NTNU – Trondheim
Norwegian University of
Science and Technology

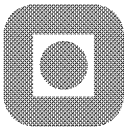
Web applications - New mobile service paradigm

Phuc Huy Ngu

Master in Security and Mobile Computing
Submission date: June 2012
Supervisor: Van Thanh Do, ITEM
Co-supervisor: Antti Ylä-Jääski, AALTO university

Norwegian University of Science and Technology
Department of Telematics

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY, MATHEMATICS AND ELECTRICAL ENGINEERING



MASTER THESIS

Student's name:	Ngu Phuc Huy
Course:	TTM4905
Project title:	Web applications - New mobile service paradigm
Project description:	Recently, smartphones with iPhone in the lead experience a huge popularity and the number of smartphones in the market has increased considerably. The reason behind this popularity is the plurality of useful and fancy applications, also called Apps. Although apps may have the same functionality there are many ways of implementing them such Javascript, HTML5, applets, widgets, etc and the situation is rather confusing. This Master thesis work focuses on the development of mobile applications and aims at shedding light on the different mobile applications: paradigms such as native apps, mobile Web, widget and HTML5. The main task of the work is to analyze and evaluate the feasibility of these mobile application paradigms. To verify the evaluation an application will be implemented using the two best mobile application paradigms and comparisons will be carried out. The main tasks of the Master Thesis work are as follows: • Analysis and evaluation of the mobile application paradigms on the mobile phone according to the applicability and feasibility of the paradigms. • Selection and implementation of an application using the two best mobile applications The thesis will contains both theoretical and practical work.
Department:	Department of Telematics
Responsible professors:	Do Van Thanh Antti Ylä-Jääski

Submission date: 2012-06-22

Abstract

The explosion of mobile applications both in number and variety raises the need of shedding light on their architecture, composition and quality. Indeed, it is crucial to understand which mobile application paradigm fits better to what type of application and usage. Such understanding has direct consequences on the user experience, the development cost and sale revenues of mobile apps. In this thesis, we identify four main mobile application paradigms and evaluate them from the viewpoints of developers, users and service providers. To ensure objectivity and accuracy we start by defining high level criteria and then breaking down into finer-grained criteria. After a theoretical evaluation an implementation was carried out as a practical verification to ensure that the method adopted in analysis and evaluation is trusted and applicable. The selected application is object recognition app, which is both exciting and challenging to develop.

Preface

Thinking of writing a thesis and starting to write it are absolutely different things. This thesis is impossible without the support of many people. I cannot find words to express my gratitude to Professor Do Van Thanh for all his guidance, assistant and invaluable feedbacks. Professor Do Van Thanh helps me initiate the structure of the thesis, gives me advice on scientific writing and shares some of his limited time with me during my time as a student at NTNU. I would also like to thank Professor Antti Ylä-Jääski for all of his instant feedbacks with compliment and encouragement, regardless of the geographical distance.

I want to gratitude NordSecMob consortium for letting me join in the programme and obtain a very generous Erasmus Mundus scholarship. Two years studying in Nordic countries give me the so many valuable things that I cannot find anywhere else. I also want to especially thank Misela Väisänen and Mona Nordaune for their seamless support in the programme.

Last but not least, I would like to thank my beloved families and my friends, who stood by me through the duration of my study.

Trondheim, May 2012

Ngu Phuc Huy

List of abbreviations and terms

HTML	Hypertext Mark-up Language
CSS	Cascading Style Sheets, the language used to describe the presentation of structured documents in the Web.
SMS	Short Message Service
WAP	Wireless Application Protocol
MMS	Multimedia Messaging Service
GPS	Global Position System
J2ME	Java 2 Micro Edition
Mbs	Megabytes per Second
XML	Extensible Mark-up Language
Nokia WRT	WRT S60 Web Runtime, a runtime platform for S60 used to execute.
AJAX	Asynchronous JavaScript and XML
http	Hypertext Transfer Protocol
DOM	Document Object Model, an API and a data model for representing (X)HTML and XML documents.
API	Application Programming Interface, a convention for accessing functionality of a computer program.
IDE	Integrated Development Environment
SDK	Software Development Kit
JDT	Java Development Tool
ADT	Android Development Tool
LLVM	Low Level Virtual Machine, the next-generation compiler technology powering Xcode 4, which compiles code twice as quickly as GCC and produces the application that also runs faster. The compiler was built from the ground up as a set of highly optimized libraries that are easy to extend, optimize, and designed for modern chip architectures.
GCC	GNU Compiler Connection
UI	User Interface
MSDN	Microsoft Developer Network
PC	Personal Computer

SVG	Scalable Vector Graphics, an XML-based markup language for defining vector graphics.
PIM	Personal Information Manager, a portable appliance.
SSL	Secure Socket Layer, cryptographic protocols that provide communication security over the Internet.
URI	Uniform Resource Identifier, a compact character string identifying or naming a resource.
W3C	World Wide Web Consortium, the main international standards organization for the World Wide Web.
USB	Universal Serial Bus
OEM	Original Equipment Manufacturers, the companies that originally manufactured the product.
ASP	Active Server Page, a web-scripting interface by Microsoft.
PHP	A General-purpose server-side scripting language originally designed for Web development to produce dynamic Web pages.
JSP	Java Server Page, a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 by Sun Microsystems.
HVS	Human Visual System
JSON	JavaScript Object Notation, a lightweight format for interchanging data.
JavaScript	A scripting language often used for client-side web development, a dialect of the ECMAScript.

Table of content

List of Figures

List of Tables

Chapter 1: Introduction

1.1. Background

In the recent years, we have seen an enormous growth in the popularity and visibility of smartphones. Smartphone sales to end users reach 115 million units in the third quarter of 2011, up to 42% from the third quarter of 2010. Smartphone changes the way people interacts with mobile phone and now become a vital part of human being. The smartphones that people use in their daily lives can run advance applications and come equipped with very powerful hardware features. In addition to calling and messaging, users can enjoy a countless number of native apps installed on their mobile phones [1], such as weather forecast, dictionary, Google Map, movies player, social networking, currency converter, calculator and game. At the same time, application distribution has increased remarkably because of the proliferation of app stores [67]. In June 2010, there are around 80 would-be application stores available worldwide according to market research firm Foresters [66]. Android market has 70,000 apps, BlackBerry store has 7,200 apps, and the number of apps at Apple app store reaches 225,000. Furthermore, a great number of mobile Web apps hosted on Web servers deliver Web content to mobile phones and serve mobile users in a different way with native apps.

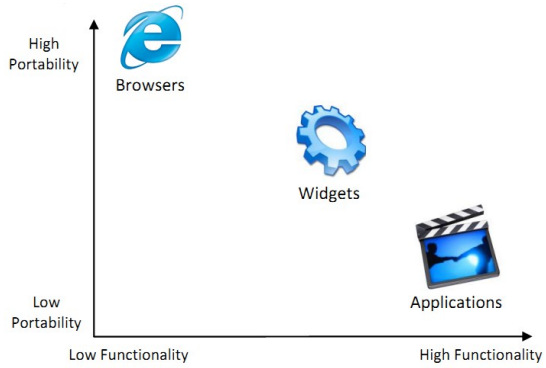


Figure 1: Mobile app landscape [35]

When we face with thousands of apps across a dozen of different platforms [2], the need of shedding light on their architecture, composition and quality is crucial. Indeed, it is very important to understand which mobile application paradigm fits better to what type of application and usage. Such understanding has direct effects on user experience, development cost, and sale revenue of mobile apps. A few years ago, people have to consider between the functionality and the portability when they deal with a mobile app as shown in Figure 1. The higher functional the apps are the lower portability they expose. However, HTML5, which is introduced by W3C in 2009, currently changes the mobile phone app landscape. It makes mobile Web app more robust but still keep the portability, creating seamless user experience.

1.2. Problem statement

The target of the master's thesis is to identify, analyse and evaluate different mobile application paradigms to show how Web app changes into a very competitive and promising mobile paradigm. In our thesis project, we identify four mobile app paradigms, including **native apps**, **mobile widgets**, **mobile Web apps** and **HTML5 mobile apps**. From the viewpoints of developers, users and service/content providers, we subsequently carry out the analysis and evaluation on the paradigms based on different criteria. The **outcome** is the decision on which paradigm a user, a developer or a service/content provider should choose in a specific development and usage circumstance. We also perform the practical verification by building an app for the two most promising paradigms, then analyse and evaluate the app to verify that the method adopted is trusted, feasible and applicable.

1.3. Structure of the thesis

The rest of the report is structured as follows

- **Chapter 2** reviews the literature on different mobile app paradigms, including native apps, mobile widgets, mobile Web apps and HTML5 apps.
- **Chapter 3** reviews the literature on object recognition.
- **Chapter 4** performs the analysis and evaluation from the viewpoint of developers, users, service/content providers.
- **Chapter 5** performs the practical verification on the paradigms.
- **Chapter 6** concludes the report and make some suggestion for the future work

Chapter 2: Application paradigms on mobile phones

In this chapter, we will identify the application paradigms for mobile devices, such as native application, widget, Web application and HTML5. We then briefly discuss the advantages and disadvantages of using each paradigm in the development and usage of mobile apps.

2.1. Native applications

A mobile native application or native app is an application program specifically developed to execute on a specific device platform [69] and machine firmware, and cannot be used for other device platform without modifications. For example, apps developed for the iPhone run only on Apple devices. Android apps are more portable in this case because they can run on different hardware such as Archos, Samsung and LG [58, 70]. To use native apps users must download them from app store and install them manually on their phones as shown in Figure 2.

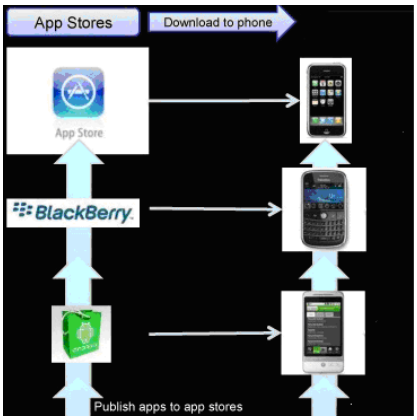


Figure 2: Native app distribution

Native apps can take full advantage of device's capability (e.g. gyroscopes, cameras, microphones, speakers and GPS) [50] and native APIs. Such advantage makes native apps more robust and functional than any other mobile app paradigms. Moreover, native app allows mobile users to store their data locally and load the data instantly for offline use, reducing the time and cost to transfer data via Internet. It also supports multitasking (e.g. in iOS 4) and creates many business opportunities for app vendors (e.g Apple app store and Android marketplace)

However, native app exposes many drawbacks. Platform diversity or platform fragmentation is the biggest problem of creating and deploying native apps on mobile phones [64]. Platform diversity means the differences in platform/OS (e.g. Symbian, iOS, Windows Phone, BlackBerry and Android) and APIs. It creates many challenges for developers, users and service providers. For example, in order to deploy a service to end users, service providers have to build three app versions for the same service to deploy on three different platforms such as Android, iOS and Windows Phone. The second issue is that uses must pay to download and install native apps on their mobile phones. The cost

associated with building a new app is obviously another important concern. Native apps demand large investment since they require a specific set of tools and expertise to develop.

2.2. Mobile Widgets

Mobile widgets represent lightweight, task-specific apps that leverage Web content [3]. Mobile widgets exploit web technologies, including HTML, CSS, JavaScript and XML. Normally, they do a particular task in an efficient and user-friendly way, for example currency converter and weather forecast. Widgets will be executed within a runtime environment known as **widget engine**. Widget engine does not have the browser UI elements and other functionalities, such as back button, URI input field and history. Different types of widgets need different widgets engines to execute [59] (e.g. Opera [8], Nokia WRT [9], Samsung TouchWiz and Yahoo!Blueprint). The W3C widget's family of specifications contains many specifications to achieve a standard for widgets and to remove the lack of interoperability among widgets engine [4, 68]. The relevant specification documents are Client-Side Web Applications (Widgets) Requirement [5], Widgets 1.0 Requirements [6, 7].

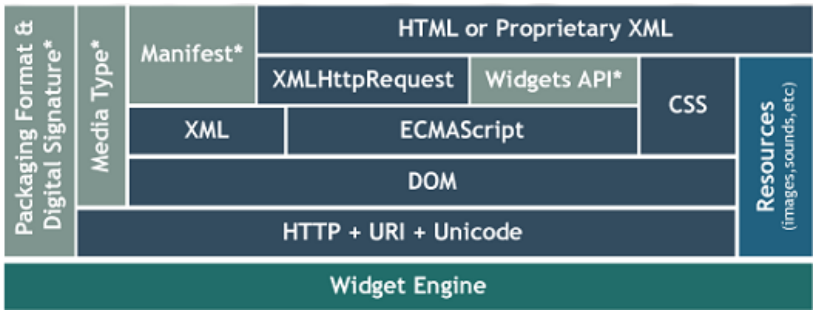


Figure 3: Widget structure [10].

Normally, a widget is a packaged file that contains many **HTML**, **CSS**, **ECMAScript (JavaScript, Jscript and ActionScript)**, **image**, **sound** files, and a manifest file to describe the metadata of the widget as shown in the Figure 3. Widget is created with the standard packager, for example ZIP. Depending on the widget vendor, the extension of single widget file will be renamed in order to be recognized by the widget engine. The packaged widget file can then be ported and installed on mobile phones. A media type (also referred to as a content type) helps to distinguish the widgets of different vendors. If the appropriate widget engine is installed on the phones and the media type is identified then the browser will automatically associate the resource with the widget engine. Furthermore, as **AJAX** grows fast, many widget engines currently support **DOM**, **XML**, **XHTML** and **XMLHttpRequest** objects to asynchronously exchange data with server over HTTP [13]. Some widget engines need to be downloaded and installed manually by users on mobile phones but some are provided by device's vendors on the phone.

Widget can access the resources of mobile phones. Mobile widgets also enable personal customization of data and creation of applications with high usability, and present a richer content experience. They drive a new approach to interact with a Web service on mobile phone, which is much more efficient than mobile Web app does. An example of such powerful ability is loading stock information from a financial service Website to

mobile phones. Users can sign up for the widget that delivers the stock information to the phones. In that way, only the data content is retrieved from a Web server very conveniently during runtime.

From the business perspective, mobile widgets can potentially deliver big profit to **device vendors** and **service providers**. By adopting the widget approach mobile apps can be developed quickly, deployed widely and hence very profitable [14]. This benefits many devices vendors (e.g. Apple, Samsung, Sony Ericsson and Motorola) in the way that the apps enable the vendors to enhance their device's quality and improve the revenue from device sales. Service providers now can deliver their content to mobile users and earn revenue in many ways. They build their own APIs and allow third party developers to create mobile widgets using the APIs to access their service. Service providers may also build their widgets and deliver the widgets to mobile users via app store or their own Website.

However, the incompatibility across widget engines is the biggest challenge in deploying widgets on the mobile phones [11, 63]. The main incompatibility issues are:

- Manifest document: Metadata fields and the manifest's filename differ between vendors.
- Packaging: It includes the packaging format (e.g. ZIP), file extension and internal package structure.
- Platform: Widget engines support different platform features and furthermore the API for accessing those features is not standardized.

Unless the efforts of W3C and many device vendors to standardize mobile widgets, interoperability across widget engines makes widgets less usable and attractive [12]. Moreover, widgets must be downloaded and installed on mobile devices. It means that mobile devices should have operating system (or mobile platform) and an equivalent widget engine to run the mobile widget.

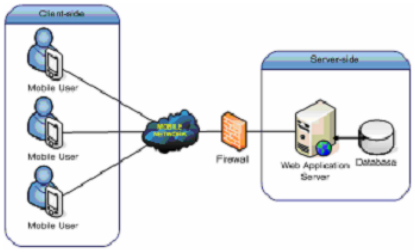


Figure 4: Mobile Web app architecture

Figure 5: Geolocation example

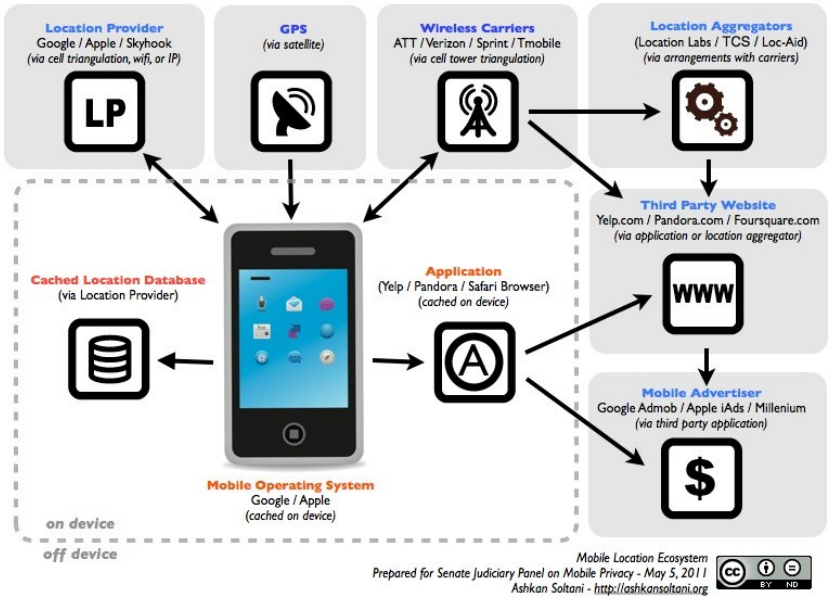


Figure 6: HTML5 capabilities on mobile platform [19]

2.3. Mobile Web applications

Mobile Web app is a good paradigm to deliver information and service to mobile phones. It enables information processing functions to be initiated remotely on Web servers. The three-tiered architecture [47] is the most popular Web app architecture, which consists of thin client layer (mobile devices), application layer (Web server) and database layer storing the data accessed by Web app. Figure 4 describes how mobile Web app works. With the architecture, a single mobile Web app targets wide variety of mobile phones that have JavaScript and CSS enable browser. Client side includes a mobile browser to send HTTP request or XML/HTTP request to Web server. Web server will process the request, collect the data from database server and reply the mobile client with HTTP or XML response. Browser will display the result on the screen of the device.

Mobile Web app reduces the workload at client's side very effectively, solving many issues of a mobile phone, for example battery and computing resource limitation [65]. Additionally, users do not have to install any software application and pay any fee for downloading and deploying the apps. Instead, they can simply access mobile Web apps by using mobile browser. However, one of the main problems of mobile Web app is the presentation at device's side because the screen size is very small and limited [51, 62]. Another problem of using mobile Web apps is the low performance and unattractive user interface of the apps.

2.4. HTML5 mobile applications

An HTML5 mobile app is also a mobile Web app in which HTML5, JQuery Mobile and CSS3 work as client's presentation technologies, instead of HTML4, JavaScript and CSS. HTML5 changes the way mobile Web apps work. It is designed by W3C to create a standard with a set of features that can handle all the tasks that the current technologies (e.g. Adobe System Flash, Apple Quick Time and Java Oracle FX) can do in a mobile

Web app. Additionally, HTML5 supports newer mobile technologies, such as Geolocation [15], Location Base Services, Scalable Vector Format and SVG. Therefore, mobile developers are able to develop mobile Web apps without the necessity of mastering multiple proprietary technologies at the same time. The question is what HTML5 offers mobile developers. Figure 6 summarizes the capabilities of HTML5 that benefits the mobile Web app.

- Canvas [49]: Canvas enables mobile developers to create and incorporate graphic, video and animation on Webpage, normally via JavaScript. The Canvas element supports 2D graphic. Graphic and animation will be rendered on mobile client instead of Web server. This helps avoid the bottleneck at server side and the restriction on network bandwidth, and make the graphic-heavy page render faster.
- Video tags [48]: HTML 5's codec neutral video tags provide a way to include non-proprietary video formats (e.g. Ogg Theora and H.264) in a page. The tag and underlying code tell the browser that the associated information is to be handled as an HTML 5-compatible video stream. They would also let users view video embedded on a Webpage without a specific video player.
- Location-based services [20, 21]: A location API offers support for location-based-service applications by enabling effective interaction with hardware related features of mobile devices, for example GPS [16]. Figure 5 depicts a sample application of Geolocation which tries to get the events from user's Facebook by using graph API, indicate current location by adopting the GPS of the mobile device and find out a route from the current point to the place of the event on Google Map.
- Working offline: HTML5 includes explicit support for offline execution of mobile Web application. The key features are the application cache and cache manifest. The manifest file is the list of items which server needs to send to application cache so that the application can be launched using the cache content instead of contacting the Web server. HTML 5 has several features that support building mobile Web applications that work offline [54]. These include support for a client-side SQLlite database and for offline application and data caching. The application cache benefits the mobile application in two ways. The first is the reduction in number of time transferring and fetching data from the server to mobile devices [17], enabling Web applications to work as native applications, even without an Internet connection. Furthermore, using application cache, mobile developer can address the concerns about network bandwidth and cost of transferring data via mobile devices.
- Web workers: They provide a standard way for mobile browser to run JavaScript in the background. The script will not be interrupted by other scripts or user interaction [18]. In order to simplify the challenge of writing such multithreaded application on mobile platform, Web workers elements communicate by sharing messages not by sharing state. The purpose of Web worker is to make the mobile Web application more responsive, thereby improving user's experience [52]. The background script can do either simple or complicated tasks, such as

mathematical calculations, network requests and cache accesses while the Web page still responds the scrolling, clicking or typing of the mobile users.

However, mobile developers face some challenges when developing HTML5 mobile apps. The biggest concern that HTML5 exposes is the limited access of HTML5 to hardware and native APIs. Moreover, HTML5 data storage on mobile device is not as powerful as it is in desktop computer. The security of HTML5 local database is also a problem [53]. If an attacker can manipulate the JavaScript code, they can easily intrude the database. Additionally, if mobile Web app can work offline then the synchronization between mobile cache and database on Web server is a big issue, requiring a good version management and intelligent synchronization mechanism.

Chapter 3: Object recognition

In this chapter, we will discuss about computer vision and object recognition. We introduce the two most popular approaches for object recognition, thereby providing a background for the practical verification in chapter 5.

3.1. Computer vision

Computer vision is the science that develops the theoretical and algorithmic basis by which useful information about the world can be automatically extracted and analyzed from observed image, a collection of images (e.g. stereo vision) or an image sequence (motion analysis) [55]. Computer vision involves the processes which extract, characterize and interpret information from images of the real world. The goal of computer vision is to build computer based vision system which provides the same functionality with the human eyes.

A typical vision system consists of the following modules:

- Sensing: The process that produces a visual image
- Preprocessing: Dealing with techniques such as noise reduction and enhancement of details
- Segmentation: The process that partitions an image into regions of interest
- Description: Dealing with the computation of features (size, shape, etc) suitable for distinguish one object from the others.
- Recognition: The process that identify the objects.

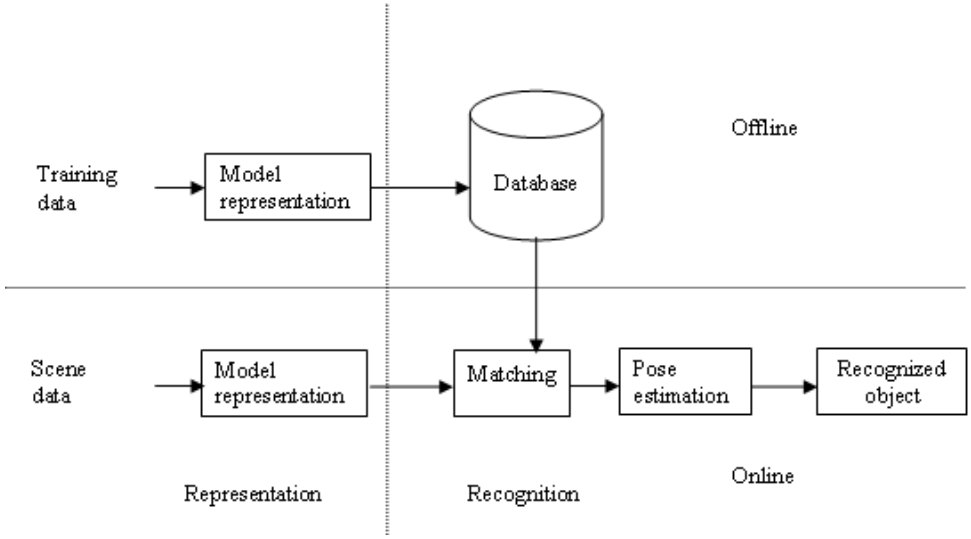


Figure 7: Object recognition model. In the figure, we assume that the scene has the same representation as the model in the database.

3.2. Object recognition

In computer vision, object recognition is concerned with determining the identity of an unknown object being observed in the image from a collection of known labels. The aim of object recognition is to detect the generic objects in images taken under different imaging conditions (e.g. viewpoint, illumination and occlusion) [56]. Several single objects have been detected successfully, including handwriting digits, fingerprints, faces and road signs. Additionally, significant progress towards object categorization from images has been made in the recent years. Therefore, object recognition has been studied extensively in psychology, computational neuroscience and cognitive science.

However, artificial object recognition faces many challenges [57]:

- The details of human visual and perceptual processes are both myriad and incompletely understood, creating many difficulties to provide a mathematical description for these processes.
- Extensive research activities in psychology are performed to find out how the brain functions but this is not an easy task due to the complexity of the brain which has more than 10 billion neural cells with very complex interconnection.
- The expectation is that the artificial vision system, which try to collect 3D information about the real world mostly from 2D image, is capable of achieving similar result of a human being. Unfortunately, this is impossible because humans are equipped with two eyes which make the human visual system (HVS) for perfect stereo vision and consequent estimation of depth. Another advantage of human being is that the human eyes are mounted inside a socket and are allowed to rotate and thereby, control the field and direction of view.
- Invariance of the HVS to rotation, size and position of the object under consideration is another problem that recognition system attempt to model.

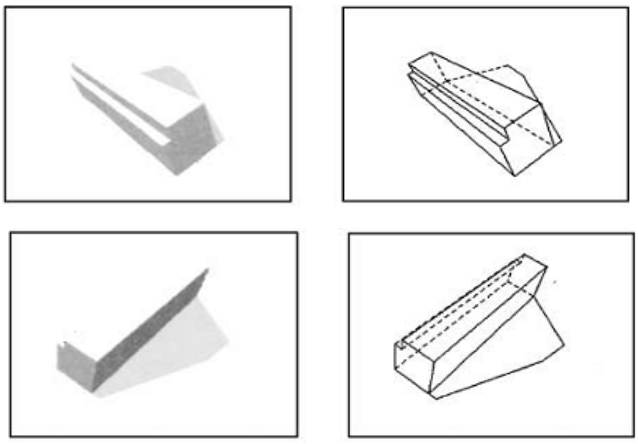


Figure 8a: Target model (training data)

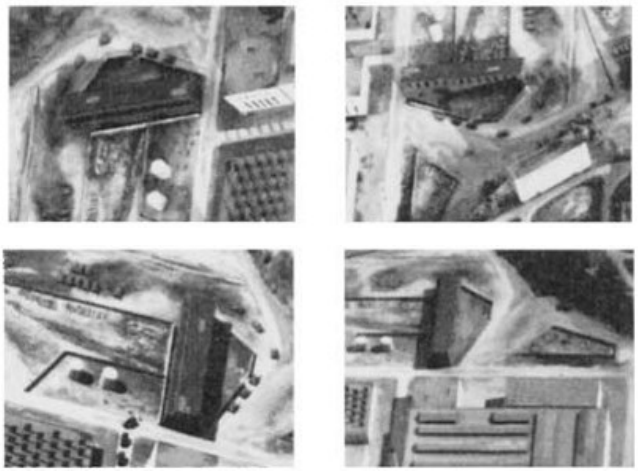


Figure 8b: Aerial images as input images

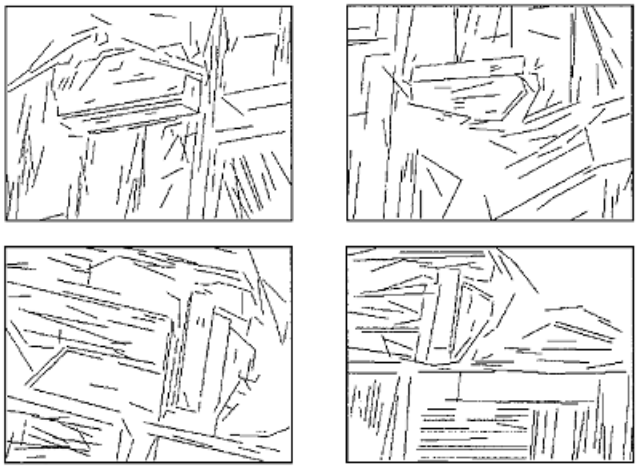


Figure 8c: Extracted line segments

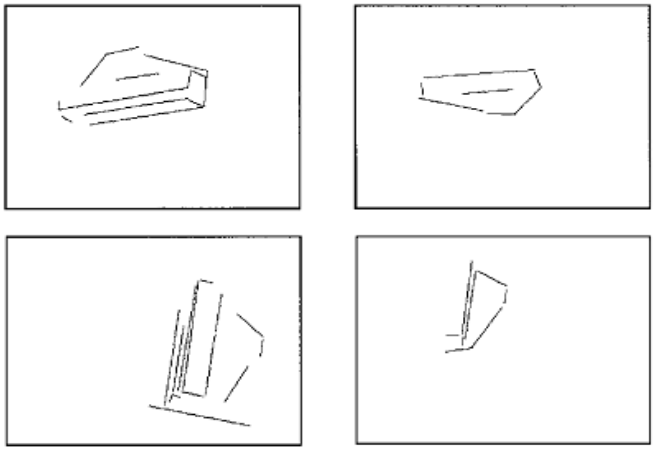


Figure 8d: Matching results after recognition process

The basic model based recognition system starts with the formation process that reduces dimensions of the object via an orthographic or perspective projection. For example, the 3D scene is reduced to 2D image array. After preliminary processing which deals with the sensor noise and spurious data, the pixels are grouped in a manner that reflects the

objects in the scene with their spatial interrelationships. The top left corner of the figure presents the offline portion of the process, where the image of the objects that may be present in the scene are manipulated and the database of representations is constructed. Once the scene data is collected, the same techniques are applied online to obtain a representation of the objects that may be present in the scene as in the lower right corner of the figure. The lower right corner of the figure is the recognition portion of the system, including a spatial transformation which computes the pose of the object in the scene and the verification that indicate if the match between the scene and the model is correct.

3.3. Object recognition approaches

3.3.1. Geometry-based approaches

Geometric models of objects are adopted to deal with the appearance variation due to the viewpoint and illumination change. The geometric description of a 3D object allows the projected shape to be presented in a 2D image under projective projection, thereby facilitating recognition process using edge or boundary information which is invariant to certain illumination change. Additionally, much attention is made to extract geometric primitives such as line, circle and rectangle, which are invariant to viewpoint change. However, the research in [36] shows that such primitives can only be reliably extracted under limited conditions, for example lighting and occlusion.

Figure 8 depicts the experiment of Bong Seop Song, Kyoung Mu Lee and Sang Uk Lee on recognizing aerial images by using geometry based approaches [39]. The target model is provided with the characterized line features in Figure 8a. Four input test images with different viewpoints are given in Figure 8b. The line segments extracted from each input image and the matching results are shown in Figures 8c and 8d, respectively.

3.3.2. Appearance-based algorithms

Appearance-based techniques which are the most recent effort have advanced feature descriptors and pattern recognition algorithms. The eigenface method attracts much attention once it is one of the first face recognitions that are computationally efficient and relatively accurate. The eigenface approach has been adopted in recognizing generic objects across different viewpoints [37] and modeling illumination variation [38]. As the goal of object recognition is to distinguish one object from the others, discriminative classifiers can help exploit specific class information. Classifiers such as k nearest neighbor, neural networks with radial basis function (RBF), support vector machines (SVM), sparse network of Winnows (SNoW), and boosting algorithms have been applied to detect and label objects in real world from 2D images.

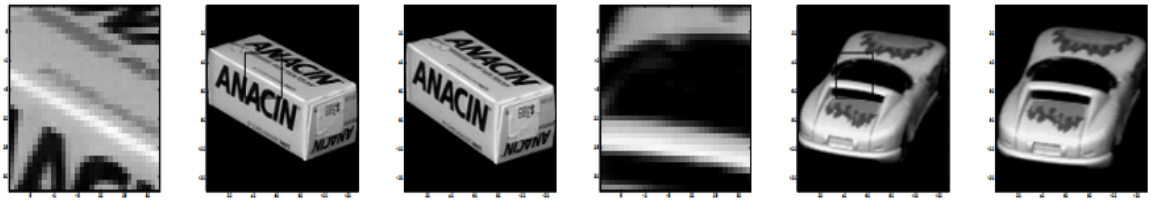


Figure 9: Object recognition and pose estimation using appearance-based approach and information sampling method.

While appearance-based methods present promising results in object recognition under viewpoint and illumination variations, they are less efficient in dealing with occlusion. Additionally, a large set of exemplars needs to be segmented from images for generative or discriminative methods to learn the appearance characteristics.

Niall Winters and Jos´e Santos-Victor perform an experiment on appearance based object recognition on a collection of 720 unknown objects and conclude that the correct recognition rate of the method is 95.3% and the correct pose estimation is 73.8% [40]. Figure 9 depicts the results obtained from using appearance based approach with information sampling method. Figures 9a and 9d depict the matching windows extracted from the unknown objects which are shown in figure 9b and 9e. Figures 9c and 9f show the closest match at the correct pose in training data.

3.4. Application of object recognition

Biometric recognition and handwriting document recognition are the most widely used applications. Face recognition has been studied extensively for decades and proved very applicable in crime investigation. Similarly, biometric recognition system based on iris or fingerprint has become feasible and reliable technology. Other object recognition applications such as content-based image retrieval, human computer interaction and industrial inspection are also popular today.

Chapter 4: Analysis and evaluation

In this chapter, we will analyse the strengths and weaknesses of the mobile application paradigms according to the viewpoint of a developers, mobile user and a service/content provider. Points from 1 to 5 are given for each criterion in which 1 is the lowest and 5 the highest. The points are given based on the information collected on the Internet, especially the different user and developer social communities. To ensure the objectivity of the evaluation a quite thorough investigation has been carried out.

4.1. Native application

4.1.1. Developer viewpoint

The biggest challenge for native mobile application developers is the platform fragmentation. We will perform the analysis and evaluation of native app paradigm on different mobile platforms (Android, iOS and Windows Mobile 7) from developer viewpoint.

- **Android**

Ease of developing

- Programming language

  Android allows development of managed code using Java-like language. The language follows the Java syntax but utilizes the libraries and the APIs provided by Google instead of the Java standard libraries and APIs. The language is as powerful as Java and reduces the workload to develop an Android app. Therefore, developers who have experience with Java or any other object oriented programming languages can easily create a native app on Android.

- Software Development Kit (SDK)

  The SDK is very usable, and easy to download and install. Android developers can find open source Android SDK free on Android Website and a series of understandable tutorials to install it. The SDK provides the developers with a selection of variety of Android platforms from Android 1.5 version and a set of core elements to build any type of applications from beginning: edit text, text view, image view, Web view and other complicated elements. Furthermore, the SDK comes with the functionality to debug and test applications that are currently under development, and includes an emulator to allow testing on your personal computer before installing the new application on the phone itself. With the support of multiple processes and component reuse, Android SDK provides support for intents and activities, which create a better user experience.

- Support from community

  Android has the benefit of being open source and Android Java is a simple programming language to learn. Therefore, there are large amounts of Android developers in mobile industry and **Android developers can receive full support**

from a worldwide community. Moreover, Android itself offers many valuable **tutorials, code samples and instructions** that are helpful for creating and deploying an app on Android.

Ease of developing	Points
- Programming language	5
- Software Development Kit (SDK)	5
- Support from community	5
Average points	5

Ease of coding

We can use other IDEs such as NetBean and JBuilder but Eclipse is always the best selection for developing Android apps.

Eclipse is a powerful IDE for coding and debugging an application on Android. Android has excellent Java development tools (JDT) which are standard with every installation of Eclipse. The best feature of JDT is the incremental compiler, which gives **immediate feedback with errors and warning** on the code. Furthermore, developers can also manage the application files very efficiently and speed up the process of coding and maintaining the code. For instances, files for graphical user interface are mainly stored in res folder, coded Java files in src folder and generated Java files in gen folder. It is also possible to **organize imports** in Java source files by simply pressing Ctrl-Shift-o, deal with bugs by adopting the **quick fix feature** of Eclipse, use **open type feature** by pressing Ctrl-Shift-T and learn the unfamiliar API by employing **content assistant**.

However, we found **Android UI Builder** inefficient to use. There are no instantly viewable and drag-and-drop capabilities provided by the UI builder. The layout files in the SDK cannot compensate for that. We have to create the UIs by coding on the layout.XML and value.XML files. Moreover, the code to access the device's hardware is longer and more complicated than HTML5's code.

Ease of coding	Points
Average points	3

Ease of debugging

Debugging tool in Eclipse works quite efficiently. It is easy to manage the breakpoints in the Java files and debug the code at runtime. The logcat is useful for finding the exceptions and bugs that make application crash. The debugging tool maybe complicated at first but with some experience, the tool is very useful to find and fix bugs. However, the tool runs slow and the complicated debug screen make it difficult to trace the value of Java objects.

Ease of debugging	Points
Average points	3

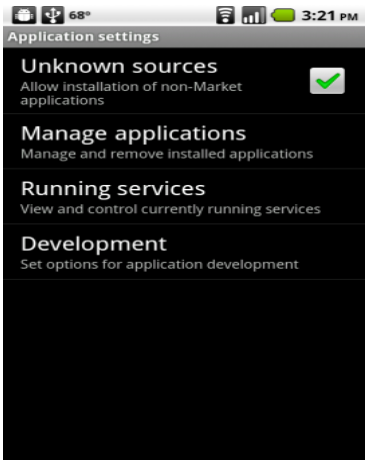


Figure 10: Deploying application on Android devices.

Ease of testing

- Test project

  Android offers a powerful but easy-to-use testing framework to create test cases, add unit tests and run the tests [24]. The result of the test includes the time for the test runs, the number of test runs, errors, failures and test method summary.

- Emulator

  The biggest drawback of the testing with Android SDK stays the **emulator**. We mainly use emulator for testing purpose but it is the heaviest component of Android SDK. Android emulator starts in more than 2 minutes, and run slowly on laptop with Intel core 2 duo 2.4 GHz and 2 Gigabyte RAM. Moreover, the default size of the emulator is too big and its size is unchangeable at runtime. Another problem related to the testing on both emulator and real device is that if a bug occurs, the application will stop immediately without giving any information. We can find out the exceptions on the logcat as discussed in Ease of debugging section.

- Real devices

  In addition to using the emulator, we can connect the device to the personal computer via the USB drive and try the application on the device flexibly.

Ease of testing	Points
- Test project	5
- Emulator	2
- Real devices	5
Average points	4

Ease of deploying and updating

Deploying an app onto Android phones cannot be easier. We can simply copy the .apk file onto our mobile phone and install it. The Android system requires that all deployed applications to be **digitally signed with a certificate** whose private key is kept by the developer. ADT plugin for Eclipse offers two signing modes: debug mode and release mode. In debugging mode, the signing process runs automatically under the compilation.

In release mode, we can use ADT Export Wizard to compile the app, generate private key and sign the apk.

In order to upgrade application to a new version seamlessly, developers have to sign the new version of the app with the same key and consider the validity period of the key. Android system treats the applications that are signed with the same key as a single application, allowing the developers to deploy and update their application in modules.

Ease of deploying and updating	Points
Average points	5

Ease of distributing

- Without app store

  One very attractive feature of Android platform is that it allows developers skip Android market completely and distribute application directly to the end users. This is very convenient in many situations, for example, a corporate IT department wants to distribute a private app to employees or a developer wants to test it before uploading to Android market. The developer just simply delivers the signed .apk file to target phones. In order to install the app on mobile phone, the end users should allow installation of non-Market applications by navigating to **Settings** then **Applications** and enabling the **Unknown Sources** as shown in Figure 10.

- With app store

  A traditional way to distribute an Android app is to publish it onto Android market. It is not difficult to publish the application and update on Android market. Developers must create a Google account and pay the registration fee with $25. The app published must meet the requirement enforced by market servers [25]. However, in order to use the license service to enforce licensing policies for paid apps on Android market, developers must follow many complicated steps as described in [26].

Ease of distributing	Points
- Without app store	5
- With app store	4
Average points	4.5

- **iOS**

Ease of developing

- Programming language

  The language for developing an application on iOS is objective C [60]. Objective C is not an easy programming language since it has many complicated and cumbersome rules relating to pointers, patterns and declarations. Therefore, the developer should have many experiences with C or C++ to know how to manage Objective C. Android developers are not necessarily professional in Java but iOS developers should have great programming skills in Objective C to work with iOS. However, many iOS developers consider **objective C** very appealing and impressive to motivate their programming.

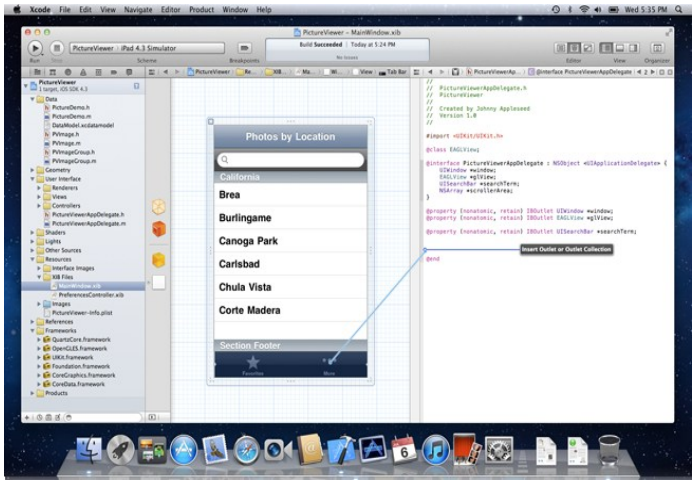


Figure 11: Xcode 4 user interface

- Software Development Kit (SDK)

  Developers can download iOS SDK from Apple sites. It costs developers $99 annually for the license of the SDK to publish their applications onto Apple app store and get revenue from selling the apps. iOS SDK is a powerful toolset to create great applications for Mac, iPhone and iPAD, including the Xcode IDE, iOS simulator, instrument analysis tool, Quartz Composer and the libraries for building an attractive and robust app [27].

- Support from community

  Like Android, iOS developers can receive a great benefit from a worldwide community of iOS developers and from Apple sites. They can access many videos, sample codes and tutorials provided by Apple, learning how to make a powerful app to create a seamless user experience.

Ease of developing	Points
- Programming language	3
- Software Development Kit (SDK)	5
- Support from community	5
Average points	4.33

Ease of coding

Xcode 4 (Figure 11) is a robust tool for coding. It has a set of powerful tools such as the UI builder, Apple LLVM compiler, and other supporting developer tools. Xcode 4 supports developers as much as Eclipse does. Xcode 4 is more effective than Eclipse in the way that it support drag-and-drop and instantly watchable capabilities. The advance features of Xcode 4 include the **Live Issues**, **LLVM** (low level virtual machine), **IDE Assistant** and **Version Editor**. Similar to Android, the code on iOS 4 to access hardware features (e.g. camera) is also verbose and complicated.

Ease of coding	Points
Average points	4

Ease of debugging

The debugging tool in iOS SDK is superior to that of Android, making it very easy to debug an iPhone app. Xcode 4 introduces LLDB, a brand new **debugging engine** contributed by Apple to the LLVM.org open source project [28]. The new debugging engine is built from the ground up to consume much less computing resources and become a rocket when it performs. It is also easy to create breakpoints, view the object's value by hovering the pointer over them, continue execution up to a particular code line, and step into, out of or over function or method call. Developer can also simply view the console log to analyse the code and track the bugs.

Ease of debugging	Points
Average points	5

Ease of testing

- Test project

  The instrument is one of the most appealing features of Xcode 4 to create a robust test for iPhone app. New data collection instruments are also available, including OpenGL ES (OpenGL ES for embedded system) for tracking iPhone graphics performance, new memory allocation monitoring that can find unintended memory growth, Time Profiler for collecting samples with very low overhead, and complete System Trace to find out how all system processes interact.

- Emulator

  iOS emulator is an advanced feature as compared to Android emulator. It is very quick to launch and performs as a real device does, enabling a perfect test bed to ensure that the user interface will work in the way we expect.

- Physical device

  In order to test the apps on a real device, developers must obtain an iPhone Development Certificate from the iPhone Developer Program Portal. They must follow many complicated steps to get the certificate and sign the apps before running the apps on the physical devices. The steps include signing up Apple developer account, paying $99 for membership, login to the provisional portal and installing WWDR intermediate certificate, creating development certificate, creating an app ID, finding and adding device unique identifier, creating and downloading and installing provisioning profile, and configuring build settings [34]. After these steps, developers can deploy, update and distribute the apps as discussed below.

Ease of testing	Points
- Test project	5
- Emulator	5
- Physical device	2
Average points	4

Ease of deploying and updating

After all steps described in **Physical device** section, iOS SDK enables developers to deploy their application to the iOS devices in some simple steps [32]: go to ActiveSDK and select version of iOS running on their device, and choose **build and run on opened**

project to run the app. Another way to deploy the application is to create .ipa file by clicking Build under Xcode menu and subsequently install the app via iTunes. Without the steps in Physical device section, the app will not run, hence iOS receives many complaints from the iOS developers for such difficulties.

Ease of deploying and updating	Points
Average points	3

Ease of distributing

- With app store

  Developers can distribute their apps via Apple app store and it is a very solid way for distribution. Developers can pick the price, get 70% of sale revenue and receive the sale payment monthly. Apple app store will validate the apps and check if it is eligible to publish the software on Apple store. The developers must pay $99 for the membership in App Store and download the iOS Software Develop Kit. After submitting the app, they must wait for the approval from the App store. While the apps are published on app store, Apple will manage the apps very effectively and end users can purchase the apps and download the update very simply. The app store will notify users when the update is available.

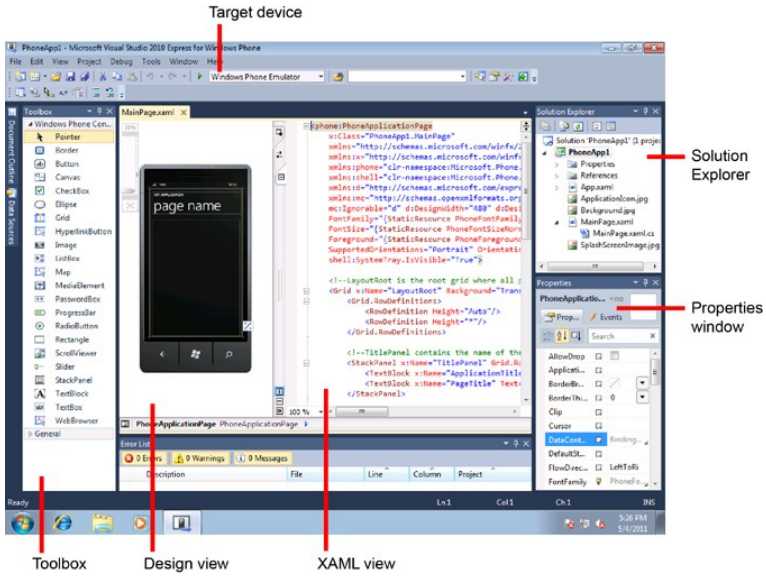


Figure 12: Visual studio 2010 Express

- Without app store

  Mobile developers can distribute their own applications using adhoc without going through the app store, but the developers will be limited to a maximum of 100 devices. The advantage of the method is that developers can distribute their apps from a Website, an email and so on. However, they still have to register and pay $99 a year for the registration fee.

  Mobile developers can also develop an app for internal distribution within a company, organization, government or educational institution. However, they have to pay $299 a year for registration.

Ease of distributing	Points
- With app store	4
- Without app store	3
Average points	3.5

- **Windows Phone 7**

Ease of developing

- Programming language

  The language for developing Windows Phone 7 is C#. Like Java, C# is very easy to learn, use and master. If the developers are familiar with Object Oriented Programming language (e.g. C++ and VB.NET) they do not face any difficulty in programming with C#.

- Software Development Kit (SDK)

  Windows Phone 7(WP7) SDK provides all the features Android SDK and iOS SDK support, or even more. The IDE for creating WP7 application is Visual Studio (VS) 2010 Express (Figure 12). VS 2010 includes features such as Windows Phone-based design surface, a code editor, Windows Phone project templates, and a **Toolbox** that contains Windows Phone controls. The other components of the SDK are **Windows Phone Emulator**, **XNA Game Studio**, **Expression Blend**, **samples** and **documentation**. The WP7 emulator is as excellent as the emulator of the iOS SDK, and much better than Android emulator. XNA framework provides the game developers with seamless support. Expression blend enables developers to simply create many rich Internet applications and XAML-based interfaces. However, one of the problems of WP7 SDK stays on the amount of money developers have to pay when they want to work with the SDK, which is much more expensive than iOS SDK. Another problem is the system requirements the developer must deal with when installing the WP7 SDK in their PC [29]. Such the problems limit the number of developers in WP7.

- Support from community

  Documentation, How-to Guides, sample code, and sample applications are provided to aid developers ramping up on Windows Phone development. **Forums, blogs, and Websites** are available for developers to ask questions and share information with the greater WP7 community. The new Visual Studio Help system will allow developers to continuously update their documentation sets. WP7 developers cannot **get supported from mobile app community as what Android and iOS get** because WP7 is quite new and only shares a small part of the market as compared to the two platforms.

Ease of developing	Points
- Programming language	5
- Software Development Kit (SDK)	4
- Support from community	4
Average points	4.33

Ease of coding

As discuss above, **C# language syntax** is very simple to handle with and the **IDE's UI** is simpler and more intuitive than eclipse and Xcode 4. It supports full features that the two IDEs have, for example coding aids and instant error feedback. The features help developers create, maintain and update the code effectively. **MSDN** is also helpful for coding because it offers developers the information about package, class, interface and method which help developers code faster and more appropriate. In addition, the UI designer is very powerful in the way it enables developers create a friendly and attractive user interface by simply dragging and dropping UI components from the Toolbox. WP7 SDK provides developers with many useful APIs (e.g camera and photo album APIs) thereby, enhancing the access to device's hardware. The code to interact with hardware on WP7 is less verbose and complicated than on Android and iOS.

Ease of coding	Points
Average points	5

Ease of debugging

It cannot be easier to debug the code with the IDE [30]. Developers can create a debug build while they are developing their application and a release build for testing purposes before deploying it to the physical device. The debugging process happens inside the code editor and the emulator can be launched and run during the debugging.

Ease of debugging	Points
Average points	5

Ease of testing

- Test project

  Developers can create WP7 test projects on Visual Studio Express to start unit testing. When developers create the test project they should use Silver Light Unit Test Framework for WP7 and WP7 essential testing.

- Emulator

  Testing on emulator is also easy because the emulator works perfect. The only thing the developers have to do is to select the release build in the debug tool.

- Physical device

  Unlike testing on emulator, testing Windows mobile app **on physical device** requires the developers to unlock the device which takes many complicated steps as in [31].

Ease of testing	Points
- Test project	5
- Emulator	5
- Physical device	2
Average points	4

Ease of deploying and updating

Deploying apps on Windows Phone is also described in [31], which is very complicated for mobile developers. However, **after finish all steps** described in physical device section, it is simple to deploy and update the apps on real devices.

Ease of deploying and updating	Points
Average points	2

Ease of distributing

- With app store

  The unlock process as discussed above creates a lot of difficulties for the distribution of mobile application on Microsoft marketplace. Similar to Android and iOS apps, WP7 apps must be approved by the marketplace but the difference is that the marketplace is the only place where developers can publish their apps and the end users can get the apps.

- Without app store

  Currently, it is impossible to distribute WP7 apps to end users without app store.

Ease of distributing	Points
- With app store	3
- Without app store	1
Average points	2

Application types

- Application using device capabilities

  In addition to the generous SDKs and the well-formed libraries available, the most exciting feature for developers in Android, iOS and WP7 is that we now have access to anything the operating system has access to, creating new and robust native apps. For example, if we want to create an application that dials the phone, we have access to the phone's dialer or if we want to create an app to utilize the phone's internal GPS, we are allowed to access it.

- Application using server capabilities

  Native apps take advantage of processing capability of server and/or storage of server. In the former case, native apps will send requests and wait for response from the server. Server processes the requests and replies the native apps with the result. An example of the capability is using the native apps which employ Google translate service in mobile platform. In the latter case, users keep their data on a server and pull it once necessary. For example, mobile cloud computing enables users to store data on the cloud and access it when they connect to the Internet.

- Application employing real time communication (e.g. gaming, banking, weather forecast and stock market):

  Native app is an excellent selection for gaming because it runs locally on device and access to all device capabilities (e.g. keypad, touch screen), providing mobile users with attractive image, sound and animations.

Native app can also enable mobile users to access banking, weather forecast and stock market service. Normally, service providers will build native apps to access their services and distribute the apps to end users. For example, Nordea bank creates a native app that runs on Android and iOS to serve customers with their banking services.

Application types	Points
- Application using device capabilities	5
- Application using server capabilities	5
- Application employing real time communication	5
Average points	5

Powerful APIs and libraries

Mobile platform SDKs (e.g Android SDK, iOS SDK and .NET Framework) provide developers with powerful APIs and libraries, which reduce the workload to build a native app.

Powerful APIs and libraries	Points
Average points	5

Payment possibilities

Developers can only publish their apps on app store and get revenue for the sale of the apps. For example, developers get 70% of sale revenue. It is most solid and efficient way to deliver native apps to end users.

Powerful APIs and libraries	Points
Average points	3

In the table below, we summarize the evaluation of native application paradigm on Android, iOS and Windows phone from the developer viewpoint.

Table 1: Summary of evaluation on native app paradigm from developer viewpoint

	Android	**iOS**	**Windows Phone**	**Average points**
Ease of developing	5	4.5	4.5	4.67
Ease of coding	3	4	5	4
Ease of debugging	3	5	5	4.33
Ease of testing	4	4	4	4
Ease of deploying and updating	5	3	2	3
Ease of distributing	4.5	3.5	2	3.33
Application types				5
Powerful libraries and APIs				5
Payment possibilities				3
Average points				*4.04*

4.1.2. User viewpoint

Native mobile apps share a big proportion of mobile app market. Zokem [22] shows that although Web apps are very popular on mobile platform, native apps are increasing their

share of face time relative to the Web application. The main reason for this, according to Dr. Hannu Verkasalo, founder of Zokem, is that mobile users learn to require more functionality on their own devices when mobile consumption gets richer and native apps can provide the best user experience. Flurry [23], a mobile analytic firm, presents in their report the fact that native applications are commanding more attention on mobile platform than Web applications.

In this section, we will analyse and evaluate mobile native app paradigm from user viewpoint. The criteria for the evaluation are **the ease of use, functionality, installation and update**.

Ease of use

- Performance

  The strongest points of native apps are the **excellent performance** and the **responsiveness.** Native apps run very smoothly on mobile devices and create very good user experience. Currently, mobile Web apps, widgets and HTML5 apps cannot compete with native apps in the race of performance and responsiveness.

- User interface

  Native apps have very attractive and intuitive user interface, including button, text view, label, list view and animation. It is the most advantageous feature of native apps to defeat the other mobile app types.

- Operation

  Usability is one of the strengths of native apps as compared to other mobile paradigms. Users can launch the apps by easily clicking the app's icon on home screen, close the apps and restart it without any effort. Additionally, a native app dedicated to a particular Web service provides one click to access the desired content. Mobile native apps can pull the data provided by remote server automatically and display the information on mobile screen. The feature makes social networking applications more usable and attractive.

Ease of use	Points
- Performance	5
- User interface	5
- Operation	5
Average points	5

Functionality

- Working offline

  All of native apps can work offline and save their data in local storage (external memory card or internal memory). For some applications using Web service (e.g. Google Map and IQEngine), native apps still need network access to make API request.

- Accessing device's hardware

As discussed in the previous section, native apps are **very functional** thanks to the capabilities of **using rich and interesting devices features** such as GPS, PIM contact, calendar information, camera and all of the new features that devices provide for the apps. Therefore, users can reach wide range of native apps, including the applications available on the devices and the apps installed manually by the users such as game, Geolocation, currency converter, photo maker and PDF reader.

- Using real-time communication

  Native apps are excellent choices for real-time gaming and other real-time mobile apps (e.g. banking, weather forecast and stock market information display).

Functionality	Points
- Working offline	3
- Access device's hardware	5
- Using real time communication	5
Average points	4.33

Installation and update

- Compatibility

  Device and platform compatibility is the biggest issues of native apps. For example, the app that runs on Android cannot run on iOS.

- Downloading and installing

  Users **must download and install** native apps on their devices manually. It requires the users to access the Internet and find out the sufficient installation package for the platform they are using. Fortunately, app stores will compensate for this shortcoming. Mobile users can search and download the apps directly from the app store of the platform providers (e.g. Apple, Android and Microsoft). Users will also receive the notification of new update from the app store when the update packages available.

Install and update	Points
- Compatibility	1
- Downloading and installing	3
Average points	2

The table below summarizes the evaluation for the native apps from the user viewpoint

Table 2: Summary of evaluation on native app paradigm from user viewpoint

Ease of use	5
Functionality	4.33
Installation and update	2
Average points	3.78

4.1.3. Service/ content provider

This section focuses on analysing and evaluating the effect of mobile native app on service/ content providers.

Content management

- Content presentation

  Native apps work independently with mobile browser. This can decrease the workload of content providers in presenting the content on mobile devices. They only have to format the data to make it fit the app's interface.

- Content delivering

  The advantage of delivering content to native app over mobile Web app is that service providers deliver only data to mobile devices rather than deliver both user interface and data. It will help decreasing the network overhead and saving many data delivery cost. It will help decreasing the network overhead and saving many data delivery cost.

Content management	Points
- Content presentation	4
- Content delivering	4
Average points	4

Administration

- Security

  Service providers are responsible for securing communication between mobile user and Web server. Because native apps are independent with mobile browser, service providers can implement some security schemes on native apps deployed on devices (e.g. encryption and authentication by a pass code) and therefore reduce their workload of server to perform the confidentiality and integrity on the data transferred over the Internet.

- Maintenance

  Service providers are responsible for the administration on the content delivered to native apps. They must guarantee that mobile users adopt the same version of one native app (e.g. the same functionalities and user interface) to access their service. Service providers must also make sure that the content they offer display intuitively on native apps.

Administration	Points
- Security	4
- Maintenance	3
Average points	3.5

Distribution

The biggest problem for service providers is the platform fragmentation of native apps. They should build and deploy different mobile apps running the same service on different platforms. For example, Nordea bank must provide their customers with the native apps running on both iPhone and Android. Another challenge is that the apps and update package must be downloaded and installed on mobile devices, and the service providers must notify every customer when the update is available. Therefore, the service providers

publish their apps on app stores to distribute the apps to their customers, which require a lot of overhead to do.

Distribution	Points
Average points	2

The table below summarizes the evaluation for the native apps from the service/content provider viewpoint

Table 3: Summary of evaluation on native app paradigm from service/content provider viewpoint

Content management	4
Administration	3.5
Distribution	2
Average points	3.17

The table below summarizes **the evaluation for the native apps** from the developer, user and service provider viewpoint

Table 4: Summary of evaluation on native app paradigm

Developer	User	Service/content provider
4.04	3.78	3.17

4.2. Mobile Web apps

4.2.1. Developer viewpoint

In mobile Web app paradigm, developers will have to focus on the server tasks, such as building the apps, deploy and updating the apps on Web server, and distributing the apps to end users.

Ease of developing

- Programming language

  To create a mobile Web app, developers should have experience and programming skill at both client-side (HTML, CSS, JavaScript, XML and SVG) and server-side (e.g. ASP.NET, JSP, Python, Ruby and PHP). Learning and mastering the technologies to create a mobile Web app requires many efforts of developers.

- Software Development Kits (SDK)

  Similar to Web app, there are many SDKs, APIs and libraries to build a mobile Web app, such as .NET framework, Dreamweaver and Java EE. Developer can choose the most suitable SDK to develop mobile Web app.

- Support from community

  Mobile Web app no longer get attention of mobile developers. In fact, developers still receive the support from community but the support is not as strong as the support that native app developers obtain from their community.

Ease of developing	Points
- Programming language	3

- SDK	5
- Support from community	3
Average points	3.67

Ease of coding

Developers can use PHP, JSP and ASP.NET for server side code, and HTML, JavaScript and CSS for client side presentation. The IDE for coding include Dreamweaver, Visual Studio .NET, JetBrain and NetBean. The ease of coding depends on the Web programming language and the IDE that the developers select. For example, Visual Studio .NET, Dreamweaver and Aptana Studio are the most efficient IDE for mobile Web application development with many supports for developers. The biggest disadvantage of the paradigm is that mobile Web app cannot make mobile Web app hook into device's hardware. It is absolutely a minus point for the paradigms. Moreover, mobile Web app is cross platform but browsers on different platforms may support different components of a Web page. Therefore, in order for mobile users to view the whole page, developers should have a significant amount of information about the device and mobile browser in use. Otherwise, they should design a mobile Web page as simple as possible so that all mobile browsers can display on screen.

Ease of coding	Points
Average points	2

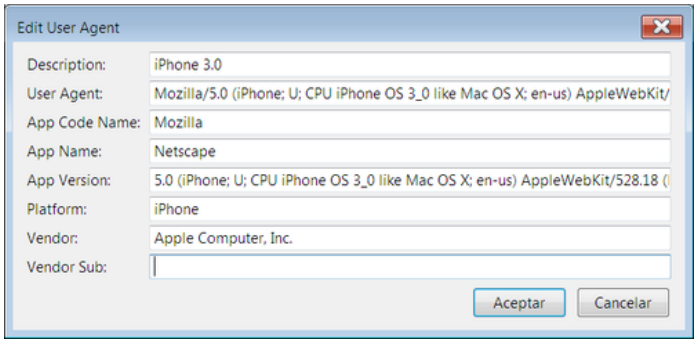


Figure 13: Using User Agent Switcher to debug mobile Web apps with any user agent

Ease of debugging

Debugging mobile Web app is **complicated** as compared to native app debugging. It includes many steps such **as server side debugging, markup debugging and client side debugging**, which requires the capability to view the source code, debug JavaScript or execute JavaScript command from a console in each step.

- Mobile developers can adopt HTTP tools such as User Agent Switcher to change the user-agent string that Firefox uses for making HTTP requests to the server (Figure 13). They can then browse to any Website and see how the server manages the user agent and which content it serves.
- Developers can also adopt W3C mobile checker and ready.mobi for markup debugging.
- **Debugging JavaScript is the most difficult activity in mobile Web development** because sometime script that works on a device does not work on

the other devices. Typically, developers must check their JavaScript on desktop platform before debugging on mobile platform. However, the code that works on desktop platform may not work on mobile browser. Developers should hence employ the console logging feature on mobile platform such as Safari Debug Console and Android Debug Bridge of Android SDK. Furthermore, developers can employ many debugging tools such as SocketBug, Opera Dragonfly, jsConsole and Weirne. Using the IDEs as discussed in the previous criterion (Aptana, Visual Studio and DreamWeaver) for debugging purpose is also a good choice because the IDEs always have debugging tool.

Ease of debugging	Points
Average points	3

Ease of testing

- Test project

  Most of the current IDEs do not offer the ability to create test project.

- Test on emulators

  Device emulator is obviously valuable for viewing how our Website can render on the real device. If mobile Web app works appropriately on emulator, it cannot be sure that the app will work on a real device. Moreover, several platforms that do not offer emulator for testing mobile Web apps limit the opportunity to test the new apps.

- Test on the real device

  The testing on physical device should be mandatory because there are many problems with the test solution on emulator. However, mobile device diversity creates many difficulties in testing a mobile Web app. There are many differences (e.g. screen size, color, resolution) between real devices and hundreds of unexpected bugs may happen on the devices. For example, when we encounter a JavaScript error, many devices do not show any notification, making it difficult to detect the problem. Another example is that mobile Web apps may display properly on this device but do not work on the other devices. These make complicated to perform the test on mobile Web app and require the developers to test their apps on as many real devices as possible.

Ease of testing	Points
- Test project	1
- Test on emulator	2
- Test on real device	2
Average points	1.67

Ease of deploying and updating

Like any Web app, developers deploy their own mobile Web app on a server and the app can be accessed from everywhere. Developers gain two benefits from mobile Web app paradigm. Firstly, they do not have to publish the apps on any app store and wait for the app store's approval. Secondly, developers earn most of revenue from selling the apps and only pad for the hosting fee. Developers will simply update their apps on Web server instead of pushing the update packages to all mobile devices as native app developers do.

However, developers must be responsible for all steps in driving the apps to the end users, for example searching for the hosting service, publishing the apps on the Web server, maintaining the apps and authorizing the access to the apps.

Ease of deploying and updating	Points
Average points	3

Ease of distributing

As soon as developers publish their mobile Web apps on a server, end users can access the apps from everywhere by typing the app's URI on browser. Therefore, the developer can distribute their applications easily without any cost beyond the server fee. Additionally, the platform fragmentation does not cause any problem when delivering the app to mobile devices.

Ease of distributing	Points
Average points	5

Application types

- Application using device capabilities

  Mobile Web apps are unable to access device's hardware. Therefore, mobile Web app paradigm is not an intelligent selection when developers want the apps to take advantage of the device capabilities.

- Application using server capabilities

  All mobile Web apps use network content, processing capability and storage of Web server.

- Application employing real time communication (e.g. gaming, banking, weather forecast and stock market)

  Mobile Web app is not suitable for gaming because gaming requires heavy graphic renders and instant reactions, but mobile Web app is low responsive and device capabilities are limited. However, mobile Web app can provide the end user with banking, weather forecast and stock market service. Normally, a bank will create a mobile version of the service beyond the full version of the service. For example, Nordea bank have mobile version for their Web app with a simpler user interface. Users will get the newest data as soon as they access the app.

Application types	Points
- Application using device capabilities	1
- Application using server capabilities	5
- Application employing real time communication	3
Average points	3

Powerful APIs and libraries

Many powerful APIs and libraries ease the development of mobile Web apps as discussed above. However, developers cannot find any API and library that help mobile Web apps hook into device features (Camera, GPS and accelerator).

Powerful APIs and libraries	Points
Average points	3

Payment possibilities

Developers can earn revenue from publishing their apps on Facebook as in [33]. They can also distribute their apps to the app store and earn money per subscription of the apps. Another way to earn revenue from selling the app is to adopt **pay wall model** by blocking the access to the apps by a form requiring payment before using them.

Payment possibilities	Points
Average points	5

The table below summarizes **the evaluation for the mobile Web apps** from the developer viewpoint

Table 5: Summary of evaluation on mobile Web app paradigm from developer viewpoint

Ease of developing	3.67
Ease of coding	2
Ease of debugging	3
Ease of testing	1.67
Ease of deploying and updating	3
Ease of distributing	5
Application types	3
Powerful APIs and libraries	3
Payment possibilities	5
Average points	*3.26*

4.2.2. User viewpoint

In this section, we will analyse and evaluate mobile Web app paradigm on mobile device from user viewpoint. The criteria using for the analysis and evaluation are **ease of use, functionality, and installation and update**.

Ease of use

- Performance

  Time for loading is the biggest problem of mobile Web apps due to the limited network bandwidth and processing capabilities of the devices (e.g. rendering heavy image). Mobile Web apps hence always lag and have very low performance.

- User interface

  One of the drawbacks of a mobile Web app is that its user interface is completely dependent on the mobile browsers. Scrolling, animation, transition, form and other native app specific features are not possible or are not as attractive and seamless for users. Furthermore, user interface will be loaded each time the devices access the app, increasing the network overhead and making it less responsive.

- Operation

Mobile users have to open mobile browser and input URI to access mobile Web apps. Moreover, it is **very difficult** to enter the URI because of the small screen size and the unusable keypad of the devices. Moreover, moving back and forward between pages is also very difficult and slow.

Ease of use	Points
- Performance	2
- User interface	1
- Operation	1
Average points	1.33

Functionality

- Working offline

  Mobile Web apps cannot work offline

- Accessing device's hardware

  Mobile Web apps cannot access device's capabilities. This reduces the applicability and attractiveness of mobile Web apps.

- Using real-time communication

  Mobile Web app is a bad choice for real-time gaming and other games that require responsiveness and animation. Furthermore, by using mobile browser, device cannot automatically pull new data from a Web server without accessing the app on the server. It makes the mobile Web app associated with social networking less efficient and usable for mobile users. However, mobile Web app can work well with banking, weather forecast and stock market information display.

Functionality	Points
- Working offline	1
- Access device's hardware	1
- Using real time communication	3
Average points	1.67

Installation and update

- Compatibility

  Some components of a Web app cannot work on every mobile platform. For example, flash component works on Android but on iOS. However, generally, mobile Web apps are compatible for most of mobile devices. They can run on different platforms on different devices.

- Downloading and installing

  Users do not have to download and install mobile Web apps on their devices but they can always approach the newest version of the apps and the newest content provided by the apps. Service provider will deploy the apps on Web server and distribute their apps to mobile users via Web browser.

Installation and update	Points

- Compatibility	4
- Downloading and installing	5
Average points	4.5

The table below summarizes **the evaluation for the mobile Web apps** from the user viewpoint

Table 6: Summary of evaluation on mobile Web app paradigm from user viewpoint

Ease of use	1.33
Functionality	1.67
Installation and update	4.5
Average points	2.5

4.2.3. Service/ content provider viewpoint

Content management

- Content presentation

  Accessing Web apps on mobile device often results in poor or unusable experience due to the limitations of the devices, such as small size display screen, network bandwidth, keypad, battery and processing capability. It hence creates **many problems** for service providers to drive their Web apps to the end users. The requirement for the service provider hence is much harder than that for normal Web app providers. These include **capability exploit**, **navigation and link**, **page layout** and **content**, **page definition** and **user input**.

  Service providers try to exploit the capabilities of mobile device because the more capabilities the apps exploit the better user experience the apps create. Unfortunately, it is impossible to build a mobile Web app with the aims to access device hardware.

  Because of the limitations in device screen and input mechanism [65], the URI to the app should be as short as possible. Keeping the URI short will reduce the possibility to enter it wrong, providing more satisfactory user experience. The navigation on the page should be solid and simple, making easy to navigate forth and back on the pages of the Web app.

  They should **also eliminate all the images** in the page content or **resize the image** on the server as much as possible to reduce the amount of data transferred and the amount of processing the device has to perform to render the image. Moreover, service providers must ensure that the page will fit the device's screen or limit the scrolling into one direction (usually vertical direction).

  Service providers must provide a short descriptive title to allow easy identification but keep the title short to reduce the page's size. They should not use table, frame and other components that can make the page more complicated. The text on the page should be readable for the end users.

  Because the input on mobile device is difficult, the Web apps must avoid the free text input. Content providers ensure that the interface must be as far as possible minimize the user input. Where possible, the page should use selection list, radio button and other controls that do not require typing.

- Content delivering

  Mobile users always look for specific pieces of information rather than browsing all of them. Therefore, service providers must limit the content to what user has requested to decrease network overhead. Moreover, content providers must deliver the whole pages, including presentation (user interface) and data to mobile users when they first time access the mobile Web apps.

Content management	Points
- Content presentation	2
- Content delivering	3
Average points	2.5

Administration

- Security

  Service providers are responsible for authenticating the users who want to access mobile Web apps on server. In several sophisticated system that require extremely high security (e.g. banking system or e-commerce), service providers must protect the integrity of sensitive data exchanged between mobile users and Web server by using SSL (Secure Socket Layer) and other security approaches.

- Maintenance

  Service providers must host, update and manage the versions of mobile Web apps. An advantage of maintaining mobile Web apps is that service providers only update their apps on server. Mobile user will reach the new app version automatically when they access to the app.

Administration	Points
- Security	3
- Maintenance	4
Average points	3.5

Distribution

Service provider will deploy their Web application on a server and update the app manually at server side. Customer can subsequently access the newest version of Web app via mobile browser from everywhere. It is hence superior to native app paradigms whereas the service providers must build different apps to serve the same service on different platforms and send the update to every mobile device.

Distribution	Points
Average points	5

The table below summarizes **the evaluation for the mobile Web app** from the service/ content provider viewpoint.

Table 7: Summary of evaluation on mobile Web app paradigm from service/content provider viewpoint

Content management	2.5
Administration	3.5
Distribution	5

Average points	*3.67*

The table below summarizes **the evaluation for the mobile Web app** from the developer, user and service provider viewpoint

Table 8: Summary of evaluation on mobile Web app paradigm

Developer	User	Service/content provider
3.37	2.5	3.67

4.3. Mobile widgets

4.3.1. Developer viewpoint

Ease of developing

- Programming language

  As discussed in chapter 3, developers create mobile widgets by using Web technologies, such as HTML, JavaScript, CSS, XML, SVG and Ajax. These are well-known standards amongst a large Web developer community. The technologies are simple to learn and use, and hence mobile widgets are much easier and faster to develop than native applications.

  JavaScript.

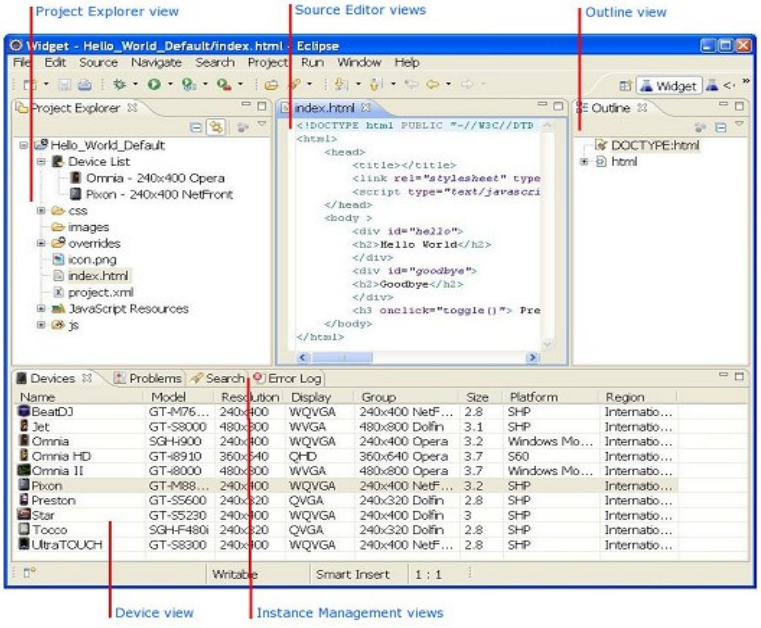


Figure 14: Coding mobile widgets by using Eclipse and Samsung TouchWiz SDK.

- Software Development Kit (SDK)

  There are many SDKs available for creating mobile widgets: Samsung TouchWiz, Opera mobile widget and LG mobile widget SDKs. Developers can download the SDKs free and subsequently install them on their development machine. Developers then integrate the SDKs into third party IDEs. For example, Samsung TouchWiz SDK implementations will be integrated into Eclipse as a series of Eclipse plugin.

The SDKs include several tools such as DOM inspector, JavaScript debugger, network inspector, resource inspector and other utilities to build attractive and cross device mobile widgets. The SDKs are lightweight, applicable and easy to use for most of mobile widget developers. Developers can learn how to use the functionalities of the SDK from several documents accompanied with the SDKs. The SDKs can also provide developers with the emulator, which is very valuable for debugging and testing Support from community

Mobile widgets developers receive many supports from a wide community of programmers. In addition, the developers can download several tutorials from the many Websites for developers such as Opera, programming 4us, W3C and so on.

Ease of developing	Points
- Programming language	5
- SDK	5
- Support from community	5
Average points	5

Ease of coding

The Web technologies (e.g. CSS, HTML and JavaScript) do not require any complicated IDEs. However, in order to obtain the full support and speed up the coding process, developers can select one of many powerful IDEs available for building mobile widgets, such as Eclipse, Aptana Studio, OpenKomodo, UltraEdit and Emacs. All of the IDEs, which have intuitive user interface, support mobile widget developers seamlessly by providing the code assist, context menu, auto complete, integrated debugger, error feedback and other IDE specific supports. The IDEs can ease the development of mobile widgets by offering developers with the seamless capabilities such as creating and configuring the new project, managing the files (CSS, HTML, JavaScript files and other resources) precisely, selecting the available templates for the project and import existing widgets. Figure 14 depicts using Eclipse for coding mobile widgets.

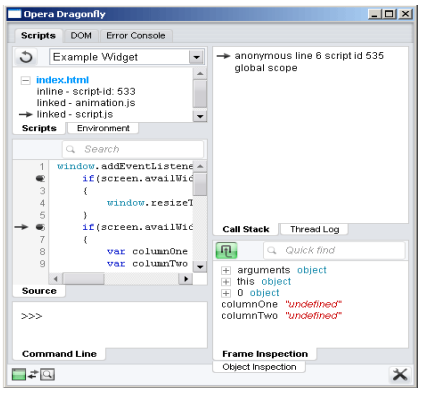


Figure 15: Debugging mobile widget by using Opera Dragonfly

However, the fragmentation in mobile platforms and devices still create many challenges for mobile widget developers. Different devices have different display capabilities (e.g. screen size and resolution) and widget engines. In order to create a cross device mobile widget, developers should make the app's user interface adapt to different circumstances by using media queries and media rules in CSS. The media queries will enable the app to

detect the screen capabilities of the device and create the media rules accordingly. The media rules include the screen size (width and height), resolution property and the availability of colours. Furthermore, to adapt mobile widgets to different devices more easily, developers should separate the apps into several small parts using MVC pattern to make data separate from view. In addition, they should ensure the responsiveness of widgets to improve user experience by optimizing the code as much as possible. The APIs provided by OEMs allow mobile widgets to access device's hardware but when we use the APIs, the widget is no longer cross-platform.

Ease of coding	Points
Average points	3

Ease of debugging

Debugging is one of the most difficult steps in the development of mobile widgets due to the interoperability issues of JavaScript handling. Developers can select among good debugging tools when they develop their mobile widgets. The IDEs (Visual Studio .NET, Eclipse, Aptana Studio and OpenKomodo) discussed above should themselves have debugging tools which ease the debugging steps. Additionally, developers can simply launch the plugins in browser for debugging purposes (e.g. browsing the DOM, examining source code and creating breakpoint). The plugins include firebug in Mozilla's Firefox browser, Opera Dragonfly and Safari Web inspector. Furthermore, Dragonfly enables widget developers to perform the remote debugging, which is useful in case the developers are debugging the widgets installed a mobile device from a computer. Figure 15 presents how to use Dragonfly to debug a mobile widget.

Ease of debugging	Points
Average points	5

Ease of testing

- Testing project

  There are very few IDEs or debugging plugins having the capability to create a test project for testing Widget.

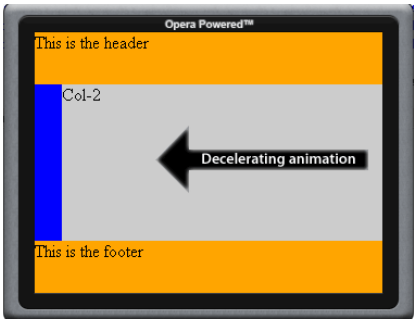


Figure 16: Opera emulator

- Test on emulators

  Instead of copying the widgets onto the device each time they have a change, many SDKs have emulator for testing purpose. For example, Opera widget SDK provides developers with Opera Widget Mobile Emulator. The debugging tool and the widget

emulator in the SDKs combined give developer extreme flexibility in developing applications for various devices. The debugging tool (e.g. Opera Dragon fly) will provide drop down menu options for the widget emulator and developers can select one of them for testing. The widget emulator works effectively and developers can run the debugging tools and the emulator simultaneously to trace the activities of the widget on emulator. Figure 16 presents the display of a widget on Opera emulator.

- Test on physical device

  Beyond testing on the emulator, developers can easily make the final test to ensure that the widgets work appropriately on the real device. Developers can deliver the packaged widgets to their device easily and test the widgets from device. Due to the fragmentation on device and platform, mobile widgets should be tested on as many devices as possible to guarantee that they work on the real mobile devices.

Ease of testing	Points
- Test project	1
- Test on emulator	5
- Test on real device	3
Average points	3

Ease of deploying and updating

By using many IDEs, developers can easily deploy mobile widget onto mobile devices. The steps to deploy the widget are as follows. Firstly, developers create the standard package .zip file including Web pages, icon graphics files, style sheets, JavaScript files, and other resources. Then, they copy the package file to the devices and install/run it on the device.

It cannot be expected that a mobile widget, once created can run on different widget engines without any modification. Most probably, the widget engines also employ different means to wrap the widget application in a deployable package. For example, Samsung Touch Wiz and Windows Mobile will create different file extensions of the zip package. However, it is .widget on Samsung TouchWiz but .wgt on Windows Mobile. The ways the widgets installed on different mobile platform are also different because the widget engines distinguish between different platforms.

Ease of deploying and updating	Points
Average points	3

Ease of distributing

- With app store

  One of the benefits that mobile widgets offer their developers is the low time to market. Carriers, original equipment manufacturers (OEMs) or independent widget vendors host the app stores or marketplaces. For example, developers can distribute their mobile widgets to mobile users via Android marketplace, Samsung app store, Apple dashboard and Opera widgets repository. The distribution and monetization infrastructure in mobile widget platform provides a mean for developers to get their widgets to market and provides an opportunity for user to discover new widgets and content. An example of the efficient distribution is publishing the widgets onto Opera

widget repository. Each widget published will be reviewed by Opera Software to check if it works on different devices. Opera will filter the list of available widgets based on the mobile devices viewing the page, thereby enabling users to access the widgets that are executable on their type of device.

- Without app store

  Developers can publish their mobile widgets on their own Website and users can access the Website to download and run the widgets on mobile devices. However, the developers must by themselves take into account all of the fragmentation problems of the mobile widgets and test the widgets on as many devices as possible to guarantee that the widgets will run on different devices.

Ease of distributing	Points
- Without app store	5
- With app store	3
Average points	4

Application types

- Application using device capabilities

  Mobile widgets can access device hardware (e.g. phone book, camera and GPS) by using the APIs and libraries provided by carriers, device OEMs and third party vendors. This makes mobile widgets much better than any mobile Web app because there is no mobile Web app can do such things.

- Application using server capabilities

  Mobile widgets can access mobile Internet and network services. They adopt the processing capabilities and storage on server in an indirect way via Web API provided by the service provider. For example, mobile widgets for Google translate makes use the processing capabilities of Google to run the translation on Google server and obtain the result after the translation is comspleted.

- Application employs real time communication (Gaming, banking, weather forecast and stock market)

  Developers can only create the simple and casual game on mobile widgets, such as Sudoku, snake and Tic-Tac-Toe. Mobile widget is not a good choice for 2D and 3D gaming associated with real time communication because the games do require a complicated infrastructure and processing, but the widget's architecture is simple and the performance of widgets on mobile device is limited (e.g. video and animation).

  Mobile widgets normally provide a narrow range of functionality for a single context. They are hence not suitable for driving banking services to mobile user, which requires a robust and functional application on mobile agents.

  Mobile widgets work efficiently to display the weather forecast and stock market information on the real time because mobile can connect to the Internet, adopt Web services and communicate with service providers via Ajax.

Application types	Points

- Application using device capabilities	4
- Application using server capabilities	4
- Application employing real time communication	3
Average points	3.67

Powerful APIs and libraries

Carriers, device OEMs and third party vendors provide mobile developers with mobile widget platform that includes very powerful APIs, integration tools and libraries. Some mobile widget platforms are Samsung TouchWiz, Blackberry, OperaWidget, PhoneGap, Windows Mobile, Yahoo! Blueprint and NokiaWRT.

Powerful APIs and libraries	Points
Average points	5

Payment possibility

Developers can publish their mobile widgets on marketplaces or app stores, and earn the revenue for each download from the mobile user. It is the only way to get revenue from selling mobile widget.

Payment possibility	Points
Average points	3

The table below summarizes **the evaluation for the mobile widgets** from developer viewpoint

Table 9: Summary of evaluation on mobile widget paradigm from developer viewpoint

Ease of developing	5
Ease of coding	3
Ease of debugging	5
Ease of testing	3
Ease of deploying and updating	3
Ease of distributing	4.5
Application types	3.67
Powerful APIs and libraries	5
Payment possibility	3
Average points	*3.91*

4.3.2. User viewpoint

Ease of use

- Performance

  Mobile widgets require less time for loading than Web apps because widgets are pre-installed on mobile devices. Therefore, m**obile widgets run faster than mobile Web app but cannot perform as good as native apps and HTML5 mobile apps**. Moreover, widget is only simple and single functional app that provides mobile users a single purpose service, such as showing the latest news, the current weather, the time and a dictionary.

- User interface

  Mobile widgets have very simple user interface. They are also **less responsive** than native apps and do not have much animation, video and audio handling. However, users are capable of customizing widgets when they install the widgets on their devices.

- Operation

  Unlike mobile Web app, users can **directly** launch widgets on the home screen, gaining immediate access to Web content without having to open a browser and enter URI. Furthermore, mobile widgets improve the Web experience by providing Web data through the widgets. Mobile users can access the content of a site very easily via widget because the widget needs very little data transmissions to get live data from the Web.

Ease of use	Points
- Performance	3
- User interface	3
- Operation	5
Average points	3.67

Functionality

- Working offline

  Many mobile widgets (e.g. clock, calculator, currency converter and dictionary) can work offline and have local storage to save data temporarily for offline use. However, the widgets that serve the newest data (e.g. weather forecast and stock market display) need Internet access to work appropriately.

- Accessing device's hardware

  Mobile widgets can access a device's hardware and platform-level APIs, and are more valuable to the mobile users than mobile Web apps. For example, a widget can be used for capturing a picture and storing it to flickr account. The widget starts the camera, take the picture, add location tags automatically by using the GPS and asynchronously upload the picture to flickr.

- Using real-time communication

  Mobile widget is a portable and simple app providing low range of functionalities. It is hence not suitable for driving the services that have complicated capabilities (gaming and banking). However, mobile widgets can work well to show weather forecast and stock market information on real-time.

Functionality	Points
- Working offline	3
- Access device's hardware	5
- Using real time communication	2
Average points	3.33

Installation and update

- Compatibility

  The incompatibility of mobile widgets exists in different levels: widget engine (or browser), platform and device. The widget that runs on a widget engine cannot run on the other engines. For example, calculator widget working on Yahoo! Widgets cannot work on Opera widget engine. In addition to widget engine incompatibility, the incompatibility on platform and device is still a big concern even though Opera, Samsung, LG and Yahoo! have several solutions based on W3C standards to create a widget that runs on multi platforms and devices.

- Downloading and installing

  Mobile widgets must be **downloaded and installed** on mobile devices. Users must firstly install the mobile widget runtime before installing the widget. Some widget engines are cross platforms and devices (e.g. Opera widget engine and SurfKitchen) but some are fragmented (e.g. Myriad and Nokia WRT). For instances, Opera runtime integrated in Opera mini browser should be installed before any Opera widget is installed on mobile devices. On some mobile devices, widget engine is pre-installed and users do not have to download and install it again. After mobile widget engine is available on devices, users can find and download their favourite mobile widgets from app stores. The app stores will filter and display the widgets based on the mobile devices connecting to the app store. It also notify the users when update for the widgets is available, making it easy and convenient to download and update the widget onto the devices.

Install and update	Points
- Compatibility	3
- Downloading and installing	3
Average points	3

The table below summarizes **the evaluation for the mobile widgets** from user viewpoint

Table 10: Summary of evaluation on mobile widget paradigm from user viewpoint

Ease of use	3.33
Functionality	3.33
Installation and update	3
Average points	3.22

4.3.3. Service/ content provider viewpoint

Content management

- Content presentation

  The presentation of content is pre-defined by mobile widgets on the devices. Content providers only have to format the data transferred to the devices so that its display can fit the layout of mobile widgets.

- Content delivering

  Currently, implementing the mobile versions of many Web services is difficult. The support for Web standard based widgets is a huge step forward to reach the Web

services successfully on mobile devices. When mobile widgets access the service, content providers deliver only data instead of sending both user interface and data to mobile device. It will help release the content provider's workload and decrease network traffic overhead.

Content management	Points
- Content presentation	4
- Content delivering	4
Average points	4

Administration

- Security

Since mobile widgets are single and simple purpose applications, they do not require much security in communication between server and mobile users. In very few sensitive cases, the responsibility of authenticating user still stays on Web server because mobile widget does not support security schemes.

- Maintenance

Service providers are responsible for maintaining content on server, ensuring that mobile users will receive the most updated content on their devices. Moreover, they must guarantee those mobile users who access their content have the same version of mobile widgets and the content delivered to the users must fit the layout of the widgets.

Administration	Points
- Security	4
- Maintenance	3
Average points	3.5

- Distribution

Service providers can set up their own APIs to make their Web services accessible for the mobile widgets built by a third party developer (e.g. Yahoo! Weather). They may also develop their own mobile widgets and distribute to their user. In the latter case, service providers should take platform and device fragmentation into account otherwise a widget work on a device but cannot run on other ones.

Distribution	Points
Average points	3

The table below summarizes **the evaluation for the mobile widgets** from service/content provider viewpoint

Table 11: Summary of evaluation on mobile widget paradigm from service/content provider viewpoint

Content management	4
Administration	3.5
Distribution	3
Average points	*3.5*

The table below summarizes **the evaluation for the mobile widgets** from developer, user and service provider viewpoint

Table 12: Summary of evaluation on mobile widget paradigm

Developer	User	Service/content provider
3.91	3.22	3.5

4.4. HTML5 mobile app

4.4.1. Developer viewpoint

Ease of developing

- Programming language

  Similar to mobile widget developers, HTML5 mobile developers do not need to have any programming skills in Objective C, C# or Java to build a simple mobile Web app. Instead, they only have to be familiar with HTML5, CSS3, JavaScript (JQuery Mobile) and other related Web technologies. Because the technologies are not complicated to master, skilled Web app developers can easily create many **simple** HTML5 mobile Web apps. However, developers should know one of many server side-scripting languages (e.g. PHP, ASP.NET, Python and Perl) in order to build **large and complicated** Web applications, such as banking and ecommerce. The advanced features of HTML5, CSS3 and jQuery Mobile (discussed later in Ease of Coding section) can significantly improve the client-side presentation and hence make HTML5 mobile Web apps superior to traditional mobile Web app and widgets.

- Software Development Kit (SDK)

  Developers can use preferred choices of authoring tools, rather than specific platform SDKs. There are several developer frameworks freely available that give developers the environment to build mobile Web apps quickly and easily using HTML5, CSS3 and JQuery Mobile. These provide developers with powerful features to code, to test and deploy HTML5 mobile apps, and to replicate the features of native app UIs, such as native Webkit animations for pages and buttons, and recognition of touch interactions on modern smart phones, such as swiping, taping, pinching and rotating. The frameworks include SenchaTouch, jQTouch, JQuery Mobile, Titanium Mobile and Aptana. Moreover, developers can choose among commercial frameworks such as .NET framework and Deamweaver to develop their mobile Web apps.

- Support from community

  HTML5 mobile developers can receive many supports from a worldwide community of Web apps developers, even more than native app developers do. Furthermore, the developers can access many tutorials and documents available on the SDK's Website. MSDN, Google Developer, Dev Opera and W3C are the most famous Websites that provide many tutorials, sample codes and documents for HTML mobile app developers.

Ease of developing	Points
- Programming language	4

- SDK	5
- Support from community	5
Average points	4.67

Ease of coding

The IDEs such as Aptana Studio, Dreamweaver, NetBeans and Visual Studio 10, which are very valuable for client side coding (e.g. HTML5, CSS3, JavaScript and other related technologies) and server side coding (e.g. PHP, ASP.NET and Python). The code editor provides mobile developers with many powerful features such as **code assist, instant code checking and error feedback**, helping developers easily indentify, and fix bugs and issues. In addition to the code editor, most of the IDEs equip mobile developers with integrated debugging tools, emulator for testing and deployment wizard to deploy the Web apps. Another benefit for mobile developers lie in the **jQuery Mobile**, which is a lightweight, cross browser JavaScript library that simplifies the client-side scripting of HTML5, event handling, animation and Ajax on mobile phones. The new library helps remove the cumbersome that developers have when dealing with traditional JavaScript. In the other words, jQuery Mobile is designed to change the way we write JavaScript, which is very complicated. Furthermore, CSS3 is the latest standard of CSS, which allows for greater control over typography, animations and styling touches such as shadowing, gradients and rounding off box corners. Developers can shorten the time for their coding by employing the capabilities that HTML5 provides as discussed in chapter 2. For example, with HTML5's introduction of the new <video> element, developers can now include video within their pages without the need for embedding it in a plugin like Flash. It hence remove the headache of iOS developers since currently iOS does not support for Flash.

Esiness of coding	Points
Average points	5

Ease of debugging

The problem of mobile Web apps is that developers have to debug both client-side and server-side code, which requires **many efforts**. However, most of the IDEs discussed above (e.g. Visual Studio 2010, Aptana and DreamWeaver) include debugging tools that simplify debugging process at both client-side (HTML5, CSS3 and jQuery Mobile) and server-side (e.g. ASP.NET, PHP and JSP). The capabilities of the debugging tools include creating breakpoints, stepping through code, expression watching, stack tracing and other functionalities. Debugging HTML5 mobile Web apps is much easier than debugging traditional mobile Web apps because the jQuery Mobile is simple and concise. Moreover, developers can easily adopt some plugins in the Web browser to debug jQuery code, such as FireBug, FireQuery, FireFinder and Opera Dragonfly.

Esiness of debugging	Points
Average points	3

Ease of testing

- Test project

Developers can use many tools available to create test project testing HTML5, jQuery Mobile, CSS3, for example CSS3 Test, Front Drag and Aptana. The tools have very intuitive interface that bring developers a friendly interaction. Moreover, there are many tutorials and document to instruct how to create test projects by using the tools.

- Emulator

  Developers can easily adopt the emulators provided by the IDEs as discussed above to test their HTML5, CSS3 and jQuery Mobile code. They can test their application under development or after deploying it on the server. Additionally, developers can use Opera emulator available for testing Web apps to test their application.

- Real device

  Testing HTML5 mobile app on a real device is straightforward. We only have to load the app onto our mobile browser and run the app from our device. Sencha Touch, jQTouch, jQuery Mobile and other powerful frameworks ease the development of HTML5 mobile Web app but using such frameworks cannot guarantee flawless cross-browser compatibility. For example, the HTML5 mobile apps that use device camera can only run on Opera Mobile for Android. This means that developers should still try to test their apps on as many different devices as possible to guarantee that the apps will work across platforms and browsers.

Ease of testing	Points
- Test project	4
- Emulator	5
- Real device	4
Average points	4.33

Ease of deploying and updating

Developers can deploy their application on their Web server. The deployment means no payment to app store platform and no approval process. After the apps are hosted on the Web server, there is no lengthy process of having to resubmit improved and updated versions of the apps and have to go through approval processes all over again. It means that developers can easily maintain and update the code, and the negative impact of any problems on the delivery of service to the end users can be very efficiently addressed and minimised. However, developers must set up their own Web server and deploy the apps by themselves on the server. They are also responsible for creating access control to the apps.

Esiness of deploying and updating	Points
Average points	3

Ease of distributing

The distribution of HTML5 mobile Web apps is easier than ever before. As soon as HTML5 mobile apps are hosted on a server, they are accessible for every mobile device from everywhere [61].

Powerful tools and libraries	Points
Average points	5

Application types

- Application using device capabilities

  Mobile Web apps cannot fully access hardware features of the devices. However, the W3C is delivering a set of APIs that make mobile Web apps become much more powerful and capable of interacting with the devices in the same manner in which native apps do. The implemented APIs include Geolocation, contacts, calendar, media capture API, local storage API, messaging and gallery. Furthermore, HTML5 enables mobile Web apps to adopt local storage of mobile browser for temporarily saving data to work offline. Developers can flexibly specify which file should be stored on mobile devices and this save mobile Web app from blank page if the user does not have Internet connection.

- Application using server capabilities

  Mobile Web apps can employ every capability of the Web server, such as storage, network content and processing ability.

- Application using real time communication

  One of the greatest features of HTML5 is the benefit it provides for gaming industry. HTML5 introduces many features for making an appealing game such as canvas, jQuery Mobile, video and audio elements, and Web worker providing the game apps with the ability to run multi-threaded script on the background without interacting with mobile user. As we combine HTML5 with other exciting Web technologies that make up an Open Web Platform, we create very interesting real time games with multi players.

  HTML5 mobile Web apps also work great to serve mobile users with banking, weather forecast and stock market services.

  Geolocation is another HTML5 application that can make very good user experience. The apps can pinpoint exact position by using GPS sensors built in mobile devices. The apps can also adopt Google Maps to show the current location of mobile users on a map.

Application types	Points
- Application using device's capabilities	4
- Application using server's capabilities	5
- Application using real time capabilities	4
Average points	4.33

Powerful tools and libraries

As discussed above, there are many powerful tools and libraries for creating HTM5 mobile Web apps that are open source or commercial. W3C are developing several APIs to provide developers with capabilities to create mobile Web apps which are as powerful as native apps. Some are implemented, such as Geolocation, contacts, calendar, media capture API, local storage API, messaging and gallery. Some are still under development.

Powerful tools and libraries	Points
Average points	5

Payment possibility

There are many payment possibilities for HTML5 mobile Web apps. Developers can take advantage of mobile app store to earn the revenue from each subscription to their apps. They can also publish their apps on their Web server and make them available for subscription. In this way, developers do not have to share their revenue with app store but they have to create the access control on each subscription by themselves. Another way to distribute their mobile Web apps is via Facebook as mentioned in [33].

Content management	Points
Average points	5

The table below summarizes **the evaluation for the HTML5 mobile Web apps** from **developer viewpoint**

Table 13: Summary of evaluation on HTML5 mobile app paradigm from developer viewpoint

Ease of developing	4.67
Ease of coding	5
Ease of debugging	3
Ease of testing	4.33
Ease of deploying and updating	3
Ease of distributing	5
Application types	4.33
Powerful APIs and libraries	5
Payment possibilities	5
Average points	*4.37*

4.4.2. User viewpoint

Ease of use

- Performance

  HTML5 mobile Web apps are lightweight and hence run fast in most of smart phones. It is more responsive and attractive than old mobile Web apps and widgets but have *slower performance* than mobile native apps. Web apps sometimes lag because they should be loaded into mobile browser.

- User interface

  HTML5 provides several new elements to improve the presentation of the Web apps, making it more attractive than mobile widgets and old Web apps. The improvements include animation (e.g. fade, slide, pop, flip and cube), audio and video handling. Hence, HTML5 apps now look similar to native apps. Additionally, the users who do not have Flash installed on their devices (e.g. iOS user) really benefit from the capability of playing video on HTML5 Web apps.

- Operation

  Similar to using any Web apps, we have to open mobile browser and enter the URI to load all the HTML5 apps at first, including both user interface and content. Even though we can simply create a flashy-app like icon on user home screen by creating a

bookmark of the site, entering exactly the URI into a small address bar in the first time is still a problem.

Ease of use	Points
- Performance	4
- User interface	4
- Operation	3
Average points	3.67

Functionality

- Working offline

  One of the most interesting features of HTML5 mobile apps is the capability to work offline by storing temporarily data on local storage supported by mobile browser. This capability drives HTML5 mobile apps closer to native apps as the key features of the apps still work even if users have a poor Internet connection. However, the HTML5 apps that work offline have some concerns, such as synchronization between the data in local storage and data in remote database.

- Accessing device's hardware

  Another advantage of HTML5 mobile Web app is that it can access several device hardware features (but not all). The apps hence are functional and make a good user experience.

- Using real time communication

  Mobile HTML5 gaming is excellent but real-time HTML5 gaming is still under development. Other real time apps (e.g. banking, weather forecast and stock market display) run very effectively on mobile devices and attract many mobile users. In addition, mobile user now can enjoy Geolocation capability of HTML5 mobile apps. The apps can help users find out their current location by adopting Geolocation service. A sample of using Geolocation service is that Facebook users can get the event location from their Facebook account, find out their own location by adopting built-in on their phone and figure out how to travel to the event location from current location.

Functionality	Points
- Working offline	3
- Accessing device's hardware	3
- Using real time communication	4
Average points	3.33

Installation and update

- Compatibility

  Compatibility is currently a problem of HTML5 mobile apps. There are still some devices do not support HTML5 and HTML5 Web apps cannot run on such devices.

- Downloading and installing

Mobile users do not have to download and install HTML5 mobile apps on their device. Update also happens on Web server. Therefore, users will be able to reach the newest content provided by the HTML5 mobile apps and newest version of the apps.

Installation and update	Points
- Compatibility	3
- Downloading and installing	5
Average points	4

The table below summarizes **the evaluation for the HTML5 mobile Web apps** from **user viewpoint**

Table 14: Summary of evaluation on HTML5 mobile app paradigm from user viewpoint

Ease of use	3.67
Functionality	3.33
Installation and update	4
Average points	*3.67*

4.4.3. Service provider viewpoint

Content management

- Content presentation

The limitations of mobile devices (e.g. small screen size, unusable keypad and low computing capabilities) create many difficulties for content providers to structure their Web apps. Fortunately, HTML5, CSS3 and JQuery Mobile can change the way content providers structure their content. Below are some new features that HTML5 introduces to content providers.

- <input> has autocomplete and autofocus attributes to ease the user input. The attribute autocomplete will limit the length that the user must enter and autofocus makes the <input> get focus when the page loads.
- <canvas> is the new element which can be used to draw graphics using scripting (JQuery Mobile). <canvas> make it easier to draw a graph, make a photo composition or do simple animation. On other words, <canvas> helps content providers port a picture or animation to end users very effectively.
- <video> and <audio> help present the content that contain animation, video and audio much more easily than old mobile Web application.
- Local storage, new input time (e.g. date time) and browser-support form validation which are natively supported by the language can ease the display of content and hence make communication between mobile users and content providers easier. For example, instead of launching JavaScript-based calendar date-pickers in old mobile Web apps, date element is available in HTML5 display the date faster on mobile browser

- HTML5 can also help content providers hide the address bar of the HTML5 mobile app when the app loads on mobile devices, making it similar to a native app.
- CSS3 takes the idea of media type one-step further and enhances their functionality with media queries. Media queries extend the usefulness of media types by allowing more precise labelling of style sheets. This customizes the content's presentation to a specific range of output devices without having to change the content itself.

- Content delivering

Delivering content to mobile user is no longer much complicated thanks to HTML5, CSS3 and JScript. However, the adoption of local storage can challenge content providers in the way that they must ensure the synchronization between local storage and database server. Moreover, content providers must port the whole page to end users for the first time user access the Web apps. This can cause increase in network overhead and is quite inefficient as compared to native apps and mobile widgets

Content management	Points
- Content presentation	3
- Content delivering	3
Average points	3

Administration

- Security

Service providers must authenticate the users who want to access HTML5 mobile apps on the server. They must also protect the integrity of sensitive data (e.g. in banking transaction) exchanged between mobile users and Web server. The responsibility in ensuring security mainly stays at server's side.

- Maintenance

Service providers are responsible for hosting, updating and managing versions of HTML5 mobile apps. An advantage of maintaining HTML5 mobile apps is that service providers only have to update their apps on server. Mobile user will automatically reach the newest app version when they connect to the server and use the apps.

Administration	Points
- Security	3
- Maintenance	4
Average points	3.5

Distribution

Service provider can easily distribute their apps to mobile users. Instead of having to develop dedicated applications for each mobile platform, to give their customers a compelling experience, they will be able to offer a single HTML5-based app that will run across mobile devices-greatly reducing their development costs. Furthermore, making the

sites available through HTML5 mobile apps can break the monopoly of app store. For example, instead of paying a proportion of revenue for each subscription on newspaper or magazine to the app store, publisher/ content provider can sell their subscription directly to the end users. They should need a simple Web authentication of the subscription and the Web apps will be available on any mobile device that supports HTML5.

Distribution	Points
Average points	5

The table below summarizes the evaluation for **HTML5 mobile Web apps** from **service provider viewpoint**

Table 15: Summary of evaluation on HTML5 mobile app paradigm from service/content provider viewpoint

Content management	3
Administration	3.5
Distribution	5
Average points	3.83

The table below summarizes **the evaluation for HTML5 mobile Web apps** from the **developer, user and service provider viewpoint**

Table 16: Summary of evaluation on HTML5 mobile app paradigm

Developer	User	Service/content provider
4.37	3.67	4

The table below summarizes **the evaluation for different mobile application paradigms from developers, user and service**

Table 17: Summary of evaluation on four mobile app paradigms

Mobile app paradigms	Developer	User	Service/content provider
Native apps	4.04	3.78	3.17
Web apps	3.26	2.5	3.67
Widgets	3.91	3.22	3.5
HTML5	4.37	3.67	3.83

Chapter 5: Practical verification

In order to verify the evaluation carried out in the previous chapter, we will build a mobile object recognition/visual search app using the two most promising paradigms: native app and HTML5 mobile app. The objective of the practical verification is to ensure that the performed evaluation is conformed to the reality and consequently usable. The same app will be developed using both paradigms and evaluated according to developer and user's viewpoints. Comparisons with the former evaluation will be then carried out.

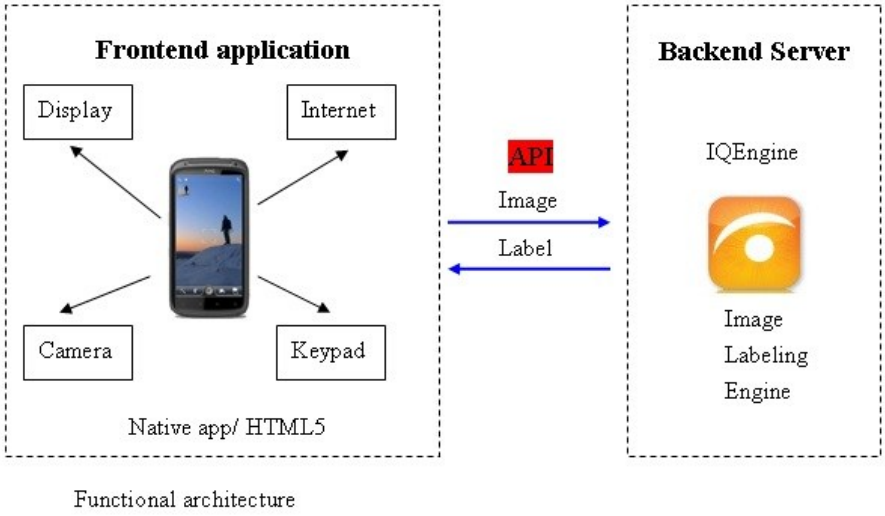


Figure 17: Technology architecture implementation

The purpose of the object recognition app is to provide to the user information about a requested object which can be anything from a glass, a car, a person to a building, a monument or a mountain. The user uses the camera on his/her mobile phone to capture a picture of the wanted object and sends it to a server asking for identification and information.

The motivation for choosing the object recognition app is manifold. First, this app needs to interact with a server on the Internet and requires a connection. Second, it needs to use local facilities on the mobile phone such as camera. Finally, the object recognition app is in itself a useful and exciting application on mobile phones.

Figure 17 depicts our technology architecture implementation. We employ the **IQEngine API** as the backend service, and a native app or an HTML5 as the frontend app which runs on the device. The reason we choose IQEngine for the backend service is that IQEngine API is very usable, and the object analysis and recognition is quite fast (up to 1 minute per object). Moreover, we receive a lot of support from IQEngine developer centre such as tutorials and troubleshooting, and 1000 free visual scans from the service. We can also create our own training database to reduce the time of visual search, which is an advantage over the other online visual discovery engines. The detail about IQEngine visual discovery service can be found in the annex section. The other selections for the

backend service are Tineye and Google Goggle. However, they are not as advantageous as IQEngine. Tineye costs $300 for 1000 visual searches and it does not provide much support for mobile developers. Meanwhile, Google Goggle does not have the API for developers.

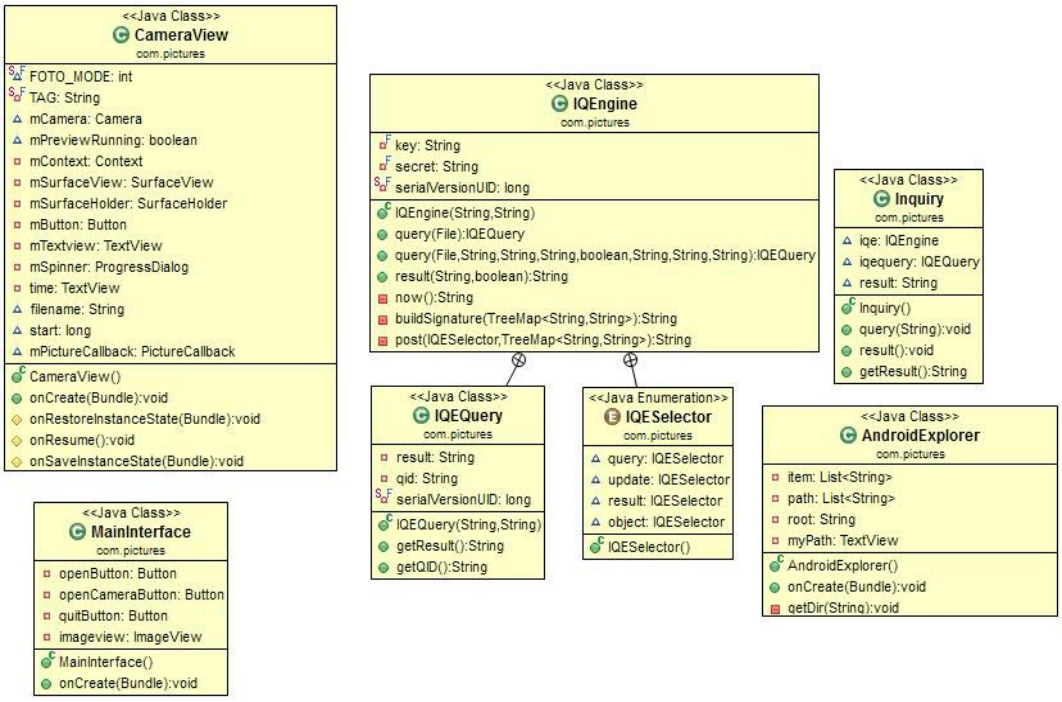


Figure 18: Class structure of the native app

The frontend apps can access the camera and keypad, connect to the Internet and display the result on device's screen. User will capture the image of unknown objects by using device's camera and send it to IQEngine server by using the available APIs. The server subsequently analyzes, compares the image with the images in its training database, labels the objects in the image and responds the device. Finally, the frontend app will display the label on device's screen. The mobile phone adopted for the implementation is the Samsung Galaxy GTI-5500 with Android 2.1.

5.1. Native app

Android, iOS and Windows Phone SDK provide developers with several valuable APIs and libraries to create an app which can interact with device's camera. Therefore, we choose Android SDK and Eclipse IDE to build the native app for the practical verification. The app then can run on the device with Android platform.

5.1.1. Native app architecture

- **Class structure**

The app's architecture has five main classes, including HomeInterface, CameraView, AndroidExplorer, Inquiry, IQEngine classes (Figure 18). HomeInterface defines the home user interface of the app. From the home interface, users can choose to capture an image by using camera or load an available image from device's storage. CameraView is responsible for creating device's camera interface. The CameraView can be considered as

the client for the Camera service, which manages the actual camera hardware. AndroidExplorer helps users select the available image file to upload to IQEngine server. Inquiry initiates an API object using API key and secret, queries the image, retrieve the result in JSON format and decode JSON object into String object. IQEngine creates the package (e.g. the timestamp, image, API key and API signature), sends it to IQEngine server and obtains the result from the server.

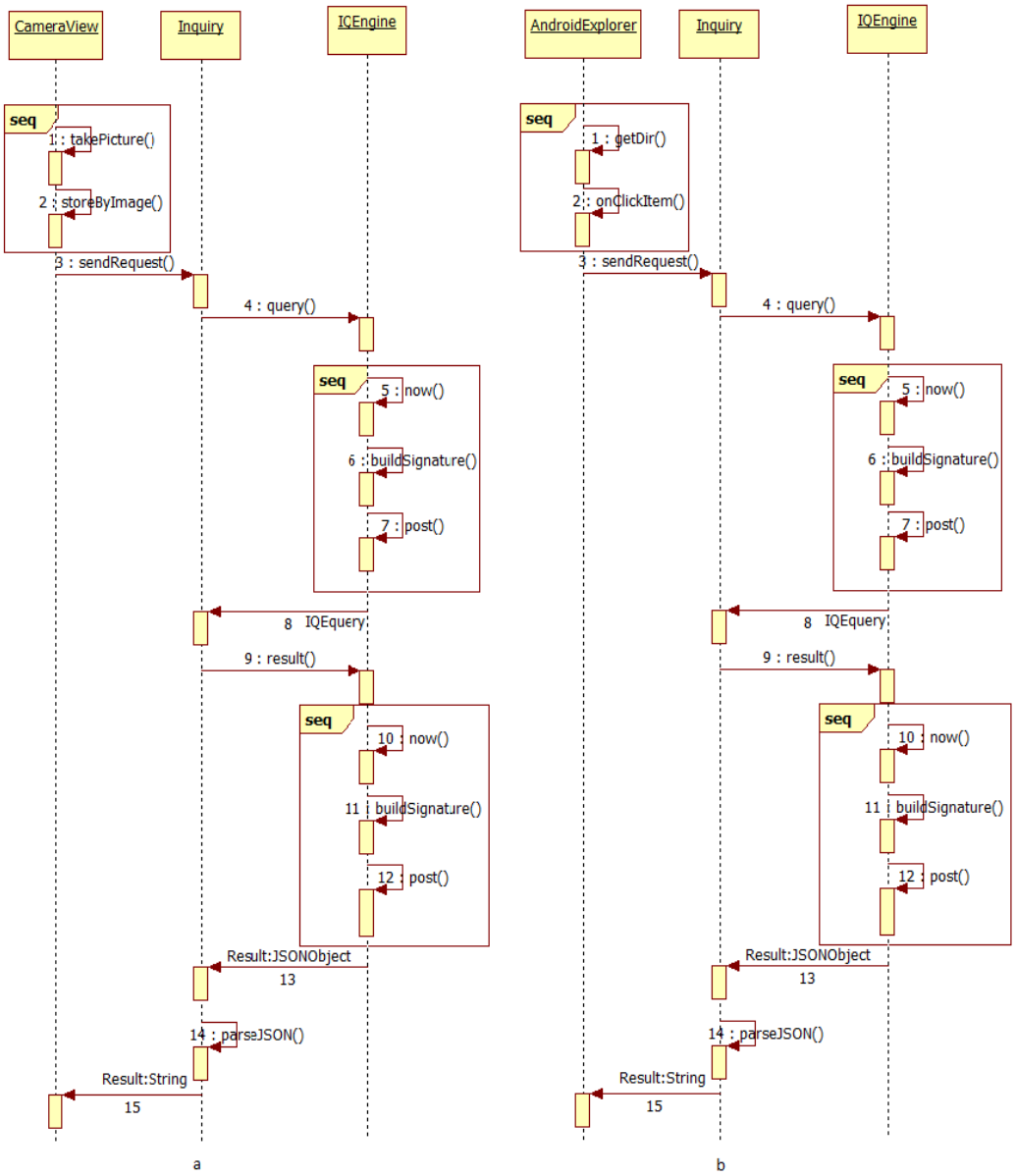


Figure 19: Sequence diagram of the app when users want to capture a picture by using device's camera (19a) or select a picture from device's storage (19b)

- **Sequence diagram**

Figure 19 shows the sequence diagram of object recognition native app. Users will interact with the app via CameraView class (**1**), capture an image and save it on their device's storage (**2**) as in Figure 19a. They can also choose to open an available image from storage's device as in Figure 19b. The app asks the users whether they want to send

the inquiry. If the users want the inquiry, the app sends the request (**3**) to Inquiry object. Inquiry object call query() method to IQEngine object (**4**), thereby initiating the call to IQEngine API. IQEngine object gets the timestamp (**5**), builds the signature by using HMAC-SHA1 algorithm (**6**) and send all the parameters (e.g. timestamp, API key, signature and image) to IQEngine server by using httppost (**7**). IQEngine object then retrieves the IQEquery object from IQEngine server and responses Inquiry object with the IQEquery object (**8**). Inquery object calls result() method with qId argument which is extracted from IQEquery (**9**). IQEngine object again gets timestamp (**10**), computes signature (**11**) and sends request which include qId to IQEngine server (**12**). IQEngine object retrieves the JSON object from the server and sends it back Inquiry object (**13**). Inquiry object parses the JSON object (**14**) to obtain the label and sends the label to CameraView/ AndroidExplorer (**15**). CameraView/AndroidExplorer finally displays the label on device's screen.

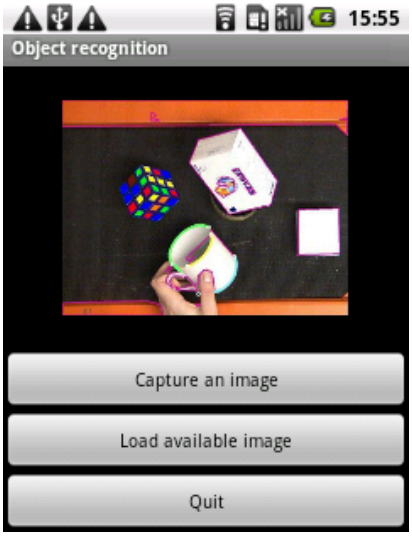


Figure 20: Home interface of the native app

5.1.2. Analysis and evaluation

- **Developer viewpoint**

Ease of developing

- Programming language

  As we discuss in the previous chapter, Android allows programmer to write the code with Java-like language which is easy to learn and use. We do not have any challenge in programming language to build the native app.

Figure 21: Users choose to capture a picture of the wanted object. 21a is the interface of device's camera. 21b shows the screen when users tap on the screen to capture a picture. As soon as they choose "yes", the app sends the picture to IQEngine as shown in 21c, receives the result after about 25 seconds, and displays it on device's screen as in 21d.

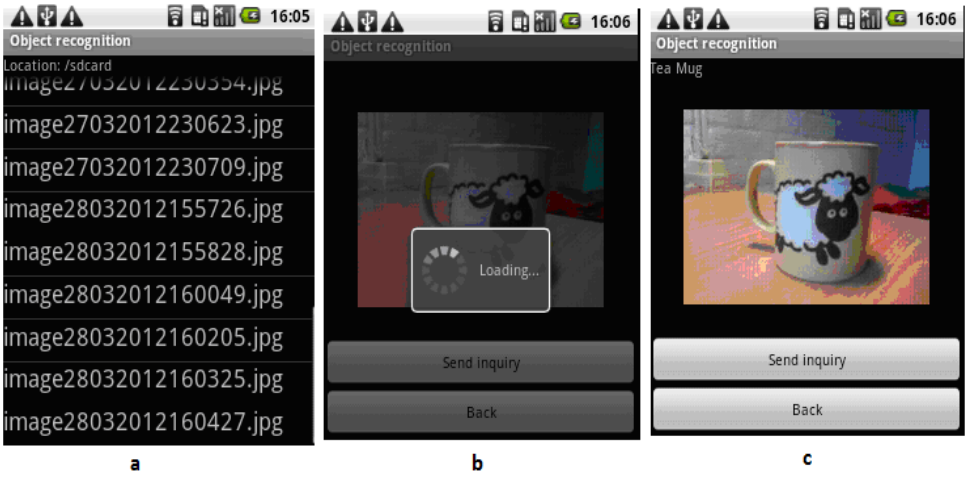


Figure 22: Users want to load an available picture from the SD card. 22a shows the list of image files stored in sdcard folder. 22b depicts the waiting screen when the picture is sent to IQEngine server. 22c describes the result obtaining from the server.

- Software development kit (SDK)
  - Applicability

    The SDK is very applicable and easy to use. We can write the code, debug, and perform the test case and the implementation of the app very effectively with many supports from the SDK. For example, the SDK supports us to create intents and activities, and call a new activity from an activity. When users click on the

button "capture an image" on the home user interface (Figure 20), the app will create a new intent CameraView, thereby building and running CameraView activity as shown in Figure 21a. When users finish the camera, they simply click the button "Quit the camera" to destroy the CameraView activity and return the HomeInterface activity. Similarly, when users click to select the button "Open available image", a file dialog appears as shown in Figure 22a.

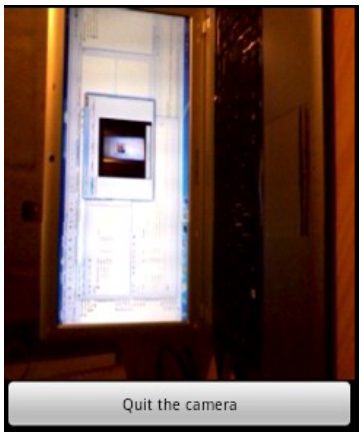


Figure 23: Camera preview is sideways on Android 2.1 or lower.

- Specifications and tips

  We can find many useful specifications and tips from the developer page of the SDK by simply clicking on the class or method on our code editor.

- Interaction with device's hardware

  The SDK supports interactions with camera, keypad, WLAN card and device's storage. We can deal with the device's camera by using android.hardware.Camera, android.view.SurfaceView, and android.view.SurfaceHolder. We can also interact with the device's storage by using java.io.file and check the Internet connection by using android.net.ConnectivityManager before sending the request to IQEngine server.

  However, on Android 2.1 and the earlier versions, the SDK does not support the methods and fields android.view.display. getRotation(), Surface.Rotation and android.hardware.Camera.setDisplayOrientation(), making the camera preview display at an incorrect angle degree in portrait mode as shown in Figure 23. Moreover, there is no file dialog support from Android SDK to locate a file on the device's storage. Therefore, we have to write the code by ourselves to list all the files in an indicated folder and then select the wanted image file.

- Downloading, installation and configuration

  The SDK can be downloaded free from Android developer Website. However, it requires a large amount of time to install the SDK and the integration of the SDK into the IDE (e.g. Eclipse) is complicated at first as described in [41].

- SDK	Points
o Applicability	5
o Specifications and tips	5

○ Interaction with device's hardware	4
○ Downloading, installation and configuration	4
Average points	4.5

- Support from community

  Community of Android developers is worldwide, which benefits us a lot. We have found a lot of helpful tutorials, code examples and instructions to build the app and fix the bugs. Several supports that we have found on the Internet are from developer.android.com, stackoverflow.com and code.google.com.

Ease of developing	Points
- Programming language	5
- SDK	4.5
- Support from community	5
Average points	4.83

Ease of coding

- IDE's capability
  - Code editor

    The IDE (Eclipse) is very usable. The code editor is intuitive, which enhance the code management. The instant feedback with error and warning, import structure, open type feature, content assistant and quick fix features are great benefit to us while we write the code by using the code editor.

Figure 24: The home user interface of the native app in portrait and landscape modes. The buttons and the image display inappropriately in landscape mode.

  - User interface builder

    We have found Eclipse UI builder inefficient to use. There is no drag and drop capability provided by Eclipse UI builder. Therefore, we have to write the code directly in main.xml, camera.xml, imageview.xml and folderview.xml, and declare the UI variables (e.g. android.widget.button and android.widget.textview) to

detect the UI controls defined in the xml files. We can preview the user interface by clicking the graphic layout tab.

- IDE's capability	Points
o Code editor	5
o UI builder	3
Average points	4

- User interface

  Android SDK provides us with a range of Android layout: AbsoluteLayout, FrameLayout, LinearLayout, RelativeLayout and TableLayout. The layouts are straightforward to create an intuitive user interface but they are not effective to build an adaptive user interface. For example, we cannot make the home user interface display properly in both landscape and portrait modes. The UI elements (e.g. button, textview, and imageview) do not automatically change the size and rearrange to adapt to the screen size when the device rotates. Figures 24a depicts the home user interface in portrait mode. The user interface displays inaccurately in landscape mode as in Figure 24b.

- Device's interaction

  - Camera

    To make the app access the device's camera is quite complicated though we use android.hardware.Camera provided by Android SDK. Firstly, we declare camera permission in our manifest.xml to ensure that the app is allowed to access the camera. Then, we create a camera instance and set the parameters for the camera. Without the SurfaceHolder configuration, the camera will be unable to start the preview. Therefore, we also have to build the camera preview by using a SurfaceView and a SurfaceHolder. The preview must be started before we take a picture and close after the picture is taken. Finally, we have to write the code in the Camera.Picture.Callback to store the image onto device's storage as soon as the image is captured. To access the camera, capture an image and save the image onto device costs at least 140 lines of code.

    As discussed in SDK criterion, we cannot make the camera work in portrait mode due to the limitation of the android.view.display, surface and android.hardware.Camera classes on Android 2.1. Therefore, we must develop two versions of the code, one for Android 2.1, and the other for Android 2.2 or higher so that the app can run properly on all versions of Android.

  - Downloading and uploading an image

    Unlike HTML5 mobile app, the native app does not download the captured image but we have to write the code to encode the captured camera stream into an image in .jpg or .png format and save it on device's storage.

    Android SDK does not support file dialog. Therefore, we have to create our own file dialog from the scratch and it is a time consuming task. Firstly, we build the user interface of the file dialog by using android.widget.ListView and android.widget.TextView. Then we write the code to list all the files and subfolders

in an indicated folder by using java.io.File. We finally display the chosen image on an image view on a new activity (ImageView activity).

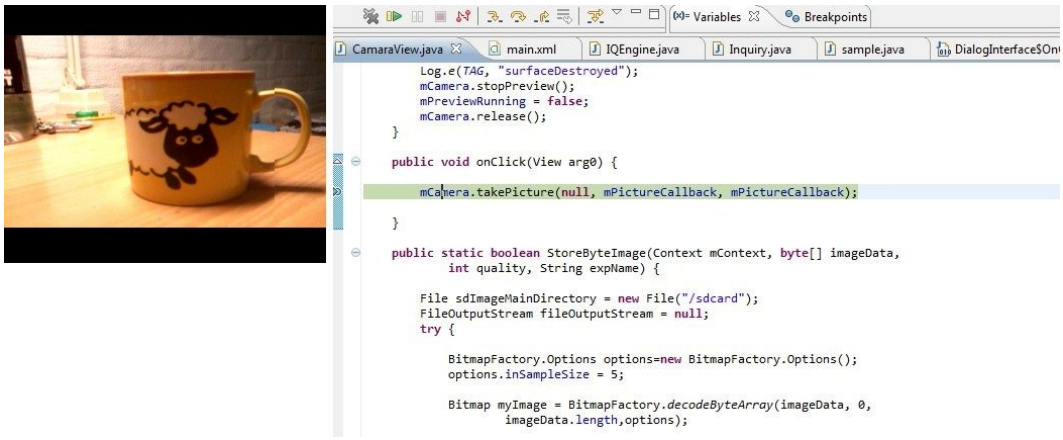


Figure 25: Debugging the image capture event

- Internet connection

  We can simply make the app connect to the Internet check the Internet connection by providing it with the permission to access the Internet.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

We can also check the Internet access by adopting Android.net.ConnectivityManager as follows

```
ConnectivityManager cm = (ConnectivityManager) getSystemService
(Context.CONNECTIVITY_SERVICE);
NetworkInfo netInfo = cm.getActiveNetworkInfo();
```

- Native apps (Camera and photo album)

  The app cannot access the native camera and photo album app.

- Device interaction	Points
o Camera	3
o Downloading and uploading an image	1
o Internet connection	5
o Native apps	1
Average points	2.5

- IQ Engine server interaction

In order to access and use IQEngine API, we need to get the timestamp on our mobile device, compute the signature and send a request to IQ Engine server and receive the result from it.

- Getting timestamp

  We can simply get the current timestamp with the following format "YYYYmmDDHHMMSS" by using java.io.utils.Calendar, java.io.utils.TimeZone and java.text.SimpleDateFormat. It costs 5 lines of code to get the current timestamp.

- Building signature

  Computing the signature requires us to import the javax.crypto and org.apache.commons.codec packages. The org.apache.commons.codec belongs to Apache common codec library which can be downloaded from [44]. We have to do the following tasks to build a signature with HMAC-SHA1 algorithm: creating a new Mac instance that provides the HMAC-SHA1 algorithm and initializing it with a secret key, digesting the Mac based on the data bytes received from the image name, timestamp and API key, and converting raw bytes to Hex and then Hex bytes to signature string. It costs 20 lines of code to build the signature.

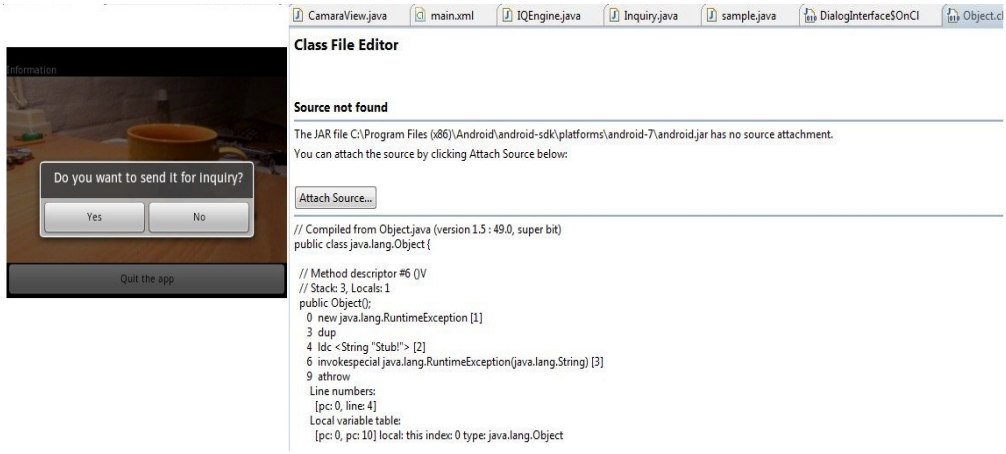


Figure 26: An unknown error happens when debugging the image capture code

- Sending request and retrieving result

  In order to send an inquiry to IQEngine server and get the result back from the server, we use http protocol (http post and response). It requires us to import the org.apache.http into our project which can be downloaded from Apache Website. Firstly, we create http request/http post which contains the needed entity including image, API key, signature and timestamp. Subsequently, we create a new http client to execute the http post and return the http response. Finally, we get entity from the response and stream it if needed to retrieve the result string. It costs 35 lines of code to send request, receive the response and stream the response to get the result.

- IQEngine server interaction	Points
o Getting time stamp	4
o Building signature	3
o Sending request and retrieving result	3
Average points	3.33

Ease of coding	Points
- IDE's capability	4
- User interface	3
- Device interaction	2.5
- IQEngine server interaction	3.33
Average points	3.21

Ease of debugging

We can easily manage the breaking points and trace the variables of the app at runtime as shown in Figure 25. The debugging tool works inefficiently with the camera-related packages. It always stops unexpectedly with a "mysterious" reason as in Figure 26. We can only find out the problem by checking the logcat file after all.

Ease of debugging	Points
Average points	2

Ease of testing

- Emulator

  The emulator is not efficient to test the apps. We cannot use the local computer camera in Android emulator.

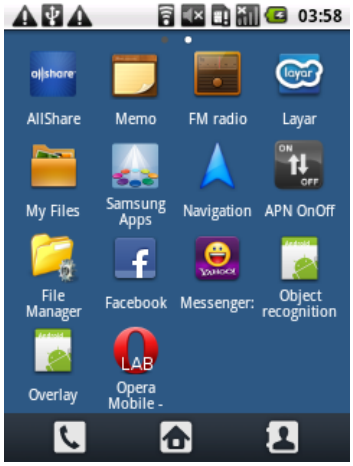


Figure 27: Deploying the object recognition app onto Android device.

- Real device

  We can easily test the app with a real Android phone in a few simple steps. In the AndroidManifest.xml, we add android:debuggable="true" to the <application> elements. We turn on "USB debugging" on our devices and install USB driver for adb. Then we plug in the device over USB and find out the name of the device listed as "device". Finally, we can test the app which is running on the device very effectively by using the SDK and Eclipse IDE. However, if an exception happens, the app will stop unexpectedly without indicating the name of the exception in the console window. Therefore, we have to check the logcat file to find out the exception and fix it.

Ease of testing	Points
- Emulator	1
- Real device	5
Average points	3

Ease of deploying and updating

It is very straight forward to deploy and update the app onto our device (Figure 27). We can connect the device with our computer and test the app. The app will subsequently be deployed on our device. As soon as we update our project on computer, the app is

updated on our device. We can also deploy the app by copying the .apk file generated through the code compilation onto our device and running the app from the device.

Ease of deploying and updating	Points
Average points	5

Ease of distributing

- Compatibility

  The object recognition app can only run on Android devices. It is not a cross-platform app.

- Without app store

  As we discussed above, to distribute the app is very easy. We only have to distribute the .apk file to the Android devices and run the app from there. There are many solutions for distributing the app. We upload the .apk onto a Website and users can conveniently download and run it. We can also share it via email or Dropbox sharing.

- With app store

  We have to pay $25 for the registration fee and then can publish our app onto Android market. We must also go through several complicated steps to get the license from Android market to publish our app.

Ease of distributing	Points
- Compatibility	1
- Without app store	5
- With app store	3
Average points	3

Application type

- Application using device's capabilities

  Our app can access the device's camera, keypad, storage (e.g. SD card) and connect to the WLAN quite effectively. Users can tap the screen to capture the image, save the image on the SD card and send it to the server afterward. A disadvantage of the app is that the camera preview is sideways and always displays the picture at an incorrect 90 degree angle in portrait mode on Android 2.1 or older (Figure 23). Therefore, users must rotate the device to get the camera preview in landscape mode when they start the camera. The reason of the disadvantage is the lack of some methods on Android 2.1 which are available on Android from version 2.2 to support the SurfaceView rotation as discussed above.

- Application using server capabilities

Our application employ the IQEngine server's capability (e.g. processing and storage) to identify unknown object.

Application types	Points
- Application using device's capabilities	4
- Application using server's capabilities	5

Average points	*4.5*

Powerful APIs and libraries

Android SDK is really useful and applicable, providing us with the powerful APIs and libraries. The libraries and APIs enable us to build the app which can access device's camera (e.g. android.hardware.Camera and android.view.SurfaceHolder), employ IQEngine API (com.pictures.IQEngine.IQEQuery) and decode the JSON response (e.g. org.json.JSONObject).

Powerful APIs and libraries	Points
Average points	*5*

Payment possibilities

We can only publish their apps on app store and get revenue for the sale of the apps. For example, developers get 70% of sale revenue. It is considered as the most solid and efficient way to sell the native app.

Payment possibilities	Points
Average points	*3*

The table below summarizes the evaluation of native app paradigm on developer viewpoint after the practical implementation

Table 18: Summary of the evaluation on the native app from developer viewpoint

Ease of developing	4.83
Ease of coding	3.21
Ease of debugging	2
Ease of testing	3
Ease of deploying and updating	5
Ease of distributing	3
Application types	4.5
Powerful APIs and libraries	5
Payment possibility	3
Average points	*3.73*

- **User viewpoint**

Ease of use

- Performance

  The app is very responsive on the Samsung Galaxy GTI-5500 with Android 2.1. The time to run the app includes the amount of time to start the app, start the camera, choose an available image at the device and send the image to the server, and the time of object recognition process on IQEngine server. We run 50 tests to measure the time needed for using the app. The average time to start the app is from 120 to 380 milliseconds. Meanwhile, the amount of time to start a camera is within 1150 and 1500 milliseconds, and to capture a picture is about 800 milliseconds. The time to send the request and get the result back from the server, and display the result is about 25 seconds.

- User interface

  The app has an intuitive and attractive user interface with button, dialogbox and camera view, making users easy to interact with the app. The home interface of the app has three buttons as in Figure 20. When users select to capture a picture, the app starts the camera as in Figure 21a. After tapping the screen to take a picture, the app will ask the users if they want to send the picture for the identification as in Figure 21b. If so, the app will send the picture and receive the label of the object as in Figures 21c and 21d respectively. In case the users want to choose an available image on device's storage and send it to the server, the app will displays as in Figures 22a, 22b and 22c respectively. However, it takes 1 or 1.5 second to start the camera and the camera interface lags when users capture an image. Moreover, the app can only display the home user interface in portrait mode, and the camera interface in landscape mode.

- Operation

  The app is very usable. Users can easily launch the app by clicking the app's icon on home screen. They can also conveniently tap the camera's interface to capture an image, save it on device's storage and send it to IQEngine server for the recognition process. The image stored onto device's storage has the name including the timestamp of the image (e.g. image28032012160049.jpg). Additionally, users can access IQEngine Web service and get the label of the object in two simple steps: taping the camera's interface or selecting an available image on device's storage and agreeing to send it to the server. They finally view the result on screen right after the app receives the label from IQEngine.

Ease of use	Points
- Performance	5
- User interface	4
- Operation	5
Average points	4.67

Functionality

- Working offline

  The app requires Internet connection to access IQEngine Web service. However, in case there is no Internet connection, users still capture the image, save it on device's storage and send it to the server as soon as they can connect to the Internet.

- Accessing device hardware

  The app can access device's camera, keypad and storage. Moreover, it can also adopt the Internet connection very effectively. However, on Android 2.1 and below, users can only use the device's camera in the landscape mode (Figure 21).

- Object recognition

  Currently, the app can be used to label the objects only. The objects detected include glass, cup, statue, building, car and so on.

Functionality	Points

- Working offline	3
- Access device's hardware	4
- Using real time communication	4
Average points	3.67

Installation and update

- Compatibility

  The app can only run on Android's device.

- Downloading and installing

  The app is very lightweight to download from the Internet and install on mobile devices. However, updating the app create a little big challenge. Users have to download the new version of the app onto their devices to update the app.

Installation and update	Points
- Compatibility	1
- Downloading and installing	4
Average points	2.5

The table below summarizes the evaluation of native app paradigm on user viewpoint after the practical implementation

Table 19: Summary of the evaluation on the native app from user viewpoint

Ease of use	4.67
Functionality	3,67
Installation and update	2.5
Average points	3.61

5.2. HTML5 mobile app

We create an object recognition mobile app by using HTML5 mobile app paradigm. Actually, it includes HTML5, which only works at client side, and PHP, which is the server side script. Similar to the native app, our HTML5 mobile app can interact with device's camera and storage. Users capture a picture, send it to IQEngine server and receive the result afterward. The difference between the two is that users must connect to the Internet and load the app onto their device, and the app only works within mobile browser. We use **Aptana Studio** as the IDE for coding the app and **Opera Dragonfly** for debugging and testing the code. We also take advantage of the 000Webhost to host our HTML5 mobile apps including HTML5 files, jQuery Mobile, JavaScript, CSS3 files and PHP scripts. Currently, the app can only work on **Opera Mobile** for Android because only that version of Opera browser supports the getUserMedia API [42] to access device's camera.

5.2.1. HTML5 mobile app architecture

The app hosted on 000WebHost.com includes HTML5 and JavaScript files to interact with our devices (e.g. using camera and storage), PHP script to upload a file from the devices and to make IQEngine API request, and the CSS3 and jQuery files to define the page loaded onto mobile browser.

HTML5 provides us with the <video> element and navigator.getUsermedia() to interact with device's camera and the <input> element to create the file dialog on HTML5 page.

Figure 28: HTML5 mobile app architecture

We adopt jQuery Mobile to design our **user interface** thereby releasing our headache to make the app look like a native app. jQuery Mobile is a framework built on top jQuery that provides a range of user interface elements and features to use in mobile apps. The app uses several interface elements and features of jQuery Mobile such as of **pages within pages**, **Ajax navigation**, **page transition, orientation on change** and **theming**.

In addition to jQuery Mobile, we use CSS3 to define our page, creating better user experience. The CSS3 features adopted in our app are media –o-paged-x and media orientation.

Unlike the native app, the device will interact with our app hosted on 000WebHost server (as in Figure 28) rather than directly communicating with IQEngine server. After analyzing and recognizing the object, IQEngine server will label the object and send the result back to the device. Figure 28 describes the way that users access and use the app. Users open mobile browser (Opera Mobile) and enter the URL huy.host56.com **(1)**. The app will be loaded onto the browser (index.html). The HTML5 page can access device's camera and storage **(2)**. Users will capture a picture via the app **(3)** and download the picture **(4)** onto the device. Users choose the picture from device's storage **(5)** and send it to 000WebHost server via http post **(6)**. The name of the image uploaded onto the server includes the timestamp when we upload it (e.g. image28032012160049.jpg). The app sends the inquiry to IQEngine server **(7)**, receives and displays the label **(8) (9)**. Similar to the native app, the parameters that the HTML5 mobile app delivered to the IQEngine server include the timestamp of the image, the API key and the signature created by using HMAC-SHA1 hashing algorithm.

5.2.2. Analysis and evaluation

- **Developer viewpoint**

Ease of developing

- Programming language

HTML5, CSS3, JavaScript and PHP are short and straightforward to code the app.

- Software development kit (SDK)
  - Applicability

    We use Aptana which is quite applicable to develop the app. Aptana SDK allows us to manage the files (e.g. image, HTML file, CSS and PHP scripts) and write the code effectively. It supports the latest HTML5 specifications and CSS3 features. It also provides us a useful IDE with several valuable functionalities such as **immediate feedback with errors and warning**, **quick fix features**, **open type features and content assistant.** However, the integrated debugging tool is useless because it does not support Opera browser to debug JavaScript code. Therefore, we use Opera Dragonfly to debug and test the app. The Opera dragonfly will compensate for the drawback of Aptana debugging tool.

  - Specifications and tips

    Similar to Android SDK, we can find a lot of useful specifications and tips about HTML5, CSS3, JavaScript and PHP both on W3C and on Aptana Studio IDE.

  - Interaction with device hardware

    HTML5 provides the <video> element and navigator.getUserMedia() which enables the app to access device's camera. However, only Opera Mobile for Android supports getUserMedia() method to interact with the camera.

    To download the image from the canvas element on our HTML page is very complicated. Whenever the toDataURL() method of the canvas is called, the method must raise a SECURITY_ERR exception. The reason is the same origin policy as discussed in [43]. In order to fix the problem, we have to configure the Opera Mobile on the device as follows. Firstly, we type in **about:config** into the address bar. Then we go to **Security Pref** and select **"Allow Camera To Canvas Copy"**.

    Unlike Android native app, HTML5 support the file dialog very efficiently. We easily work with the file dialog in two lines:

    ```
    <input name="uploadedfile" type="file" />
    <input type="submit" value="Upload File" />
    ```

  - Downloading, installation and configuration

    We can easily download and install the Aptana from Aptana.com and configure Opera Dragonfly on Opera browser.

- SDK	Points
o Applicability	5
o Specifications and tips	5
o Interaction with device's hardware	3
o Downloading, installation and configuration	5
Average points	4.5

- Support from community

The W3C benefits us a lot. We can learn several specifications and features of HTML5, CSS3, jQuery Mobile and JavaScript from W3C page. We also receive full support from several developers' forums.

Ease of developing	Points
- Programming language	5
- SDK	4.5
- Support from community	5
Average points	4.83

Ease of coding

- IDE's capabilities

  We employ Aptana as an IDE for developing the app. The IDE can be downloaded free from Aptana developer's page.

  - Code editor

    It is very easy to code the app with Aptana. The Aptana Studio is capable of generating the code thereby reducing the workload to code the app. Furthermore, we can enjoy the content assistant, immediate feedback with errors and open type features of the IDE to write the code faster.

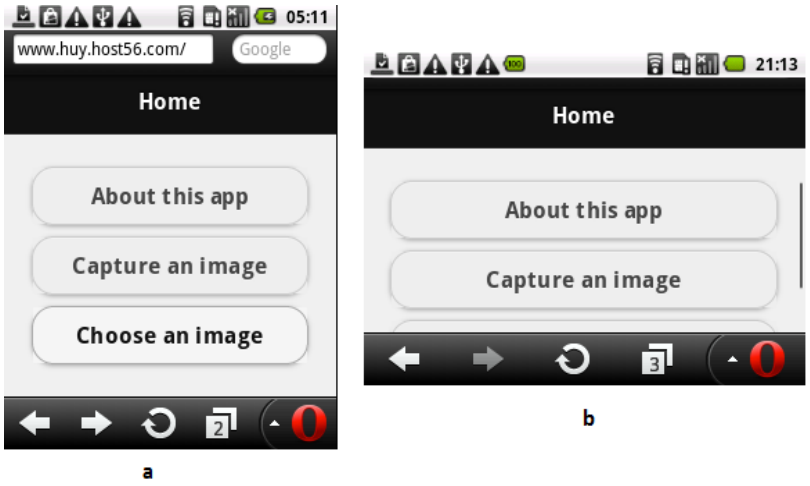


Figure 29: Home user interface of HTML5 mobile app in portrait and landscape modes. We create the user interface by using jQuery Mobile and CSS3

  - User interface builder

    Similar to Eclipse, Aptana does not have drag and drop capability to build user interface. We can only write the code in HTML5 and CSS3 files and view the user interface by running the code on Web browser.

- IDE's capability	Points
o Code editor	5
o UI builder	2
Average points	3.5

- User interface

We employ jQuery Mobile and CSS3 to create our user interface, which is much simpler than creating the native app's user interface in the previous section. jQuery Mobile and CSS3 make our user interface much more adaptive as compared to the native app. We can make the button, text, image and camera display properly regardless of the rotation and the size of the device as in Figure 29.

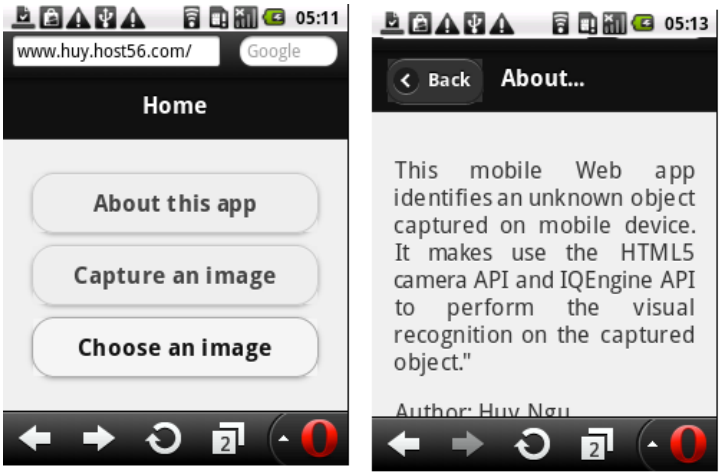


Figure 30: Page in page feature of jQuery Mobile

In order to use jQuery Mobile library, we have to include three files hosted on 000WebHost server in our page: the jQuery Mobile CSS file (jquery.mobile-1.0a1.min.css), the jQuery library (jquery-1.4.3.min.js) and the jQuery Mobile library (jquery.mobile-1.0a1.min.js). Then we can simply employ the user interface elements and features provided by jQuery Mobile to format our page, including **page in page**, **button**, **header**, **content**, **page transition**, **Ajax navigation** and **theming**. The elements and features adopted will result in a smooth experience for the users: the page is loaded faster and the user interface is more attractive. For example, Figure 30 depicts the page in page feature of jQuery Mobile. From the home page, we call the "about" page which stays in the same document with "home" page and we can back to the home page easily from "about" page. This makes user feel like they are working with a native app rather than a mobile Web app. The following code depicts the interface elements and features that we have employed in our page.

```
<div data-role="page" id="home">
        <div data-role="header">
            <h1>Home</h1>
    </div>
    <div data-role="content">
            <a href="#about" data-role="button" data-theme="c" data-transition="pop">
About this app</a>
            </div>
</div>
<div data-role="page" id="about">
        <div data-role="header">
            <h1>About this app</h1>
```

```
    </div>
        <div data-role="content">
    …
    /div>
</div>
```

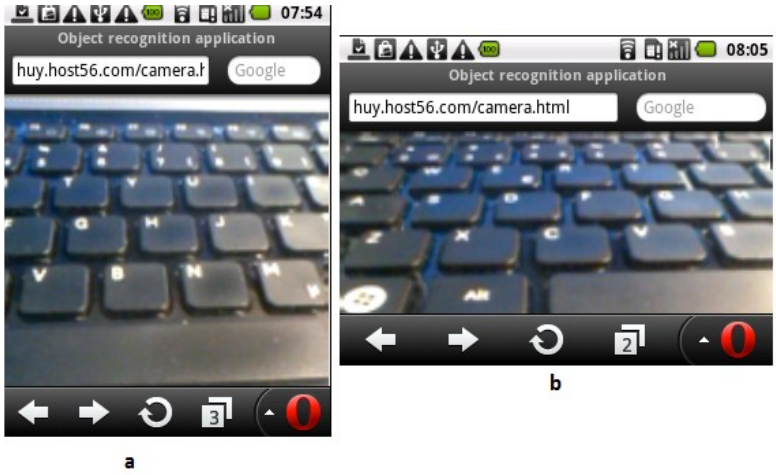


Figure 31: Camera rotation

We also use CSS3 to define our page. The new features we use in our page are o-paged, max-width and min-width. The -o-paged media type enables us to break up the contents of the <html> element into pages that fill 100% of the height of the Web browser. These pages should be navigated in between horizontally by swiping right and left on a touch screen. The media features min-width and max-width help us target different mobile devices with different screen sizes.

```
@media -o-paged {
  html { height: 100%; overflow: -o-paged-x; }
 }
```

- Device‘s interaction
  - Camera

    An advantage of HTML5 paradigm over the native app is that we can make the device’s camera work properly on both portrait and landscape modes as in Figure 31. HTML5 <video> element and navigator.getUsermedia() reduce the verbose of the code to interact with the camera. Firstly, we request a real time video stream of the device’s camera. Then we display the resulting video stream on our page by assigning the stream object returned directly to the video element’s src attribute. After touching the camera interface/video element to capture a picture, we will display the picture by drawing it on a canvas element. To access the device camera and save the image onto the device costs only 14 lines of code as follows.

```
if( navigator.getUserMedia ) {
        video.onclick = function () {
        var oCtx = oCanvas.getContext("2d");
```

```
            oCtx.drawImage(video, 0, 0, 200, 200);
        }
        var success = function ( stream ) {
                video.src = stream;
            };
        var error = function ( err ) {
                msg.innerHTML = "Error: " + err.code;
            };
        navigator.getUserMedia('video', success, error);
```

- Downloading and uploading an image

  We can download the canvas image onto our device's storage by touching the image. The code is as follows.

  ```
  var strData = oCanvas.toDataURL();
  document.location.href=strData.replace('image/png', 'image/octet-stream');
  ```

  However, we spend a great amount of time to deal with the same origin policy because such policy prevents the toDataURL() method from running on Opera Mobile on Android phone. The browser by default disables the access to any Stream pixel data via HTML5 <canvas> element. Therefore, we have to override the restriction by choosing to allow the canvas copy as discussed in the SDK criterion.

  Unlike the Android SDK, HTML support file dialog to select a file from the device's storage very effectively. We simply use <input> tag on HTML5 page to open the file dialog and select the wanted image.

  ```
  <input type="hidden" name="MAX_FILE_SIZE" value="1000000" />
  <input name="uploadedfile" type="file" data-theme="c" />
  <input type="submit" value="Upload File" data-theme="c" />
  ```

- Internet connection

  The app can surely access the Internet. We do not have to set any configuration to make it connect to the Internet.

- Native app interaction (Camera and photo gallery)

  Our HTML5 mobile app cannot make use the built-in camera and the native photo gallery apps. Actually, we can take advantage of the built-in camera app when we select a file by using the <input> tag as follows

  ```
  <input type="file" accept=”image/*;capture=camera” />
  ```

  The device will open the native camera, allowing people to take a picture and use the picture as an input. It is an excellent solution for employing device's built-in camera app but it only works in Android 3.0 and later versions. Therefore, we do not employ the solution in our app.

- Device interaction	Points

o Camera	5
o Downloading and uploading an image	4
o Internet connection	5
o Native app	1
Average points	3.75

- IQ Engine server interaction

  We receive full support from PHP library and API to write a PHP script, and host the script on 000WebHost server. The script is responsible for sending the request and decoding the JSON response from the IQEngine server.

  o Getting timestamp

    Getting the current timestamp is straightforward which costs 1 line of PHP code

    ```
    $timestamp = date(' YYYYmmDDHHMMSS ')
    ```

  o Building signature

    Computing signature by using PHP script is much less verbose than using Android SDK. We only need to use the hash_mac function with API key, filename, timestamp and API secret as the input parameters.

    ```
    $api_sig = hash_hmac("sha1",$temp_api_sig,$api_secret,false);
    ```

  o Sending request and retrieving result

    In order to send the image and other parameters to IQEngine server, we use cURL file transfer tool. The advantage of cURL is that it is straightforward to use and it works effectively with IQEngine API. The following code makes request to IQEngine API and retrieve the response from the server.

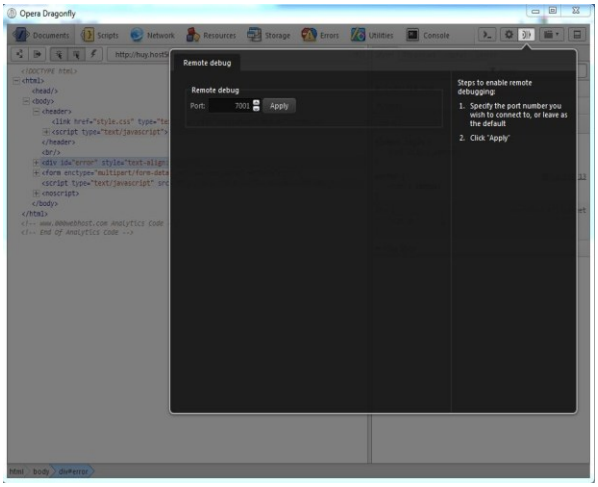
    ```
    $ch = curl_init($url);
    curl_setopt($ch, CURLOPT_USERAGENT, 'Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.1) Gecko/20061204 Firefox/2.0.0.1');
    curl_setopt($ch, CURLOPT_RETURNTRANSFER  ,true);
    curl_setopt($ch, CURLOPT_BINARYTRANSFER, true);
    curl_setopt($ch, CURLOPT_CONNECTTIMEOUT, 2);
    curl_setopt($ch, CURLOPT_POST, true);
    curl_setopt($ch, CURLOPT_POSTFIELDS, $fields);
    curl_setopt($ch, CURLOPT_HEADER , false);
    $response = curl_exec($ch);
    ```

    The biggest drawback of coding the app using the paradigms is that we cannot deliver the captured image directly to the IQEngine server. Instead, we must always select to send an image available on the device's storage to 000WebHost server and subsequently use the PHP script to make IQEngine API request.

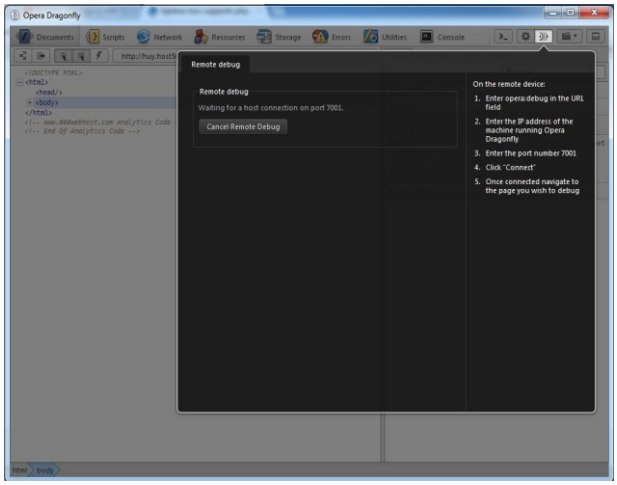
- IQEngine server interaction	Points
o Getting time stamp	5
o Building signature	5

○ Sending request and retrieving result	4
Average points	4.67

Ease of coding	Points
- IDE's capability	3.5
- User interface	5
- Device interaction	3.75
- IQEngine server interaction	4.67
Average points	4.23



a



b

Figures 32a and 32b: Configuring the debugging tool at development environment

Figure 33: Configuring the phone for debugging purpose

Ease of debugging

- Debugging client side code (HTML5, CSS3, and JavaScript)

  We cannot debug the code with Aptana Visual Studio because the tool does not support the debugging with Opera Mobile. Instead, we employ Opera Dragonfly for remote debugging. The remote debugging means that we debug the HTML5, JavaScript and CSS files running on mobile devices from the development environment (our laptop). Opera Dragonfly, which is integrated into Opera browser on desktop environment, performs the tasks very effectively. We can connect the device and the laptop to debug the app in some steps as follows.

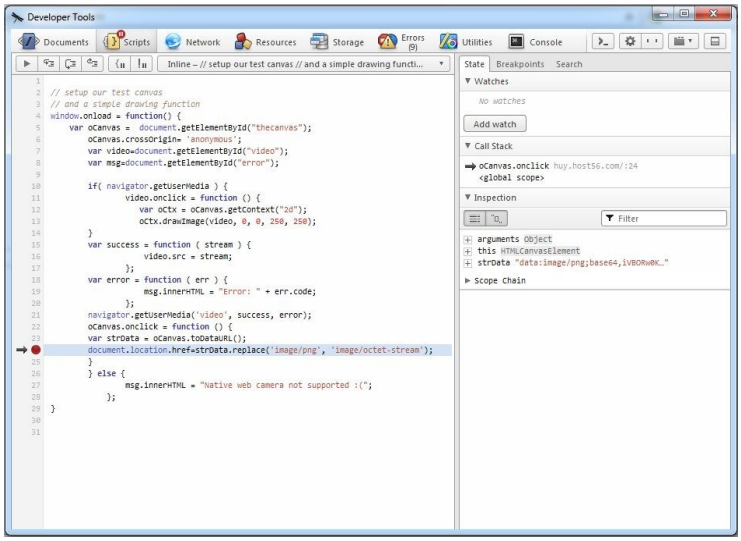


Figure 34: Debugging the app by using Opera Dragonfly.

From our laptop, we simply start Opera Dragonfly by right clicking on the browser and selecting “inspect element”. Then we choose “Remote Debugging Configuration” and declare the port as shown in Figures 32a and 32b.

From our device, we enter **opera:debug** in the URL field and the IP Address of the machine running Opera Dragonfy (our laptop’s IP address). We subsequently enter the port number and click “Connect” as in Figure 33.

After connecting the device to our laptop, we can simply manage the breakpoints (e.g. create, delete and move between the breakpoints) and debug the app running on our device as displayed in Figure 34.

Points: 5

- Debugging server side code (PHP)

  We find it impossible to debug the PHP code running on 000WebHost because debugging the code concerns several devices such as our laptop, Android phone, 000WebHost server and IQEngine server. Some tools (e.g. Aptana and XCode) can help debug PHP code but do not support the debugging of the app running on mobile devices. On the other hand, Opera Dragonfly supports the debugging mobile apps but cannot be used for debugging server side code.

Ease of debugging	Points
- Debugging client side code	5
- Debugging server side code	1
Average points	3

Ease of testing

- Emulator

  The emulator is inefficient to test the app because it does not support testing device's camera and storage.

- Real device

  We can test the app easily by using our real Android devices that have Opera Mobile installed beforehand. We open the browser and enter the URL to load the app. Currently, we already test the app with Samsung Galaxy GTI-5500 with Android 2.1 and HTC with Android HTC Sense.

Ease of testing	Points
- Emulator	1
- Real device	5
Average points	3

Ease of deploying and updating

There is much more work to deploy the HTML5 mobile app on a server than to deploy the native app on an Android device. We have to rent the hosting service of 000WebHost and upload all the files of the app including HTML5 file, CSS file and PHP scripts. We also have to configure the security parameters of such files (e.g. read, write and so on). The access control on the app should also be considered when deploying the app on the server. The app then can be deployed to user's devices via the Internet.

The update of the app is simpler than the deployment. 000WebHost provides us with a useful code editor and several specifications and tips, enabling us to edit the files and update the app conveniently. Users can subsequently retrieve the newest version of the app via their Web browser without the need to install and update any component of the app.

Ease of deploying and updating	Points
Average points	2

Ease of distributing

- Compatibility

Only Opera Mobile on Android supports getUserMedia() method. Therefore, the app can only run on Opera Mobile installed on Android phone.

- Without app store

The app can be distributed easily on the Internet. Users can load the app and run it on their device very conveniently without installing it.

- With app store

To distribute the app via Android marketplace costs $25 and we have to go through some steps to have the app checked by the marketplace.

Ease of distributing	Points
- Compatibility	1
- Without app store	5
- With app store	3
Average points	3

Application types

- Application using device's capabilities

The app is capable of working with the device's camera, keypad and storage. The camera works very effectively in both portrait and landscape mode, which is an advantage over the native app. The app can also interact with device's storage via the file dialog that is supported by HTML.

- Application using server's capabilities

The HTML5 mobile app makes use the capabilities of 000WebHost and IQEngine servers, such as storage, analysing and identifying capabilities.

Application types	Points
- Application using device's capabilities	5
- Application using server's capabilities	5
Average points	5

Powerful API and libraries

The app uses the libraries and APIs of HTML5 (e.g. canvas and video), jQuery, PHP (e.g. cURL, move_uploaded_file, timestamp) and IQEngine server.

Powerful APIs and libraries	Points
Average points	5

Figures 35a and 35b: Capturing an image by using the HTML5 app.

Payment possibilities

We can sell the app via app store, or publish the app onto a Web server and create the access control by ourselves to get paid from users. In the former, we have to pay the registration fee and share the revenue with the app store. In the latter, we only have to pay the fee for the hosting service.

Payment possibilities	Points
Average points	5

The table below summarizes the evaluation of HTML5 mobile app on developer viewpoint after the practical implementation

Table 20: Summary of the evaluation on the HTML5 mobile app from developer viewpoint

Ease of developing	4.83
Ease of coding	4.23
Ease of debugging	3
Ease of testing	3
Ease of deploying and updating	3
Ease of distributing	3
Application types	5
Powerful APIs and libraries	5
Payment possibility	5
Average points	*4.01*

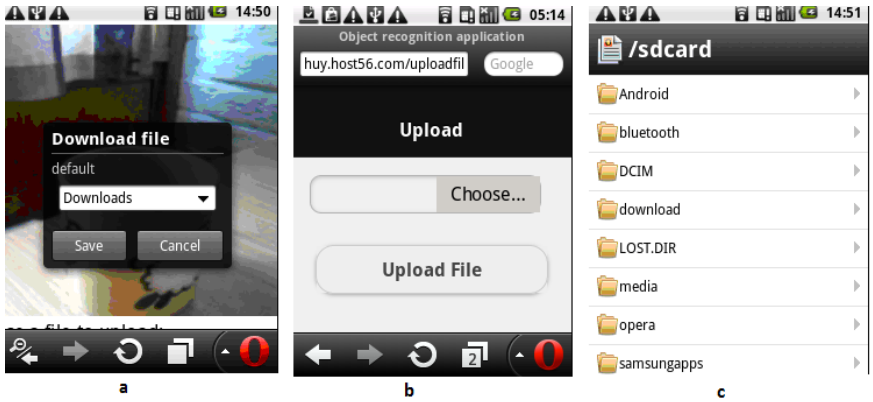
- **User viewpoint**

Ease of use

- Performance

  The app is lightweight (117kb) to run quite fast on our Android phone. It takes about 1 second to open Opera browser, 2 to 3 seconds to enter the URL. The time to load the app on the Opera browser is within 600 and 2100 milliseconds, including loading the camera, button and file dialog. Meanwhile, it takes about 1300 milliseconds to download the image and 2100 milliseconds to upload the image onto 000WebHost

server and call IQEngine API. The amount of time for analysing and labelling the image is about 25 seconds. The advantage of HTML5 mobile app over the native app is that the camera does not lag when we capture a picture of wanted object.



Figures 36a, 36b and 36c: Downloading the image onto device's storage and selecting an image from the device's storage

- User interface

The app has a simple but intuitive user interface, including camera interface, image, text and button. Users can see the hint by touching the camera and the image and rotate the device without any viewing difficulty. They can also zoom in and zoom out the page (Figure 35), and interact with the file dialog very easily. The HTML5 mobile app will look like a native app if there is no address bar and Opera trademark. However, because we need time to load the app onto the browser, the user interface lags sometime.

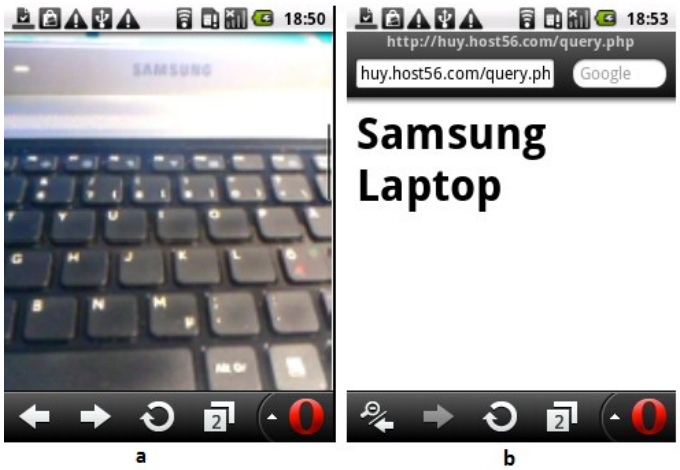


Figure 37: Object recognition capability of the HTML5 mobile app

- Operation

Similar to the native app, users can choose the wanted task on their home screen. They can capture an image by tapping the camera interface, save it onto the device by touching the image, choose to upload the wanted image to 000WebHost server and call the IQEngine API to get the image labelled. They can also select and upload an image available on their device's storage to 000WebHost to make IQEngine API

request. One of the drawbacks of the app is that users must open their mobile browser and load the app whenever they want to access it. It is very inconvenient and time consuming. Furthermore, users must download the image and send it back to 000WebHost server as described in Figures 36a, 36b and 36c, instead of uploading the image as soon as capturing it. Another disadvantage is that users must go to **Security Pref** and select"**Allow Camera To Canvas Copy"** on their Opera Mobile browser to enable the image download onto their device.

Ease of use	Points
- Performance	3
- User interface	4
- Operation	2
Average points	3

Functionality

- Working offline

The app cannot work offline. It requires an Internet connection to load the app to Opera Mobile browser.

- Accessing device's hardware

The app can access the device's hardware very efficiently. The camera and the touch screen work great. The app can also save an image onto device's storage and upload it later on.

- Object recognition

Our app can only label the object such as car, building, cup, camera and so on. The app is not capable of face recognition. Figure 37 depicts the recognition capability of the app. Figures 37a and 37b show the captured image and the label sent from the server respectively.

Functionality	Points
- Working offline	1
- Access device's hardware	5
- Using real time communication	4
Average points	3.33

Installation and update

- Compatibility

The app can only run on Opera Mobile for Android.

- Downloading and installing

Users do not have to download and install the app. They can open their Opera Mobile browser and enter the URL to get the app loaded onto their device.

Installation and update	Points
- Compatibility	1
- Downloading and installing	5

Average points	3

The table below summarizes the evaluation of HTML5 mobile app paradigm on user viewpoint after the practical implementation

Table 21: Summary of the evaluation on the HTML5 mobile app from user viewpoint

Ease of use	3
Functionality	3.33
Installation and update	3
Average points	3.11

5.3. PhoneGap application

The paradigm we employ in this section to build object recognition app is also HTML5 mobile app. We still create the app by using HTML5, CSS3 and JavaScript but we do not use <video> and getUserMedia() to access device's camera. Instead, we wrap the app with PhoneGap to enable it access the native APIs (e.g. camera, storage and photo gallery). In the other words, the PhoneGap use the standard Web technologies to bridge HTML5 Web application and our mobile phone. Therefore, we can write the app in HTML5, CSS3 and JavaScript, access native features and deploy the app to multiple platforms (Android, iOS, WP7, Symbian and WebOS). We name this app as PhoneGap app to distinguish with the HTML5 mobile app discussed in the previous section

In order to develop the app, we use **Eclipse** as the IDE and **PhoneGap** as the SDK. We also use 000WebHost to host the PHP script to interact with IQEngine server.

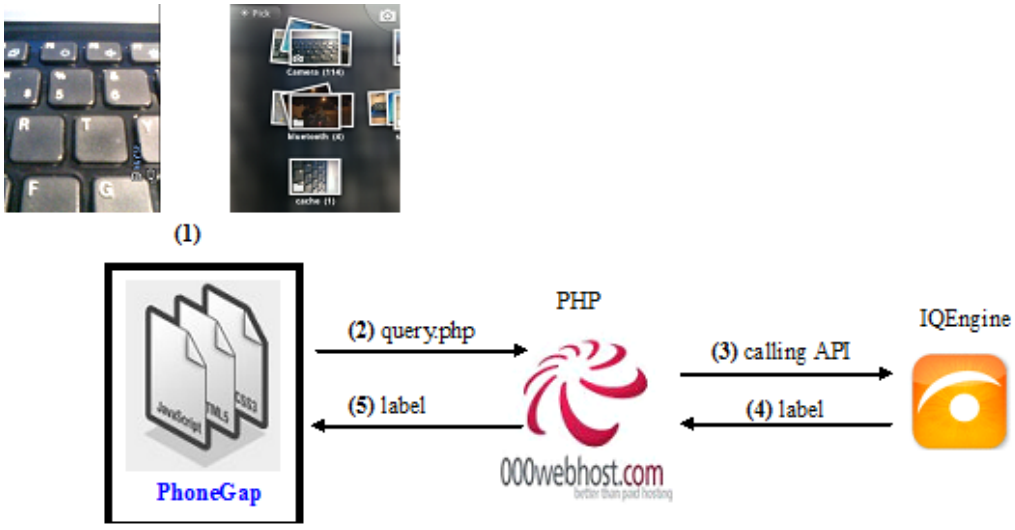


Figure 38: PhoneGap app architecture

Similar to the HTML5 mobile app, the PhoneGap app allows its users to capture a picture, send the picture to IQEngine server and get the label back. The app also takes advantage of 000WebHost to host PHP script which makes IQEngine API requests. The difference between the two is that HTML5, JavaScript and CSS3 files of the PhoneGap app stay on mobile device instead of being hosted on 000WebHost server. Therefore, the PhoneGap app works within mobile browser but its UI controls are loaded much faster

than HTML5 mobile app. Another difference is that the app does not use <video> tag and camera stream of HTML5 but camera API of PhoneGap to access device's camera.

5.3.1. Phonegap app architecture

The app includes HTML5, CSS3 and JavaScript files which stay on mobile device, and PHP script which is hosted on 000WebHost server. PhoneGap is the most significant component of the app because it provides us with very useful JavaScript library and API to get access to native APIs (Camera, photo gallery and network). The CSS3 will define the page with a range of media types. Figure 38 describes the app's architecture. From the home user interface, users capture a picture or select an available picture in their photo gallery (**1**). Then they call PHP script which is responsible for uploading the captured/chosen image onto 000WebHost server (**2**) and calling IQEngine API to recognize the object (**3**). IQEngine server then sends the result to the device (**4**) (**5**).

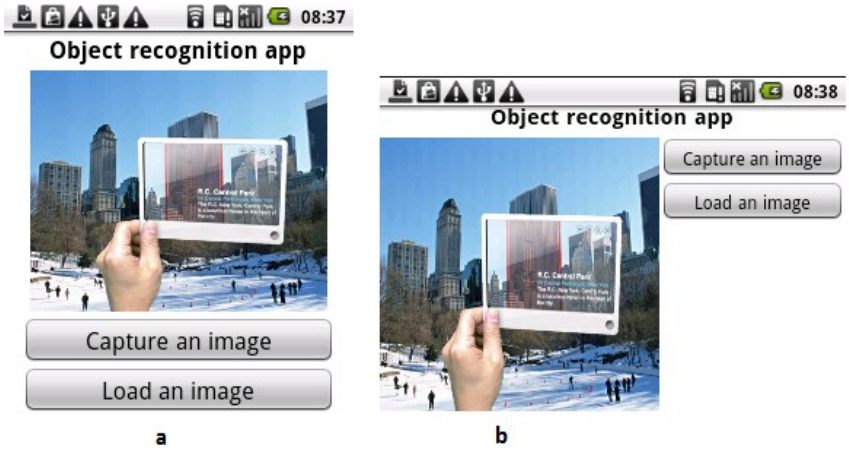


Figure 39: Home user interface of the PhoneGap app in portrait and landscape modes

5.3.2. Analysis and evaluation

- **Developer viewpoint**

Ease of developing

- Programming language

  HTML5, JavaScript, CSS3 and PHP for coding the app are not verbose and complicated. We can handle with the language easily.

- Software development kit

  - Applicability

    We receive full support from the SDK. We can write the code and test the app very effectively by using Eclipse IDE, and deploy the app onto multiple platforms and to app store conveniently by using the PhoneGap Build. However, there is no JavaScript debugging/ compiling support provided by PhoneGap SDK.

  - Specifications and tips

    We find a lot of handy specifications and tips from PhoneGap developer page [44].

- Interaction with device hardware

  PhoneGap enables us to access device's native features more effectively than any SDKs. We can create the camera object and make use camera.getPicture to take a photo or to get an available photo from photo gallery very easily. We can also simply access the files and folders on device's storage. Moreover, PhoneGap app is more advantageous than HTML5 mobile app in the way that it can send the captured picture to 000WebHost without the need to download the captured image, open file dialog and choose the image on device's storage. The app can also access the Internet and check the network connection easily.

- Downloading, installing and configuring

  We download PhoneGap free on PhoneGap developer Website. It is also easy and timesaving to install the PhoneGap on our laptop. We can choose Android which is our favorite platform and developing tool to build the app. The other developers who are familiar with the other platforms (e.g. iOS, Blackberry, Windows Phone, and Web OS) can also choose these platforms and the corresponding tools to create the app. However, the SDK's configuration requires several steps as described in [45].

- SDK	Points
o Applicability	4
o Specifications and tips	5
o Interaction with device's hardware	5
o Downloading, installation and configuring	4
Average points	4.5

- Support from community

  We receive full support from growing community such as tutorials, tools and troubleshooting to build the app. The forum we always access to find out the solution for our problem is StackOverFlow forum.

Ease of developing	Points
- Programming language	5
- SDK	4.5
- Support from community	5
Average points	4.83

Ease of coding

- IDE's capabilities

  We employ Eclipse as the IDE to develop the app. Eclipse bridges the PhoneGap SDK and our Web app, and help us manage the files, folders and packages effectively.

- Code editor

  The code editor integrated in Eclipse is inefficient to use. We can only write the code without any support on instant feedback with error and warning, import structure, open type feature, content assistant and quick fix features.

- User interface builder

  User interface builder is also inefficient to create user interface. It does not have dragging and dropping capability. Instead, we have to write the code in HTML file and CSS to define the user interface. Moreover, we can only test the user interface by right clicking on the HTML file and opening it on browser.

- IDE's capability	Points
○ Code editor	2
○ UI builder	2
Average points	2

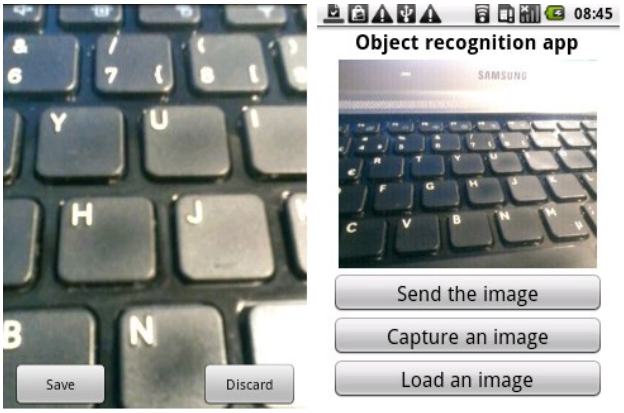


Figure 40: Capturing a picture by using device's camera via PhoneGap

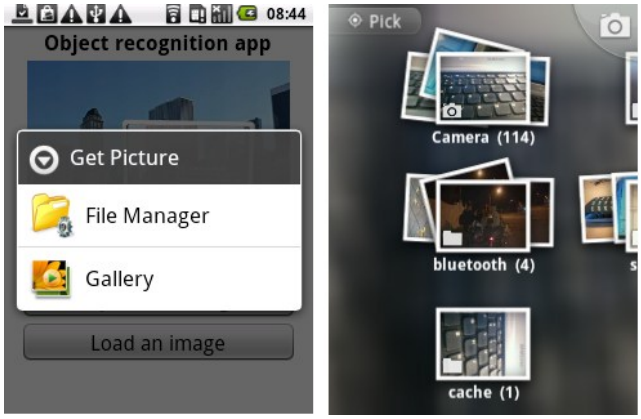


Figure 41: Selecting an image from photo gallery

- User interface

  We can make use CSS3's new features to create the user interface effectively. The features adopted to define the page are media screen, max-width, min-width and orientation. The max-width and min-width make the app target different mobile devices which have different screen widths. Orientation helps the app detect the rotation of the device (portrait or landscape) to rearrange the UI controls (button, textview and image) as efficiently as possible. Therefore, we can easily build an adaptive user interface which enhances the user experience. Formatting the user

interface of PhoneGap app is as straightforward as the HTML5 mobile app. The CSS is declared as follows

```
@media screen and (max-width: 400px) and (orientation: portrait){
   body {width:100%; margin:0px}
   .Button {width: 90% ; height:10%; margin-bottom: 0px }
   .Image {width:85% ; height:55%}
   .Header{ text-align: center; padding: 4px; font-weight:900;}
}
@media screen and (max-width: 400px) and (orientation: landscape){
   body {width:100%;height:100%; margin:0px}
   .Button {width: 40%; height:15% ;padding: 1px ;font-size:12px;}
   .Image {float: left; width:60% ; height:85%}
   .Header {text-align: center; font-weight:900;font-size:15px; margin-bottom: 5px}
}
@media screen and (min-width: 400px) {
   body  {width:100%;height:100%; margin:0px}
   .Button {width: 40% ; height:15% ;padding: 1px ;font-size:20px;}
   .Image {float: left; width:60% ; height:82%}
   .Header {text-align: center; font-weight: 900;font-size:25px; margin-bottom: 15px}
}
```

The first CSS class (max-width: 400) can target the mobile screen in portrait mode. The second CSS class can be adopted for the mobile phones but in landscape mode. The third class can target the devices which have the screen width over 400px (e.g. laptop and ipad). The float: left property can float the image to the left with the other UI controls wrapping it when the device's screen changes to landscape mode. Figure 39 shows the home user interface in both portrait and landscape modes. The image floats left in landscape mode, and the buttons keep their format and wrap the image.

Points: 5

- Device's interaction
  - Camera

    Getting access to device's camera is very straightforward. We make use the navigator.camera.getPicture(cameraSucess, cameraError [,cameraOption]) to capture a picture or to load a picture from photo gallery. cameraSuccess works as the trigger to fire when we retrieve the photo successfully. If Camera.sourceType is set to Camera.PictureSourceType.CAMERA (which is default), the camera.getPicture opens the device's default camera app so that we can take a picture. Once the photo is taken, the camera app closes and our app is restored. On the other hand, if Camera.sourceType is set to Camera.PictureSourceType.PHOTOLIBRARY, a photo chooser dialog is shown, from which a photo from the album can be selected. It

costs only 6 lines of code to access our device's camera by using PhoneGap. The code is as follows.

```
function capturePhotoWithFile() {
      navigator.camera.getPicture(onPhotoCaptureSuccess, onFail, { quality: 50,
destinationType: Camera.DestinationType.FILE_URI });
  }
 function onPhotoCaptureSuccess(imageData) {
     image.src = imageData;
  }
```

PhoneGap will save the captured picture as soon as we choose "save" to save it as in Figure 40. The greatest advantage is that we do not have to write any code to save the picture onto our devices. PhoneGap will handle with the task automatically.

Similarly, the following code deals with chosen images from photo gallery (Figure 41).

```
function getPhoto() {
    navigator.camera.getPicture(onPhotoURISuccess, onFail, { quality: 50,
destinationType: destinationType.FILE_URI,
      sourceType: pictureSource.PHOTOLIBRARY });
  }
function onPhotoURISuccess(imageURI) {
     image.src = imageURI;
 }
```

- Downloading and uploading an image

  We do not have to write the code to download captured image because PhoneGap will store the image itself. PhoneGap also supports uploading the image from photo gallery very effectively. We upload the image to 000WebHost by using FileTransfer class of PhoneGap and PHP script on 000WebHost server.

```
var ft = new FileTransfer();
ft.upload(imageURI, "http://huy.host56.com/query.php", success, fail, options);
```

- Internet connection

  Unlike HTML5 mobile Web app, in order to make the PhoneGap app connect to the Internet, we have to add the Internet access permission into manifest.xml file. Then the app can access the Internet seamlessly.

```
<uses-permission android:name="android.permission.INTERNET" />
```

- Native apps (camera and photo gallery)

  PhoneGap enable us to access the built-in apps such as camera and photo gallery. Therefore, we need only to write the code to make use the apps instead of building them ourselves.

- Device interaction	Points
o Camera	5
o Downloading and uploading an image	5
o Internet connection	5
o Native apps	5
Average points	5

- IQEngine server interaction

  We keep the PHP script of HTML5 mobile app to make PhoneGap app interact with IQEngine server. The script is also hosted on 000WebHost. Therefore, the analysis and evaluation in this criterion are the same as HTML5 mobile app.

- IQEngine server interaction	Points
Average points	4.67

Ease of coding	Points
- IDE's capability	2
- User interface	5
- Device interaction	5
- IQEngine server interaction	4.67
Average points	4.17

Ease of debugging

- Debugging client side code (HTML5, CCS3 and JavaScript)

  Unfortunately, PhoneGap does not support debugging JavaScript code. We cannot find any solution to debug PhoneGap app. We can figure out bugs and exceptions during the development by using `adb logcat`. However, it cannot replace a debugging tool such as Opera Dragonfly and Xcode since it is unable to trace the value of JavaScript objects.

- Debugging server side code (PHP script)

  Similar to the HTML5 mobile app, we cannot debug the PHP script staying on 000WebHost because the script concerns too many devices: our laptop, mobile phone, 000WebHost server and IQEngine server.

Ease of debugging	Points
- Debugging client side code	2
- Debugging server side code	1
Average points	1.5

Ease of testing

- Emulator

  The emulator is useless to test the app because it does not support device's camera.

- Real device

The ease of testing the PhoneGap app on physical device depends on the platform to develop the app. Therefore, we have to consider the testing on Android, iOS and Windows Phone to guarantee the accuracy of the evaluation on this criterion. However, since we develop the PhoneGap app for Android platform only, we will use the result from the evaluation of testing on iOS and Windows Phone in chapter 4- Analysis and evaluation.

- Android

  It is simple to test the app with a real device. We only need to connect our Android device to the laptop, set debuggable to true and run the app. We test the app similar to the native app, which is discussed in chapter 4. While we are testing, we will check the logcat file or use adb tool to figure out the exceptions that make the app stops unexpectedly.

- iOS

  In chapter IV, we give 2 to the testing on iOS platform.

- Windows Phone 7

  In chapter IV, we give 2 to the testing on Windows Phone platform.

Testing on real device	Points
- Android	5
- iOS	2
- Windows Phone 7	2
Average points	*3*

Ease of testing	Points
- Emulator	1
- Real device	3
Average points	*2*

Ease of deploying and updating

Similar to the testing on physical device, we have to consider 3 different platforms (Android, iOS and Windows Phone) on this criterion and make use the result from the chapter IV.

- Android

Because the PhoneGap app works like a native app on mobile platform, **the deploying and updating of the PhoneGap app is the same as the native app**. Therefore, we can easily connect the Android device to our laptop. Then we choose to install the app from unknown source on our device. Later on, we test the app from our laptop, thereby deploying or updating the app automatically to our device.

- iOS

  iOS has 3 points in deploying and updating the app criterion.

- Windows Phone

  We give Windows Phone 2 points to deploy and update the app.

Ease of deploying and updating	Points
- Android	5
- iOS	3
- Windows Phone 7	2
Average points	*3.33*

Ease of distributing

- Compatibility

  We can upload the PhoneGap app to PhoneGap Build service [46] and get back a market-place ready app for iOS, Android and Windows Phone or even more. By compiling on the cloud with PhoneGap, we get all benefits of cross-platform development but we can still build the app on our favorite platform (Android). However, we must pay fee to use PhoneGap Build for compiling the private app. Moreover, the app distribution depends on the platform we are targeting. For Web OS and Symbian, the app is ready for submission and distribution. On the contrary, for iOS and Windows Phone, we have to provide the correct certificates and/or signing keys to allow distribution, which cost $100 or even more.

- With app store

  - Android

    In order to publish our app on Android marketplace, we have to fulfill many requirements to make the app qualified enough and pay $25 for registration fee. The marketplace then will manage the concerns of the app distribution such as paying, selling, and security.

  - iOS

  - Windows Phone 7

Distributing with app store	Points
- Android	4
- iOS	4
- Windows Phone 7	3
Average points	*3.67*

- Without app store

  - Android

We can easily distribute the PhoneGap app by copying the .apk file onto user's mobile phone. There are many ways to copy the file such as via email and Dropbox sharing.

  - iOS

  - Windows Phone

Distributing without app store	Points
- Android	5
- iOS	3
- Windows Phone 7	1

Average points	*3*

Ease of distributing	Points
- Compatibility	4
- Without app store	3.67
- With app store	3
Average points	*3.56*

Application type

- Application using device's capabilities

  The app can take advantage of device's hardware feature such as camera, keypad, storage and WLAN connection. The app also makes use the built-in camera app and photo gallery app to make it work the same as a native app.

- Application using server's capabilities

  The app makes use the IQEngine server for the recognition (processing and training database) and 000WebHost server for hosting (storage).

Application types	Points
- Application using device's capabilities	5
- Application using server's capabilities	5
Average points	*5*

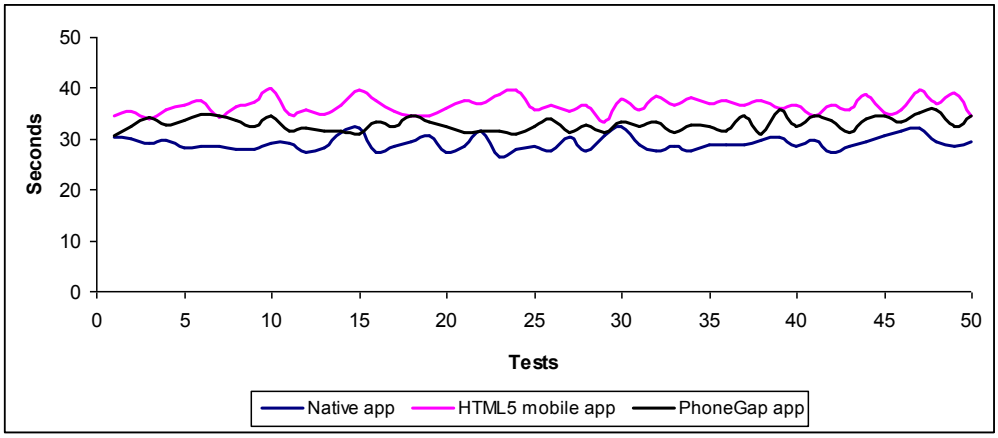


Figure 42: The amount of time to use the native app, HTML5 mobile app and PhoneGap app in comparison. We perform the tests by using the Samsung Galaxy GTI-5500 with Android 2.1. We create three timers and integrate them into the apps to measure the time needed to use the apps in milliseconds. The timer created by using Android Java code is for calculating the time using the native app and the others are in JavaScript to measure the HTML5 mobile app and the PhoneGap app. We get the system time of staring the apps and the system time of receiving the result from the IQEngine server for each test. Then, the time to use the app is calculated by the subtraction of the starting time from the finishing time.

Powerful libraries and APIs

PhoneGap's library is very powerful, reducing our workload to develop the app. Additionally, the app also uses IQEngine API to recognize the image.

Powerful APIs and libraries	Points
Average points	*5*

Payment possibilities

Since PhoneGap app can target different mobile platforms, the paradigm has more payment possibility than the native app paradigm. We can earn our revenue by selling the app via marketplace. We can also create ourselves Website to sell the app (for Android only). Users can download and install it on their device and pay via the Website.

Payment possibilities	Points
Average points	*5*

The table below summarizes the evaluation of PhoneGap mobile app on developer viewpoint after the practical verification

Table 22: Summary of the evaluation on the PhoneGap app from developer viewpoint

Ease of developing	4.83
Ease of coding	4.17
Ease of debugging	1.5
Ease of testing	2
Ease of deploying and updating	3.33
Ease of distributing	3.56
Application types	5
Powerful APIs and libraries	5
Payment possibility	5
Average points	*3.82*

- **User viewpoint**

Ease of use

- Performance

  The app works very efficiently on Samsung Galaxy GTI-5500 with Android 2.1 and HTC Desire with Android HTC Sense. Similar to the native app and the HTML5 mobile app, we run 50 tests to measure the time to start the app, capture an image, send the captured image to the server and get the label back. The time to start the app varies from 400 to 450 milliseconds. Meanwhile the time needed to start the camera is within 2 and 4 seconds. It also takes about 1400 milliseconds to capture an image, 1500 milliseconds to send the image to 000WebHost and to make IQEngine API request, and 25 seconds to get the result back from IQEngine server. Figure 42 shows the time in seconds needed to use the native app, HTML5 mobile app and PhoneGap app, from starting the app to receiving the result from the server. The native app requires the least amount of time to use. Meanwhile, it takes longer to use the HTML5 mobile app than to use the native app and the PhoneGap app. The reason is that we must run Opera browser and load the app onto the browser before using it. PhoneGap app takes longer to use than the native app because it takes much more time to start the camera and capture a picture.

- User interface

The PhoneGap app has a simple and intuitive user interface, including button, image and text. The built-in camera app which is called by the PhoneGap app works really effective. An advantage over the native app is that the camera can work both in portrait and landscape mode. Moreover, the photo gallery has amazing look and works great (Figure 41). We can easily choose the image to upload to the server. Additionally, the user interface does not lag when we rotate the device.

- Operation

We found it easy and interesting to use the app. Like using a native app, we start the PhoneGap app by clicking on the app icon on home screen. Then we capture an image by using built-in camera (Figure 40) or select an available image on photo gallery (Figure 41). We subsequently send the image for the recognition without any difficulty. Unlike the HTML5 mobile app, we do not have to download captured images onto device because the image will be restored automatically right after we capture it. After a few seconds, we will receive the result from server via alert message box.

Ease of use	Points
- Performance	4
- User interface	5
- Operation	5
Average points	*4.67*

Functionality

- Working offline

The app needs an Internet access to send captured images to server. However, unlike HTML5 mobile app, the camera works independently with the Internet. Therefore, we can still capture an image by using the camera and send the captured image to the server when we have Internet access.

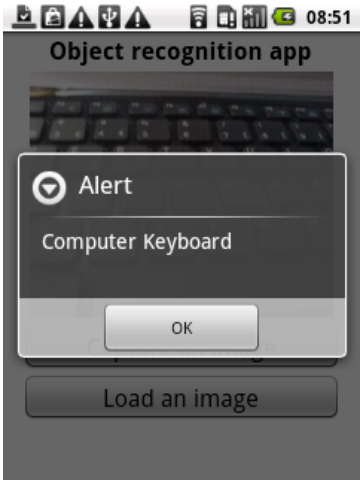


Figure 43: Object recognition capability of the PhoneGap app

- Accessing device hardware

The app can access device's camera and storage. It can also make use the built-in camera app and photo gallery to improve user experience.

- Object recognition

  The app can only recognize the objects such as computer, car, cake and character. It does not have face recognition capability. Figure 43 depicts the object recognition capability of the app. We capture a picture of our laptop keyboard and the app is able to identify it after a few seconds.

Functionality	Points
- Working offline	3
- Access device's hardware	5
- Using real time communication	4
Average points	*4*

Installation and update

- Compatibility

  Even though we have chance to find many versions of the app targeting different platforms on marketplaces, the app itself is not cross-platform. It means that we should find the most suitable version to install on the right platform on our device. For example, if we have two devices with two different platforms (e.g. iOS and Android), we still have to download and install the app targeting iOS from Apple app store and the other app targeting Android from Android marketplace.

- Downloading and installing

  We have to access the Internet, download the app from app store and install it on our mobile phone.

Installation and update	Points
- Compatibility	2
- Downloading and installing	4
Average points	*3*

The table below summarizes the evaluation of PhoneGap mobile app on user viewpoint after the practical verification

Table 23: Summary of the evaluation on the PhoneGap app from user viewpoint

Ease of use	4.67
Functionality	4
Installation and update	3
Average points	*3.89*

After the analysis and evaluation of the HTML5 mobile app and PhoneGap app, we find the points for HTML5 mobile app paradigm by calculating the average points of the two apps. The table below summarizes the evaluation of HTML5 mobile app paradigm after all.

Table 24: Summary of the evaluation on the HTML5 mobile app paradigm

	Developer viewpoint	User viewpoint
HTML5 mobile app	4.01	3.11
PhoneGap app	3.82	3.89

HTML5 mobile app paradigm	3.92	3.5

The table below summarizes the evaluation of native app paradigm and HTML5 mobile app paradigm after the practical verification.

Table 25: Summary of the evaluation on the native app and HTML5 mobile app paradigms

	Developer viewpoint	User viewpoint
HTML5 mobile app paradigm	3.92	3.5
Native app	3.73	3.61

Chapter 6: Conclusion

6.1. Discussion

Mobile phones are no longer for only calling and messaging, but provide users with many interesting apps that ease their life. To an average mobile user, the mobile app paradigms discussed in this thesis are functionally equivalent. In other words, the paradigms can meet very basic need of mobile users. However, each paradigm has its own strong points, thereby meeting the demands of mobile developers, mobile users and service/content providers in its own way.

For developers**, native apps** and **HTML5** are the best selection to build mobile app. If developers want to make an app that requires accelerated graphic processing (e.g. high-end gaming apps), they should develop a native app. Only in that way can the app truly tap in processing powers and hardware features of the device. On the other hand, for more straightforward content driven service apps, HTML5 is preferred. It introduces many new advance features to ease the development of mobile apps. W3C and other third parties are developing the new APIs and libraries to make HTML5 more powerful and seamlessly capable of interacting with mobile devices in the same manner in which native apps do. **Widgets** are valuable for developers when they want to make a lightweight, single functional and portable app on mobile phone. **Mobile Web app paradigm** obtains the lowest grade because it is very complicated to build a functional and robust mobile Web app. The result from the **practical verification** is same that from the analysis and evaluation. Developers prefer HTML5 mobile app to native app paradigm even though there is sufficient support to interact with device's hardware features (camera, storage and network access) from both of the paradigms. The reason is that the HTML5/ PhoneGap code to access device's hardware and Web service is less verbose and complicated than Android Java code. Building an adaptive user interface on HTML5 mobile apps is also much more straightforward than on native apps. Furthermore, the capability of **"write once and deploy many"** is an advantage of HTML5/PhoneGap over the native app paradigm.

For mobile user, **native apps** have the highest performance and usability. They are also very responsive and functional, which create excellent user experience. Therefore, native apps still get attention of worldwide users and keep the first position among mobile app paradigms for many years. **HTML5 mobile apps** are the second choice of mobile users since they are lightweight and cross-platform. HTML5 mobile apps are also functional and perform well on mobile devices and they are **coming closer** to native apps. Mobile widgets come third because they are lightweight and quite convenient to use. Users are not interested in mobile Web apps because they are slow, low functional and unattractive. The result remain the same in the practical verification. It is understandable that native app is more preferable than the HTML5 mobile app paradigm. The native app is very robust and convenient to use while the HTML5 mobile app has several limitations. The usability and the performance of the HTML5 app are lower than the native app. An alternative solution is to use HTML5 wrapped with PhoneGap framework that make the app more native and powerful until HTML5 specifications are completed.

For service/ content provider, **HTML5** mobile apps are the best. Service providers can build an HTML5 mobile app once and distribute it everywhere. They can hence seamlessly deliver their content and service to mobile users. Mobile Web apps work in the same way and get the second position in the race. Mobile widgets and native apps have the lowest grade because the platform fragmentation still exists when implementing the apps on mobile phones. Furthermore, service provider must deploy widgets and native apps onto every device. It is a complicated process and incredibly increases the cost and effort of service providers.

6.2. Key finding and recommendation

W3C announces that HTML5 is a cross-platform solution that can hook into device's hardware (e.g. camera) to create a powerful mobile Web app. In fact, developers face a fragmentation in mobile platforms and browsers to build an HTML5 app to access hardware. For example, HTML media capture is the first API in 2011 to standardize media capture on Web. It works by overloading the <input type="file">and adding new values for the accept parameter, allowing users to capture their snapshot with device's camera. However, the API is too limited to use and only works on Android 3.0 browser. Another example is the implementation of getUserMedia API belonging to W3C WEBRTC (Web Real-Time Communication) working group to access device's camera. Only Google and Opera currently have developer builds that include the API, and only Opera Lab on Android supports mobile developers to use it in their apps.

Native apps are well-known for their fast and responsive user interface, and the seamless capability to access hardware features. However, creating such apps is complicated and requires much effort from developers on any platform adopted. Meanwhile, jQuery Mobile simplify the code to build an attractive and adaptive user interface for HTML5 mobile apps. PhoneGap framework also let developers make the HTML5 app access native APIs much easily and deploy the app on multiple platforms. PhoneGap use the same native APIs with native apps but abstract them so that developers can write apps in HTML and JavaScript. Therefore, we recommend that the most effective solution now to build the apps that are capable of using device hardware (GPS, accelerator and camera) and working cross-platform is to wrap HTML5, jQuery Mobile and CSS3 with PhoneGap framework. However, PhoneGap apps cannot absolutely replace native apps because they perform slower than native apps due to the overhead from an abstraction and HTML render in addition to the time to execute the native processes.

6.3. Summary and conclusion

In our project, we classify mobile apps into four mobile app paradigms: Native apps, mobile Web apps, mobile widgets and HTML5 mobile apps. We then analyse and evaluate the paradigms from developer, user and service/content provider viewpoint based on pre-defined criteria. For the objectiveness and accuracy in our evaluation, we start by defining high-level criteria and then breaking them down into finer and more detailed criteria. The points for the evaluation are given based on the information collected on the Internet, especially the different user and developer social communities. After the analysis and evaluation, we conclude that **native apps and HTML5 mobile apps** keep their first places in the race of mobile paradigms. **Mobile widgets** are still

valuable but their role is no longer so important on mobile devices. **Mobile Web apps** will become a history and they are soon replaced by **HTML5 mobile apps**.

We also perform the practical verification in which we build an object recognition app by using the two most promising paradigms namely native and HTML5. The mobile app we develop will access both device's hardware features (camera, storage and internet access) and Web service to ensure that all capabilities of a smart phone are adopted in the practical verification. The verification has the same result as the analysis and evaluation. HTML5/ PhoneGap mobile app is the most preferable paradigm employed by mobile developers. However, native app paradigm should be adopted to create a seamless user experience. Users do not select HTML5 as the best mobile app paradigm because HTML5 is still limited to use. It is understandable when HTML5 is still growing and it takes time before the complete version is available. W3C announces that there will indeed be a finalized version of HTML5 until the middle of 2014 and we believe that when HTML5 is complete, it will open a new era of mobile Web app.

Reference

1. S. Tarkoma (Ed), *Mobile Middleware-Architectures, Patterns, and Practice*, pp. 3-4, Wiley, 2009.
2. S. Tarkoma and E. Lagerspetz, "Arching over the Mobile Computing Chasm: Platforms and Runtimes," *Computer,* vol. 44, no. 4, pp. 22-28, April 2011.
3. J. Le. Feuvre, C. Concolato and J. C Dufourd, "Widgets mobility," In *Proceedings of the 6th International Conference on Mobile Technology, Application; Systems* (Mobility '09), no. 25, ACM, New York, NY, USA.
4. C. Raibulet and D. Cammareri, "Automatic generation of mobile widgets," *International Journal of Pervasive Computing and Communications*, 7(2), pp. 132-146, 2011.
5. W3C. Client-Side Web Applications (Widgets) Requirements. [Online] [Cited: September 2011]

http://www.w3.org/TR/2006/WD-WAPF-REQ-20061109/

6. W3C. Widgets 1.0. [Online] [Cited: September 2011]

http://www.w3.org/TR/2006/WD-widgets-20061109/

7. S. Sire, M. Paquier, A. Vagner and J. Bogaerts, "A messaging API for inter-widgets communication," In *Proceedings of the 18th international conference on World Wide Web*, April 20-24, 2009, Madrid, Spain, doi: 10.1145/1526709.1526884.
8. Opera. Opera widgets. [Online] [Cited: September 2011]

http://widgets.opera.com/

9. A. Kostiainen, *The Web at Runtime in mobile context*, Master's thesis, pp. 67-70, Helsinki University of Technology, 2008.
10. W3C. Widgets 1.0 requirement. [Online] [Cited: September 2011]

http://www.w3.org/TR/2007/WD-widgets-reqs-20070209/

11. C. Kaar, "An Introduction to Widgets with Particular Emphasis on Mobile Widgets," *Computing*, October 2007.
12. A. Lee and P. M. Road, "Mobile Web Widgets: Enabler of Enterprise Mobility Work," *The 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web,* 2009.
13. F. W. Zammetti, *Practical JavaScript, DOM Scripting, and Ajax Projects*, pp. 149-151, Apress, 2007.
14. A. Jaokar and T. Fish, *Mobile Web 2.0*, futuretext, 2006.

15. B. Pejiü and A. Pejiü, “Uses of W3C's Geolocation API,” *Computational Intelligence and Informatics (CINTI)*, 11th International Symposium on, vol., no., pp. 319-322, 18-20 Nov. 2010.

16. Y. Liu and E. Wilde. “Personalized location-based services,” In *Proceedings of the 2011 iConference* (iConference '11). ACM, New York, NY, USA, pp. 496-502.

17. Y. Huang, J. Cao, B. Jin, X. Tao, and J. Lu, “Cooperative cache consistency maintenance for pervasive internet access,” Wirel. *Commun. Mob. Comput.,* pp. 436-450, March 2010.

18. F. Reynolds, ”Web 2.0–In Your Hand,” *Pervasive Computing, IEEE* , vol.8, no.1, pp. 86-88, Jan.-March 2009, doi: 10.1109/MPRV.2009.22.

19. Truste. HTML5 and mobile privacy. [Online] [Cited: September 2011]

http://www.truste.com/blog/2011/05/25/html5-and-mobile-privacy/

20. S. Aghaee and C. Pautasso, “Mashup Development with HTML5,” In *Proceedings of the 3rd and 4th International Workshop on Web APIs and Services Mashups, Ayia Napa, Cyprus*, December 2010.

21. E. M. Maximilien, “Mobile Mashups: Thoughts, Directions, and Challenges,” *Semantic Computing, 2008 IEEE International Conference on*, vol., no., pp. 597-600, 4-7 August 2008, doi: 10.1109/ICSC.2008.100.

22. Zokem. ”Applications capture already half of mobile Internet”. [Online] [Cited: October 2011]

http://www.zokem.com/2010/09/applications-capture-already-half-of-mobile-internet-traffic/

23. Flurry. “Mobile apps put the Web in their real mirror”. [Online] [Cited: October 2011]

http://blog.flurry.com/bid/63907/Mobile-Apps-Put-the-Web-in-Their-Rear-view-Mirror

24. J. Hansen and G. Ghinea, ”Android vs Windows Mobile vs Java ME: a comparative study of mobile development environments,” In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, June 23-25, 2010, Samos, Greece, doi: 10.1145/1839294.1839348

25. G. Chang, C. Tan,G. Li and C. Zhu, ”Developing mobile applications on the Android platform,” In: X. Jiang, M.Y. Ma, C.W. Chen (eds.) Mobile Multimedia Processing: Fundamentals, Methods, and Applications. Springer, 2010, Heidelberg.

26. S. Komatineni, D. MacLean and S.Y. Hashimi, *Pro Android 3*, pp. 927-939, Apress, 2011.

27. M. Scot and L. Mike, *In Learn OS X Lion*, pp. 629-648, Apress, 2011.

28. R. Wentk, *XCode 4*, p. 7, Apress Inc, 2009

29. Microsoft. Installing Windows Phone SDK. [Online] [Cited: October 2011]

http://msdn.microsoft.com/en-us/library/ff402530(v=vs.92).aspx

30. N. Lecrenski, K. Watson and R. Fonseca-Ensor, *Beginning Windows Phone 7 Application Development: Building Windows Phone Applications Using Silverlight and XNA* (1st ed.), pp. 11-18 ,Wrox Press Ltd., Birmingham, UK, 2011.

31. R. Cameron, *Pro Windows Phone 7 Development*, pp. 50-51, Apress, 2011.

32. S. Allen, *Pro Smartphone Cross-Platform Development: iPhone, Blackberry, Windows Mobile, and Android Development and Distribution*, 1st ed, pp. 29-33, Apress, 2010.

33. Facebook. Mobile Web app tutorial. [Online] [Cited: November, 2011]

http://developers.facebook.com/docs/mobile/Web/build/#payments

34. M. Mamone, *Migrating to iPhone and iPad for .NET Developers*, pp. 233-260, Apress, 2011.

35. P. Crocker and M. Lewis. Mobile Widget Platform Market Analysis: Understanding the Business Case and ROI, March 2010. UK. [Online] [Cited: December, 2011]

http://www.smithspointanalytics.com/MobileWidgetPlatformMarketAnalysis.pdf

36. J. Mundy and A. Zisserman (Ed), *Geometric invariance in computer vision*, MIT Press, 1992.

37. H. Muras and S. K. Nayar, ”Visual learning and recognition of 3-D objects from appearance,” *International Journal of Computer Vision*, vol. 14, pp. 5-24, 1995.

38. P. Belhumeur and D. Kriegman, ”What is the set of images of an object under all possible illumination conditions,” *International Journal of Computer Vision*, vol. 28, no. 3, pp. 1-16, 1998.

39. B. S. Song, K. M. Lee and S. U. Lee. Model-based object recognition using geometric invariants of points and lines. *Computer Vision and Image Understanding*, vol. 84, no. 3, pp. 361–383, 2001.

40. N. Winters and J. Santos-Victor, ”Information Sampling for Appearance based 3D Object Recognition and Pose Estimation,” In *Proceedings of the 2001 Irish Machine Vision and Image Processing Conference*, Maynooth, Ireland, September 2001.

41. Y. Hashimi and S. Komatineni, *Pro Android*, pp. 21-23, Apress, 2009.

42. W3C. getUserMedia: Getting access to local devices that can generate multimedia stream. [Online] [Cited: April 2012]

http://dev.w3.org/2011/webrtc/editor/getusermedia.html

43. C. Karlof, U. Shankar, J. D. Tygar and D. Wagner, ”Dynamic pharming attacks and locked same-origin policies for web browsers,” In *Proceedings of the 14th*

ACM conference on Computer and communications security, October 28-31, 2007, Alexandria, Virginia, USA.

44. Apache Commons. Common Codecs. [Online] [Cited: May 2012]

http://commons.apache.org/codec/

45. PhoneGap. API reference. [Online] [Cited: May 2012]

http://phonegap.com/start#android

46. PhoneGap. PhoneGap Build. [Online] [Cited: May 2012]

https://build.phonegap.com/

47. S. Helal, J. Hammer, J. Zhang and A. Khushraj, ”A three-tier architecture for ubiquitous data access,” *ACS/IEEE International Conference on Computer Systems and Applications* , pp. 177-180, 2001.

48. S. Pfeiffer and C. Parker, ”Accessibility for the HTML5 <video> element,” In *Proceeding of 6th International Cross-Disciplinary Conference on Web Accessibililty* (W4A '09). ACM, pp. 98-100.

49. S.Fulton and J.Fulton*, HTML5 Canvas*, pp. Ed.Orelly Media. 2011.

50. A. Charland and B. Leroux, ”Mobile application development: Web vs. Native,” *Communications of the ACM*, vol. 54 no. 5, May 2011.

51. L. Chittaro, ”Visualizing information on mobile devices,” *Computer* , vol.39, no.3, pp. 40-45, March 2006.

52. M. Pilgrim, *HTML5: Up and Running*, p. 23, O'Reilly Media.

53. W. West and S. M. Pulimood, ”Analysis of privacy and security in HTML5 web storage,” *The Journal of Computing Sciences in Colleges*, vol. 27, no. 3, pp. 80–87, 2011.

54. P. Lubbers, B. Albers and F. Salim, *Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development*, pp. 243-256, Apress, 2010.

55. P. F. Felzenszwalb and R. Zabih, ”Dynamic Programming and Graph Algorithms in Computer Vision,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on* , vol. 33, no. 4, pp. 721-740, April 2011.

56. D.G. Lowe and J. Tsotsos (Ed), ”Object recognition from local scale-invariant features. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, no., pp. 1150-1157 vol.2, 1999.

57. N. Pinto, D. D. Cox, J. J. DiCarlo and K. J. Friston (Ed), ”Why is Real-World Visual Object recognition Hard,” *PLoS Comput Biol,* vol. 4, no. 1, p. 27, 2008.

58. M.J. Johnson, H.K. A, ”Porting the google android mobile operating system to legacy hardware,” In *Proceeding IASTED Int. Conf. on Portable Lifestyle Devices (PLD 2010)*, Marina Del Rey, USA (2010), pp. 620–625.

59. P. Mendes, M. Caceres and B. Dwolatzky, ”A review of the widget landscape and incompatibilities between widget engines,” *AFRICON, 2009*, vol., no., pp.1-6, 23-25 September 2009.

60. A. I. Wasserman, ”Software engineering issues for mobile application development,” In *Proceedings of the FSE/SDP workshop on Future of software engineering research FoSER ’10*, 2010, pp. 397-400.

61. T. Melamed and B. Clayton, ”A Comparative Evaluation of HTML5 as a Pervasive Media Platform,” *1st International ICST Conference, MobiCASE*, San Diego, CA, USA, vol. 35, pp. 307-325, 26-29 October 2009.

62. G. Buchanan, S. Farrant, M. Jones, H. Thimbleby, G. Marsden and M. Pazzani, ” Improving mobile internet usability,” In *Proceedings of the 10th international conference on World Wide Web (WWW '01)*. ACM, New York, NY, USA, pp. 673-680.

63. B. Zhang, T. Xu, W. Wang and X. Jia, ”Research and implementation of cross-platform development of mobile widget,” *Communication Software and Networks (ICCSN), 2011 IEEE 3rd International Conference on* , vol., no., pp.146-150, 27-29 May 2011.

64. D. Rajapakse, ”Techniques for De-fragmenting Mobile Applications: A Taxonomy,” In *Proceedings of 20th Intl. Conf. on Software Engineering and Knowledge Engineering Conference* (SEKE'08), July 2008.

65. M. Siy, Q. Zhijie and L. Lei, ”Research on Mobile Web Applications End to End Technology,” *10th International Conference on Computer and Information Technology (CIT) IEEE 2010* , vol., no., pp.2061-2065, June 29 2010-July 1 2010.

66. K. Kimbler, ”App store strategies for service providers,” *14th International Conference on Intelligence in Next Generation Networks (ICIN) 2010* , vol., no., pp.1-5, 11-14 Oct. 2010.

67. H. Cramer, N. Belloni, F. Kaplan and P. Jermann (Ed), ”Research in the large. using app stores, markets, and other wide distribution channels in Ubicomp research,” In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing- Adjunct* (Ubicomp '10 Adjunct), 2010, ACM, New York, NY, USA, pp. 511-514.

68. S. Schäfer, S. Christmann and S. Hagenhoff, ”W3C Widgets - A solution for implementing platform-independent mobile applications,” *International Conference on Web Information Systems and Technologies WEBIST 2011*. pp. 115-118.

69. Y.-W. Kao, C.-F. Lin, K.-A. Yang and S.-M. Yuan, ”A Cross-Platform Runtime Environment for Mobile Widget-Based Application,” *International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2011* , vol., no., pp.68-71, 10-12 Oct. 2011.

70. A. Shanker and S. Lal, ”Android porting concepts,” *3rd International Conference o*n *Electronics Computer Technology (ICECT), 2011* , vol. 5, no., pp. 129-133, 8-10 April 2011.