	No	No	No	Yes	Yes	No	Yes	No	Via Plugin
Appcelerator Titanium	Yes	No	No	No	Android Only	Android Only	Yes	Yes	No	Yes	Yes	No
Adobe Air	Yes, UDP not supported on mobile	Information not readily available	Since AIR 3.2	No	Information not readily available	Information not readily available	Yes	Yes	No	Yes	No	Through Native extensions
MoSync	Yes. No UDP support, due to mobile operators feeling insecure about it being abused by p2p apps and such (even though it could be used on WiFi networks).	Information not readily available	OpenGL ES for Android and iOS.	Information not readily available	Information not readily available	Information not readily available	Only supports one camera for Android currently. C: most Android, most iOS, few Windows7Phone. *According to IDE: all but BlackBerry	No. Experimental (i.e. may not work at all) API available for recording, but not streaming	No.	HTML5: (image/video) Android iOS. C: (images/video) Android, iOS, Windows7Phone, Java2ME (some features), Moblin(some features)	C: [using sensorStart() with SENSOR_TYPE_PROXIMITY] Android, iOS	According to IDE, only has support for Android

Tool	Sensor Access				Geolocation				Notifications			Monetization
	Accelerometer	Gyroscope	Barometer	Compass	Wi-Fi Positioning	Cellular Positioning	GPS	Native Map Support	User Notifications	System Notifications	Push Notifications (Specify if using Native server or 3rd Party)	App Store Support
Mobile Web	Grouped as Device Orientation. Implemented on latest Bada, Android, iOS and Blackberry devices	Grouped as Device Orientation. Implemented on latest Bada, Android, iOS and Blackberry devices	No	No	Grouped as Geolocation. Widely implemented	Grouped as Geolocation. Widely implemented	Grouped as Geolocation. Widely implemented	No	Draft, Partial implementation in latest Blackberry	No	Draft, Implemented in latest versions of Bada, iOS, Blackberry	No
BlackBerry WebWorks	Only Playbook supported, Grouped as Device Motion	No	No	No	Grouped as Geolocation	Grouped as Geolocation	Grouped as Geolocation	Yes	Yes	Yes	Yes	Yes
Blackberry Cascades Native SDK	Information not readily available	Information not readily available	Information not readily available	Information not readily available	No	Grouped as Location	Grouped as Location	No	Information not readily available	Information not readily available	Information not readily available	Currently only for those with the Dev Alpha device.
Android Java	On devices that have one, and are running Android 1.5+	Since Android 2.3 (on devices with the proper hardware)	Since Android 1.5 (on devices with the proper hardware)	Since Android 1.5 (on devices with the proper hardware)	Yes (grouped under NETWORK PROVIDER option)	Yes (grouped under NETWORK PROVIDER option)	Yes (grouped under GPS PROVIDER option)	Yes	Yes	Yes	Native "PUSH", using Google's Cloud To Device Messaging (C2DM) service, which you have to register with. Android 2.2+	Yes
iOS Native	Yes	Yes	No	No	Grouped as Geolocation framework	Grouped as Geolocation framework	Grouped as Geolocation framework	Yes	Yes	Yes	Yes	Yes
Adobe PhoneGap	Yes	No	No	Yes	Grouped as Geolocation	Grouped as Geolocation	Grouped as Geolocation	No	Yes	Yes	3rd Party	Yes
Appcelerator Titanium	Yes	No	No	Yes	Grouped as Geolocation	Grouped as Geolocation	Grouped as Geolocation	Yes	Yes	Yes	Yes	Yes
Adobe Air	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes	Through Native extension, if supported	Yes	Yes
MoSync	Sensors currently only supported on Android and iOS. Emulator does not currently support sensors. *According to the IDE: iOS, Android, and Windows Phone supported.	C: [using sensorStart() with SENSOR_TYPE_GYROSCOPE] Android, iOS. *According to IDE: Android, iOS, and Windows Phone	No	HTML5: iOS, Android, Windows7Phone. C: (using sensorStart()) Android, iOS	On Android devices that support it, but only in select countries	Information not readily available	HTML5: Android, iOS, Windows7Phone. C: Android, iOS, Windows7Phone. *According to IDE: (under Location) all.	Information not readily available	HTML5: (Beep) iOS Android. (Vibrate) iOS, Android, Windows Phone 7. C++: iOS and Android. C: iOS and Android. *iOS uses badge notifications, Android uses StatusBar notifications. You have to handle both cases manually	iOS and Android. Eg. Calendar events can open your app	Native for Android (using Google's Cloud To Device Messaging service) and iOS	Yes (at least for Android, iOS and Windows 7 Phone). Registering with the respective sites, obtaining signing keys and passing App inspections still applies

Tool	Monetization			Security	
	In App Purchases (Native or 3rd party)	Mobile Ad Platform support (Native or 3rd Party)	Analytics Platform Provided/Compatibility	Access to Secure Storage	Code Obfuscation
Mobile Web	3rd Party	Normal advertising services	Traditional web analytics only	No	No
BlackBerry WebWorks	Yes	Yes (Beta)	Yes, RIM API	No	No
Blackberry Cascades Native SDK	Yes (FREE). Only available for Apps distributed through BlackBerry App World, requires a BlackBerry App World vendor account	Information not readily available	Information not readily available	Restricted access	Information not readily available
Android Java	Yes. Only available for Apps distributed through Google Play, and requires a Google Wallet Merchant account. A 30% fee will be taken from the amount by Google	Yes	Yes, if published through Google Play (for number of downloads and that kind of information)	Access to App data restricted to only that App. Data encryption libraries included. Note that a rooted device will expose all saved files, and thus encryption of sensitive data is encouraged.	ProGuard tool included with SDK automatically obfuscates code when App is built in release mode
iOS Native	Yes	Yes	3rd Party available	Encryption available	Yes
Adobe PhoneGap	3rd Party	3rd Party	Can make use of web analytics or third party plug-ins	No	No
Appcelerator Titanium	Via Module (Official)	Yes	Yes	No	Yes
Adobe Air	Yes	Yes	Only through third-party tools/plugins like appAnalytics	Encrypted Local Store (ELS), which uses KeyChain on iOS, but let's Android's user-level security handle it, which on a rooted phone is not secure	Yes
MoSync	No (MoSync 3.1 added support for in-app purchases for iOS and Android).	Android and iOS Native support (in C and C++). Using Google AdMob. (InMobi support added in MoSync 3.1 for JavaScript/HTML5 apps).	MoRE has performance statistics. Other analytics likely available through third-party tools.	Partial implementation. Developer enhancement required	Information not readily available