Beteckning:_____

HÖGSKOLAN
I GÄVLE

Akademin för teknik och miljö

Utvärdering av multiplattformsutvecklingsverktyg för smarta mobiler

Thomas Pehrson
september 2011

Examensarbete 15 hp, C
Datavetenskap

Datavetenskapliga programmet, inriktning IT-arkitekt
Examinator: Jonas Boustedt
Handledare: Fredrik Bökman

Utvärdering av multiplattformsutvecklingsverktyg för smarta mobiler

av

Thomas Pehrson

Akademin för teknik och miljö
Högskolan i Gävle

S-801 76 Gävle, Sweden

Email:
ndv08tpn@student.hig.se

Abstrakt

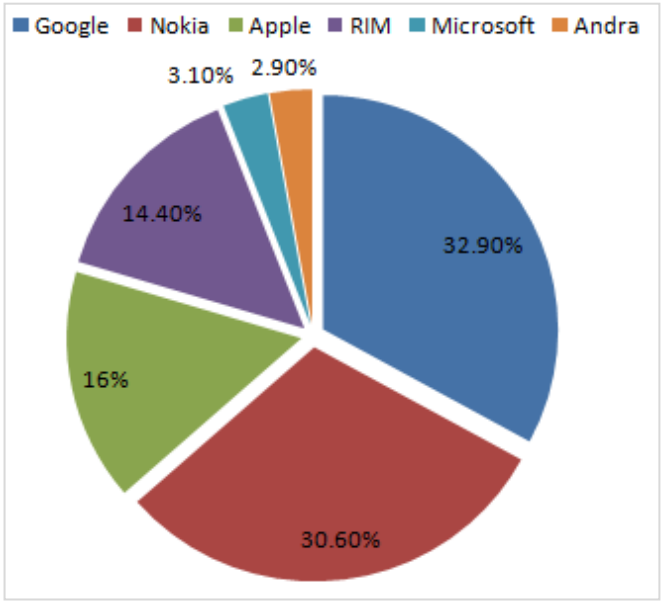
Genom att använda ett multiplattformsutvecklingsverktyg kan utvecklare distribuera applikationer till flera olika plattformar. Problemet är att dessa verktyg inte är lika välanvända eller dokumenterade som de dedikerade (t.ex. Android SDK) och det kan vara svårt för enskilda utvecklare eller företag att veta vilket verktyg de ska välja. Efter en grundlig kartläggning av flera multiplattformsutvecklingsverktyg valdes slutligen Corona SDK och PhoneGap ut för vidare utvärdering. Utvärderingen utfördes genom att skapa en applikation som sparar, raderar och visar noter i båda verktygen liknande en utvecklad med Android SDK. Därefter jämfördes utvecklingsprocessen med fokus på gränssnittsutformningen och databaskopplingen samt slutligen de färdiga applikationernas prestanda och minnesanvändning avgränsat till Android plattformen. Resultatet av utvärderingen visade att applikationen utvecklad i Android SDK hade bäst prestanda överlag och även minnesanvändningen var mindre än hos de övriga. Både Corona SDK och PhoneGap lämpar sig till att skapa liknande applikationer som använder en databas. PhoneGaps fördel var att det var lätt att utveckla gränssnittet genom HTML/CSS, dock var det långsamt samt mer minneskrävande jämfört med de övriga. Corona SDKs applikation hade bra prestanda med liknande tider som Android SDKs men en nackdel var att gränssnittutformningen var mer tidskrävande.

Innehåll

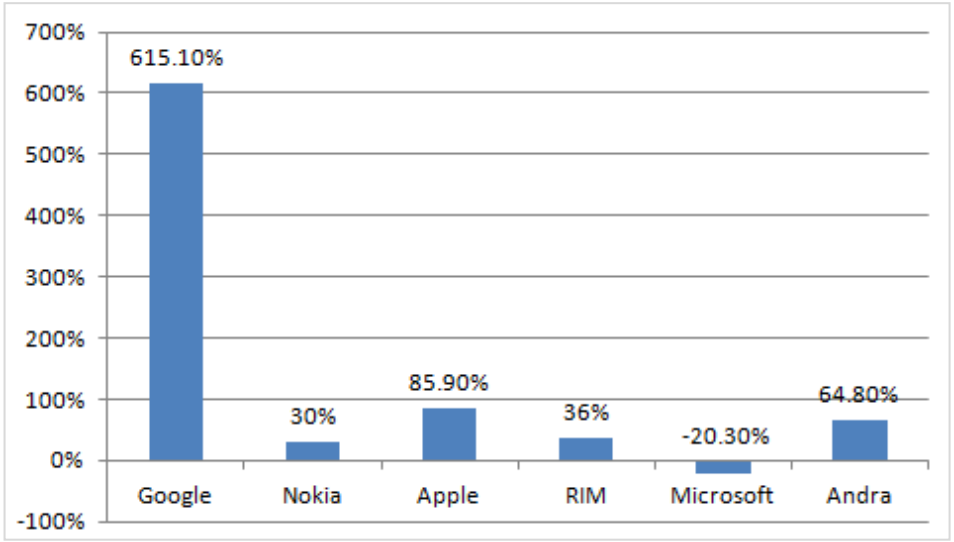
1 Inledning

Idag finns det en mängd olika varianter av mobiltelefoner. Under de senaste åren har smarta telefoner eller smartphones blivit allt populärare. Det som skiljer dessa från vanliga traditionella mobiltelefoner är att de bl.a. har ett fysiskt och/eller virtuellt tangentbord, samt att de beter sig mer likt datorer.

Smartphonemarknaden har ökat stadigt sedan 2009 då det enligt analytikerföretaget Canalys såldes 53,7 miljoner smartphones [1]. Under 2010 ökade försäljningen med 88,6% och det såldes under samma period hela 101,2 miljoner exemplar. Tidigare har Nokia varit störst och ledande på marknaden med deras Symbian operativsystem, men enligt Canalys har nu Googles Android gått förbi (se figur 1). Googles operativsystem stod för en ökning med 615,1% (se figur 2) vilket medför att de nu har 32,9% av marknaden. Även Apples försäljning ökade under samma period med 85,9%.



Figur 1. Totala marknadsandelar 2010



Figur 2. Försäljningsökning kvartal 4 -10 jämfört med kvartal 4 -09

1.1 Applikationer på smartphones

I smartphone-sammanhang pratas det om appar, en förkortning för applikationer som körs på mobiltelefonen. Appar är ett stort skäl till att de smarta telefonerna har blivit allt mer populära. Följande citat är taget ur en av Sveriges stora dagstidningar DN.se:

”Alla mobiltillverkare satsar nu på smartphones och nya operativsystem dyker ständigt upp. Det absolut senaste är förstås surfplattorna som blivit årets julklapp. Gemensamt för dem alltihop är att de kör appar och att det är detta som lyfter upplevelsen till nya höjder.” [2]

Mobila applikationer skiljer sig enligt Wasserman [3] från andra inbäddade applikationer ofta på flera punkter:

1. Interaktion med andra applikationer. Andra liknande enheter har ofta bara fabriksinstallerad mjukvara, mobila enheter har däremot många program från olika källor som kan samspela med varandra.
2. Avkännare. T.ex. en tryckskärm som svarar på olika gester, GPS, multipla nätverksprotokoll m.m.
3. Rena och hybrida mobila webbapplikationer. Till skillnad från andra inbäddade system så använder mobila applikationer tjänster som körs över telefonnätverket eller internet via webbläsare.
4. Olika versioner av hårdvara och mjukvara på plattformarna. T.ex. Android körs på flera olika enheter och finns i olika versioner.
5. Användargränssnitt. Den mobila applikationen måste dela gemensamma element i användargränssnittet med andra applikationer.
6. Komplicerade att testa. Svårt att testa t.ex. sändning genom förmedlingsnoder och telefonnätverk.
7. Kraftkonsumtion. Mobila applikationer kan oavsiktligt dränera batteriet, till skillnad från dedikerade enheter som är optimerade för maximal batteritid.

1.2 Applikationsutveckling

Även om smartphones beter sig på ett likartat sätt så kan de på insidan skilja sig åt väsentligt genom att de använder olika operativsystem, vilket medför att utvecklare måste konstruera programmen på olika sätt [4].

För att åstadkomma detta är det enligt Wasserman [3] vanligt att utvecklare använder sig av dedikerade utvecklingsverktyg som är skapade för den aktuella plattformen:

- Apples iPhone applikationer utvecklas med iOS SDK (Software Development Kit).
- Android utveckling kan ske med plugginen Android Development Tools för Eclipse.
- Windows Phone utvecklare använder sig av en specialiserad version av Microsofts Visual Studio Environment.

En applikation skriven i iOS kommer därför inte att fungera på Android och vice versa. Biao et al [5] menar att detta ställer till med problem för utvecklare:

1. Det är tidskrävande för utvecklare eftersom de är tvungna att lära sig varje OS:s SDK och API (Application Programming Interface).
2. Det blir ineffektivt och oekonomiskt att migrera applikationer till nya plattformar, eftersom all programmering till viss del måste göras om.

3. Det blir svårt att underhålla de olika applikationsversionerna. Om en version behöver uppdateras så är utvecklaren tvungen att skriva uppdateringar för alla applikationer eftersom de körs på olika plattformar.

Verktyg som möjliggör multiplattformsutveckling av mobila applikationer kan lösa dessa problem.

1.3 Multiplattformsutveckling

En av de populärare programmeringsmiljöer som kan användas för att uppnå plattformsoberoende är Java. Javaprogram kan köras på flera olika plattformar så länge plattformen har en JVM (Java Virtual Machine). En lösning för att uppnå plattformsoberoende är att använda sig av Java ME, en variant som är utformad mot inbäddade system. Ett problem med att använda Java för att multiplattformsutveckla mobila applikationer är att ett flertal smartphones saknar sådant stöd (t.ex. iPhone) och om det stöds är det inte helt säkert att det fungerar felfritt. Detta eftersom Javas portabilitet bygger på att tillverkaren har implementerat JVM helt enligt Java specifikationen, vilket är sällsynt.

Android använder sig av en skräddarsydd JRE (Java Runtime Environment) kallad Dalvik, vilket gör att program som är utvecklade mot Android inte kan köras på andra JVMs [6].

1.3.1 Utvecklingsverktyg

Eftersom Java ME inte visat sig vara ett bra val för att skapa plattformsoberoende applikationer på mobiltelefoner har det växt fram andra alternativ. Ett multiplattformsutvecklingsverktyg kan skapa mobila applikationer till flera olika plattformar från samma eller en smått modifierad kodbas.

Då detta är ett relativt nytt område så är multiplattformsutvecklingsverktygen inte lika dokumenterade eller utvärderade som de renodlade verktygen (t.ex. Android SDK eller iOS SDK) vilket kan avskräcka utvecklare från att anamma dem. Dessutom anser flera att resultatet inte blir lika bra som en native[1] applikation [7],[8]. Ännu ett problem för företag eller den enskilda utvecklaren kan vara att det är svårt att veta vilket verktyg de ska använda om de vill starta utveckling mot flera olika plattformar, då det finns en uppsjö olika och att det fortfarande dyker upp nya.

Bland gratisverktygen hittar vi RhoMobile Rhodes [9], Appcelerator Titanium Mobile [10], PhoneGap [11], moSync [12], som alla rekommenderas av Mashable.com [13] och är enligt dem ett bra alternativ till att hoppa över eller lägga resurser på att utveckla samma applikation igen på en annan plattform. Utöver dessa finns t.ex. Corona SDK [14] och AirPlay SDK [15] som till skillnad från de tidigare nämnda tar betalt för distribution av apparna.

1.4 Syfte

Det primära syftet med den här studien är att utvärdera om en applikation utvecklad med något av dessa multiplattformsutvecklingsverktyg har samma prestanda som en likartad applikation utvecklad i det dedikerade Android utvecklingsverktyget. Utöver detta ska verktygens utvecklingsprocess jämföras sinsemellan. Syftet är även att göra

[1] En applikation som körs rent, i ursprunglig form (t.ex. en applikation utvecklad enbart mot Android-plattformen i dess dedikerade utvecklingsverktyg).

en kartläggning av verktygen för att därefter utvärdera vilka som är lämpligast att utveckla applikationen med.

Utvärderingen ska resultera i prestandamätningar och en mindre analys av minnesanvändningen av de färdiga applikationerna. Utvärderingen ska även resultera i en diskussion både om fördelar och nackdelar av de valda verktygen men även om deras lämplighet för utveckling av liknande applikationer.

Målet är att resultatet skall vara användbart både för företag och enskilda utvecklare som är intresserade av att utveckla mot flera olika smartphone plattformar.

2 Metod

Genom faktainsamling med bl.a. sökningar på Google har jag identifierat ett flertal olika verktyg som utger sig för att kunna skapa mobila applikationer som kan köras på en rad olika mobiltelefonplattformar. Information om dessa verktygs specifikationer har därefter samlats in från deras officiella hemsida och dokumentation.

Av dessa verktyg valdes två ut (PhoneGap och Corona SDK) för att utvärderas mer ingående genom att utveckla och driftsätta en applikation med hjälp av dem. Utöver detta har en applikation även utvecklats i ren Android-miljö så att resultatet har kunnat jämföras. Applikationen har baserats på Android developers ”Notepad tutorial” [16], med ändringar på bl.a. det grafiska gränssnittet för att vara mer multiplattformsvänligt. För att avgränsa och öka fokuseringen har applikationerna enbart testats mot Android-plattformen.

Installation och utveckling av applikationerna har skett på samma hårdvara med Windows 7, och applikationerna har installerats och testkörts på både en Samsung Galaxy S med Android 2.2 och en Galaxy S2 med Android 2.3. Utvärderingen av notapplikationen har gjorts på den sistnämnda.

2.1 Litteraturstudie

För att hitta information om ämnet har jag använt mig av Google samt online-biblioteken IEEE Xplore [17] och ACM Digital Libary [18] och sökt efter nyckelord som: Mobile Cross-Platform, Multiplattform, PhoneGap, Corona SDK, moSync och Rhodes.

2.2 Val av applikation för utveckling

Applikationen ska ha en fungerande databas där det går att spara, hämta och radera data då detta är en viktig del i en företagsapplikation [19],[20]. Databasen ska vara plattformsoberoende så att den fungerar på minst Android och iPhone. Som utgångspunkt valdes exempelapplikationen Notepad från Android Developer [16] eftersom den hade en rimlig omfattning samt hade en bra dokumenterad databaskoppling. Gränssnittet har därefter gjorts om för att bli mer multiplattformsvänligt. Detta har åstadkommits genom att ta bort användningen av menyknappen samt händelsehantering på s.k. long-clicks[2]. All navigering i applikationen sköts genom den tryckkänsliga skärmen för att den teoretiskt ska kunna fungera på alla olika smartphones.

2.2.1 Riktlinjer

För att utvärderingen av utvecklingsprocessen och prestandamätningen ska bli rättvisande har ett par riktlinjer upprättats. Det viktigaste är att applikationerna fungerar likadant, då resultaten annars blir missvisande. Däremot är det inte viktigt att de grafiska objekten (t.ex. knappar) i gränssnittet ser exakt lika ut, det räcker att de är av samma typ.

[2] När användaren trycker och håller en kortare tid på en viss punkt på skärmen.

2.3 Kartläggning

De verktyg som identifierats har jämförts på en rad olika områden innan utvecklingen påbörjats för att avgränsa verktygen till två för utveckling av den slutgiltiga applikationen. Dessa områden är viktiga eftersom jag anser att de ger utvecklaren en bra överblick över verktyget och dess begränsningar:

- **Specifikationer**
  - Distributionsplattformar
  - Programmeringsspråk
  - Licens

- **Installation**
  - Nödvändig programvara att installera
  - Omfattning

- **Utvecklingshjälp**
  - Dokumentation
  - Community
  - Exempelapplikationer

- **Kompilering och testkörning av applikation**
  - Typ av emulator
  - Omfattning

2.4 Avgränsning och val av multiplattformsutvecklingsverktyg

För att avgränsa mig har jag valt utvecklingsverktygen utifrån dessa fem kriterier:

- **Android- och iPhone distribution**

Verktyget ska minst stöda dessa plattformar då de är de störst växande smartphone varianterna [1].

- **Dokumentation och exempel**

Utvecklingsverktyget ska vara snabbt att komma igång med genom att tillhandahålla exempelapplikationer och omfattande dokumentation, som en väldokumenterad API.

- **Databaskoppling**

En databas är en vitaldel i en mobilapplikation, särskilt om den ska användas i företaget [20]. Eftersom ett av kriterierna på applikationen var att stöda någon typ av databas är det viktigt att verktyget stöder en funktion för att spara och hämta data. Viktigt är även att det finns exempelkod att tillgå som visar hur detta implementeras.

- **Licens**

Då budgeten är limiterad ska verktygen vara gratis att använda och ladda hem. Det är däremot acceptabelt att det kostar en avgift om applikationerna därefter ska distribueras.

- **Testkörning**

Eftersom tiden är begränsad ska det vara enkelt att testköra applikationerna antingen genom den virtuella emulatorn eller i den fysiska mobiltelefonen.

2.5 Utvärdering

Under utvecklingsfasen har verktygen jämförts sinsemellan som hur implementationen av databaskopplingen och gränssnittsutformningen skiljer sig åt. Efter utvecklingsfasen har applikationerna kompilerats och installerats på en fristående smartphone och därefter utfördes prestandamätningar för olika funktioner. Den färdiga applikationens minnesanvändning har också analyserats.

2.5.1 Under utveckling

Under utvecklingen har utformningen av gränssnittet jämförts mellan de olika verktygen med fokus på hur vyerna skapas samt hur koden för att uppdatera listan med notelement implementeras. Utöver detta har databaskopplingskoden jämförts med avseenden som hur en instans av databasen skapas och hur transaktioner utförs.

2.5.2 Prestandamätning

För att mäta prestandan har fem olika mätpunkter valts ut för mätning av exekveringstid. För databasfunktioner har exekveringstid för att skapa tabellen, skapa fyra noter med tillhörande data samt tiden för att ta bort en not mätts. Gränssnittets responsivitet har beräknats genom att mäta på hur lång tid det tar för en ny vy att visas.

Prestandamätningen har utförts med stopwatch-tekniken [21] genom att logga tidpunkten i logcat [22] som är beskrivet i ”Measure performance in Android SDK” [23] före samt efter händelsen och därefter beräkna mellantiden. Genom att använda den tidigare nämnda stopwatch-tekniken fungerar tidsmätningen i alla utvecklingsverktyg så länge det har en funktion liknande Javas *currentTimeMillis()* för att beräkna tid.

2.5.3 Minnesanvändning

Minnesanvändning i Android är ett komplicerat område och minnesinformationen kan vara missledande eftersom det kan vara oklart hur mycket minne en applikation tar upp på grund av att stor del av minnet är delat med andra processer.

Dock kan vi undersöka två värden för att få en approximation av minnesanvändningen för applikationen: Uss (Private dirty) samt Pss [24]. Uss kan förklaras som minnet som skulle frigöras till systemet om processen avslutas. Pss är minnet som inte skulle frigöras om processen avslutades, men ändå bidrar till den totala minnesmängden. Genom att analysera dessa värden kan vi få en uppskattning om hur mycket minne applikationen använder [25]. För att analysera minnet har Android Debug Bridge [26] använts som beskrivet på Texas Instruments ”Android memory analysis” [27].

3 Resultat

Genom att kartlägga de identifierade multiplattformsverktygen har det sedan valts ut två verktyg att slutföra notapplikationen på från de kriterier som togs upp i kapitel 2.4. Efter kartläggningen har verktygen jämförts sinsemellan på olika områden, och därefter har bl.a. prestandan på de färdigutvecklade applikationerna utvärderats.

3.1 Kartläggning

Kartläggningens resultat är framtagna genom faktainsamling från respektive utvecklingsverktygs hemsida eller dokumentation samt personlig testning. Resultaten består till viss del av mina subjektiva åsikter som representerar en student som tidigare aldrig använt något av verktygen.

För att underlätta och minimera irrelevant data har jag generaliserat och samlat ihop t.ex. alla iOS versioner under samlingsnamnet iPhone som plattform, även om verktyget inte stöder alla funktioner fullt ut. Installationen och testkörning har enbart gjorts som jag nämnt tidigare för utveckling av Android applikationer.

Kartläggningen startades i maj 2011 och uppgifterna har inte uppdaterats efter augusti 2011. Då några av dessa uppgifter har ändrats under denna period så har det slutgiltiga valet av utvecklingsverktyg baserats på den tidigaste utförda kartläggningen. Avsnitt 3.1.1 till 3.1.6 är en sammanställning av kartläggningen.

3.1.1 Appcelerator Titanium Mobile

Specifikationer

Plattformar:	iPhone och Android (Blackberry för betatestare)
Licens:	Apache Public License (version 2)
Programmeringspråk:	HTML, JavaScript, CSS, Python, Ruby och PHP
Databas:	Internal SQLite

Installation

Omfattande installation och konfiguration. Nödvändig programvara: Python 2.7, Git, Sun/Oracle JDK, Android SDK Tools, Titanium Developer.

Utvecklingshjälp

Dokumentationen är lättåtkomlig och lättnavigerad. API finns att tillgå online på dess officiella hemsida. Community är gratis men extra tjänster som t.ex. behörighet till ”knowledge base” kostar pengar. Community är mycket aktiv med just nu 15,193 inlägg.

Officiella exempelapplikationer: Hello world exempel för att snabbt komma igång. Ett exempelbibliotek kallat ”Kitchen sink” finns att tillgå där nästan alla funktioner demonstreras.

Testkörning och kompilering

Android applikationer testkörs genom Android SDKs installerade emulator, vilken först måste konfigureras. Relativt långsam att starta jämfört med andra verktygs inbyggda emulatorer. Enkel kompilering.

3.1.2 Corona SDK

Specifikationer

Plattformar:	iPhone och Android
Licens:	Gratis att använda. Däremot krävs det ett abonnemang för att distribuera apparna. iOS Indie (199$/år) enbart iOS utveckling, Android Indie (199$/år) enbart Android utveckling samt Pro (349$/år) där allt ingår
Programmeringspråk:	LUA
Databas:	Internal SQLite

Installation
Enkel installation. Enbart Corona SDK behöver installeras för att börja utvecklingen, och kod skrivs i valfri text editor t.ex. anteckningar. För att köra programmen på annat än den inbyggda emulatorn så behövs även Java SDK.

Utvecklingshjälp
Omfattande och välskriven dokumentation. API referensen är enkel att förstå då den förutom att beskriva funktionerna även visar hur de kan användas genom exempelkod. En introduktion kallad ”Corona in 5 minutes” visar enkelt hur utvecklaren snabbt kommer igång med utvecklingsverktyget. Deras officiella forum är mycket aktivt med tiotusentals inlägg i flera undergrupper.

Officiella exempelapplikationer: Mycket omfattade, 95st uppdelade i flera kategorier från enkla som ”Getting started” till mer avancerade. Ingående och bra dokumenterade demonstrationer gör det lätt att komma igång.

Testkörning och kompilering
Applikationer körs genom Corona SDKs egna emulator, vilken stöder de flesta funktioner. Några av funktionerna kan inte testas genom emulatorn utan måste först kompileras och sedan testas på mobiltelefonen, vilket blir tidskrävande. Kompileringen är jämfört med tidigare kartlagda verktyg tidskrävande, då den först måste verifiera användaren mot en extern server.

3.1.3 moSync

Specifikationer

Plattformar:	iPhone, Android, J2ME, Windows Mobile, Symbian och MeeGo
Licens:	Gratis att använda. GNU General Public License (GPL) version 2. Gratis att använda och distribuera applikationerna om källkoden också publiceras. Indie licens som är gratis finns för att distribuera applikationer utan källkod. Det finns även olika betalalternativ för support
Programmeringspråk:	C/C++
Databas:	Ingen officiell

Installation
Simpel installation. Programvara: moSync SDK och Sun/Oracle JRE

Utvecklingshjälp
Mycket dokumentation finns att tillgå från moSyncs ”Dev Center” uppdelat under flera underkategorier för att lätt hitta rätt. API referensen är just en referens, inte

tillräckligt förklarande och saknar exempel till skillnad från Coronas motsvarighet. Både steg för steg exempel för att snabbt komma igång och skapa en ”Hello world” applikation och video finns. Forum med aktiv Community och snabba svar på frågor.

Officiella exempelapplikationer: En hel del, 33st allt från en väldokumenterad ”Hello world” till mer avancerade exempel. Förutom de officiella exemplen finns ”moSync Cookbook” där användare kan dela med sig av exempelkod.

Testkörning och kompilering
Testkörning görs genom moSync SDKs egna emulator som först måste konfigureras för vilken enhet den ska emulera. Enkel kompilering genom valet ”Finalize for this profile”.

3.1.4 PhoneGap

Specifikationer

Plattformar:	iPhone, Android, Blackberry, WebOS, WP7, Symbian och Bada
Licens:	Val mellan New BSD license eller MIT License
Programmeringspråk:	HTML, JavaScript och CSS
Databas:	W3C Web SQL Database

Installation
En inte fullt så omfattande installation som Titanium, men ändå en hel del som behöver installeras och konfigureras. Förutom Android SDK behöver PhoneGap även Eclipse eftersom det används som utvecklingsmiljö. PhoneGap är mer som en plugin till Eclipse då det inte har en egen installation utan binds in som ett Androidprojekt.

Utvecklingshjälp
JavaScript API referens online. Mer liknande dokumentation än en referens då den med detaljerade kodexempel visar hur de flesta funktioner kan implementeras. Förutom API finns en Wiki med dokumentation och guider som t.ex. för att skriva en ”Hello World” app. Community består till största del av en Google grupp som till viss del är sämre än konkurrenternas diskussionsforum då det inte är uppdelat i några underkategorier. Gruppen är däremot aktiv med 26,944 inlägg.

Officiella exempelapplikationer: Inga att importera rakt av, däremot finns det en del kodexempel som utvecklaren själv kan förädla.

Testkörning och kompilering
Testkörning görs precis som på Titanium genom Androids egna emulator som först måste konfigureras. Det går även att testköra direkt på enheten vilket är mycket smidigt. Enkel och snabb kompilering genom export (un)signed application package.

3.1.5 RhoMobile Rhodes

Specifikationer

Plattformar:	iPhone, Android, Blackberry, Windows Mobile och Windows Phone 7
Licens:	MIT License men även betalalternativ för utökad support.
Programmeringspråk:	Ruby, HTML, JavaScript och CSS
Databas:	Rhom (översätts till SQLite eller HSQLDB beroende på plattform)

Installation

Jag misslyckades att köra applikationer genom Android emulatorn efter installation och konfigurering. Då problemet kvarstod även efter ominstallation av verktyget resultera det i att jag inte gick vidare med det p.g.a. tidsbrist.

Utvecklingshjälp
Välskrivna guider för att snabbt komma igång med utvecklingen. En Google grupp som Community som är aktivt. Just nu 16,834 inlägg.

Officiella exempelapplikationer: Några stycken, mest intressant är ett API exempel som enligt beskrivningen ska demonstrera alla funktioner.

Testkörning och kompilering
Ej utfört p.g.a. problem med installationen.

3.1.6 AirPlay SDK[3]

Specifikationer

Plattformar:	iPhone, Android, Symbian, WebOS och Bada
Licens:	Utvärderingsperiod på 30dagar, därefter ett antal betalalternativ beroende på val av distributionsplattformar.
Programmeringspråk:	C, C++
Databas:	SQLite

Installation:
Endast AirPlay SDK behöver installeras.

Utvecklingshjälp:
Omfattande API referens med beskrivningar för varenda funktion. ”Getting Started” guider för att snabbt komma igång och skriva en ”Hello world” applikation. Ett officiellt forum som är lättnavigerat med underkategorier och aktiva medlemmar.

Officiella exempelapplikationer: Många API exempel som demonstrerar hur funktionerna kan användas. Även mer avancerade exempel finns som t.ex. grafiska spel.

Testkörning och kompilering
Ej utfört p.g.a. licens med utvärderingsperiod.

3.2 Val av utvecklingsverktyg efter kartläggning

Alla ovanstående verktyg uppfyllde det första kriteriet, dvs. det skulle stödja utveckling för både iPhone och Android. Alla verktyg förutom moSync hade en dokumenterad databaskoppling, vilket resultera i att moSync valdes bort som kandidat. Då AirPlay SDK vid kartläggningen hade en utvärderingsperiod på endast 30 dagar vilket var mindre än min beräknande tid så valdes även det bort. Rhodes fick jag aldrig att fungera riktigt, och gick därför vidare med kartläggningen.

Kvar från listan var Titanium, Corona SDK och PhoneGap. Efter en översiktlig utvärdering av verktygen utifrån specifikationer och testkörning av

[3] Från juni 2011 heter produkten Marmalade istället för AirPlay SDK vilket gör verktygets kartläggning föråldrad.

exempelapplikationer valdes PhoneGap och Corona SDK ut för fortsatt utveckling. Appcelerator Titanium valdes bort p.g.a. tidsbegränsning.

3.3 Utveckling med Android SDK, PhoneGap och Corona SDK

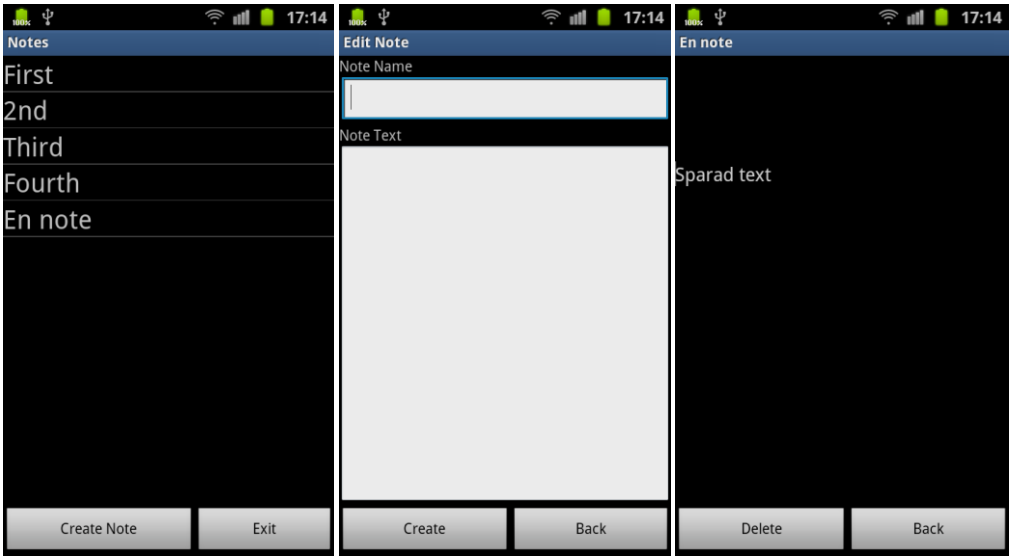
Under utvecklingen har verktygen jämförts åt vid implementationen av databaskoppling och gränssnitt. Då det går att implementera och utforma dessa på flera olika vis så har officiella exempel från deras respektive hemsida använts som utgångspunkt för att utvecklingsprocessen skulle bli så likartad som möjligt.

3.3.1 Gränssnittsutformning med Android SDK

Eftersom applikationen utgick ifrån Android Developers Notepad Tutorial [16] är det mesta av gränssnittet redan utformat. De ändringar som gjorts från exempelapplikationen är för att vara säker på att applikationen vore plattformsoberoende om det var möjligt. Detta har åstadkommits genom att sköta navigeringen med knappar längst ner på skärmen istället för att som i guiden använda menyknappen på telefonen. Även användningen av long-clicks har tagits bort vilket resulterar i att funktionen för att ta bort noten nu ligger direkt på vyn som visar noten.

Utformningen av gränssnittet görs huvudsakligen i XML. Den slutgiltiga utformningen blir då att vi genom XML har definierat tre vyer:

1. List-vyn: *notes_list.xml* med lista på noterna samt knapparna ”Create Note” och ”Exit” (figur 3).
2. Skapa-vyn: *notes_edit.xml* som är lite omgjord från guiden genom att den nu bara kan lägga till noter (figur 4).
3. Not-vyn: *notes_view.xml* som visar den valda noten och dess information samt med valen ”Delete” och ”Back” (figur 5).



Figur 3,4,5. Android SDK: List-vyn, skapa-vyn och not-vyn.

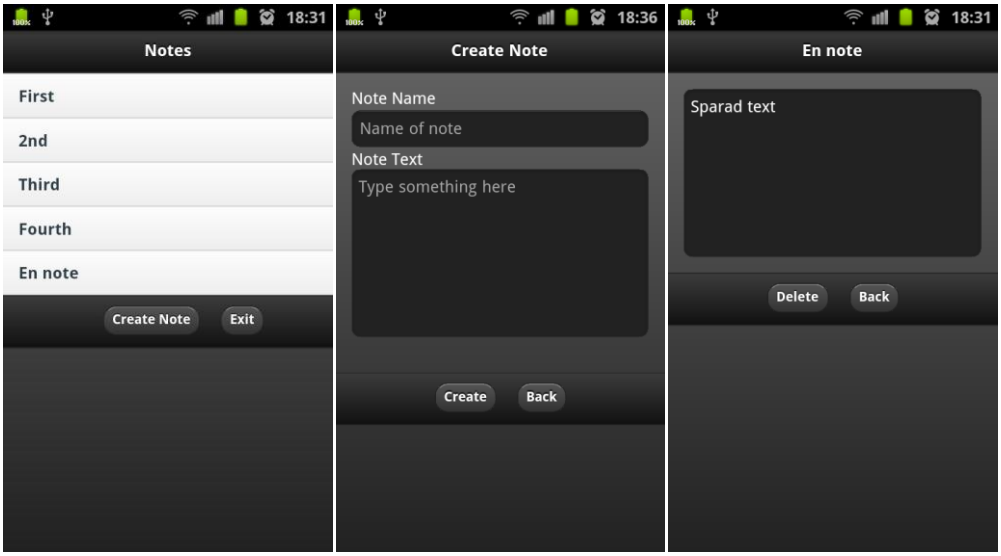
Navigeringen sköts genom *new Intent()* och därefter definieras XML-vyn med exempelvis *setContentView(R.layout.note_view).*

3.3.2 Gränssnittsutformning med PhoneGap

Gränssnittet har skapats av ren HTML samt funktioner och CSS från ramverken jQuery och jQueryMobile. Dessa tillhandahåller ett snyggt och flexibelt gränssnitt för utvecklaren, samt en hel del betydelsefulla och användbara funktioner för mobila applikationer. Ett exempel är navigeringsfunktionen *$mobile.changePage* vilken

ändrar vy med en popup-effekt liknande den som ofta ses på andra nativa applikationer. Gränssnittet består av tre HTML-dokument som bildar olika vyer:

1. List-vyn: *Index.html* visar en översiktsvy med lista på valbara noter, samt knapparna ”Create Note” och ”Exit” (figur 6).
2. Skapa-vyn: *Add.html* visar en vy där användaren lägger till noter med hjälp av olika inmatningsfält, samt knapparna ”Create” och ”Back” (figur 7).
3. Not-vyn: *Note.html* visar vyn för den valda noten samt knapparna ”Delete” och ”Back” (figur 8).



Figur 6,7,8. PhoneGap: List-vyn, skapa-vyn och not-vyn

Nedan visas funktionen för att byta till not-vyn efter att användaren har valt en av de inlagda noterna:

```
function gotoNote(id) {
//Spara ID för senare användning
selectedID = id;
//Ändra sida till note.html
$.mobile.changePage('note.html', 'pop',false, false);
}
```

3.3.3 Gränssnittsutformning med Corona SDK

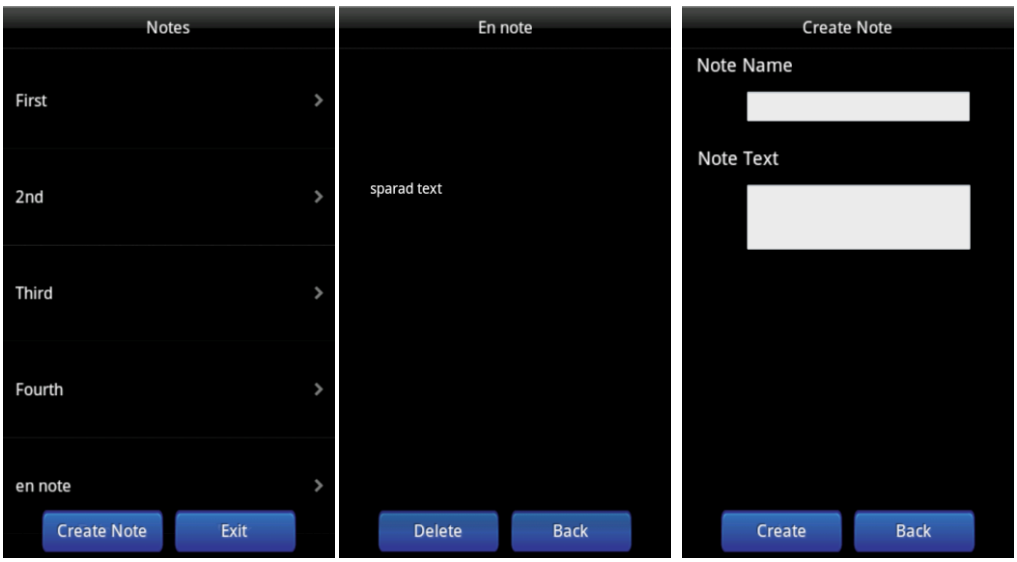
Då Corona enbart använder sig av LUA som utvecklingsspråk blir det en hel del kod att skriva för att åstadkomma det som PhoneGap med jQuery åstadkom med några få rader. Vyerna skapas genom att vi ritar upp objekten från grunden och därefter lägger in dessa i s.k. display groups[4]. Nedan visas hur en lista med noterna skapas och sedan stoppas in i displaygruppen *listScreen* (figur 9):

[4] En typ av behållare för objekt i Corona SDK.

```
listScreen = display.newGroup()
function createList()
    if myList then
        myList:removeSelf()
    end

    myList = tableView.newList{
        data=noter,
        default="listItemBg.png",
        over="listItemBg_over.png",
        onRelease=listButtonRelease,
        top=display.screenOriginY + 45,
        bottom=display.screenOriginY - 25,
        backgroundColor={0, 0, 0 },
        height = 12,
        callback=function(row)
        local t = display.newText(row.noteName, 0, 0,
                  native.systemFontBold, 16)
                  t:setTextColor(255, 255, 255)
                  t.x = math.floor(t.width/2) + 12
                  t.y = 46
        return t
    end
    }
listScreen:insert(myList)
end
```

Ungefär på samma vis skapas not-vyn (figur 10) och skapa-vyn (figur 11), fast istället för *tableView* fyller vi displaygruppen med ett rektangelobjekt istället. Funktionen *transition.to()* används för att navigera mellan vyerna, vilken kan liknas med Android SDKs *new Intent()*.



Figur 9,10,11. Corona SDK: List-vyn, not-vyn och skapa-vyn.

3.3.4 Gränssnittsutformning - sammanfattning

Att utforma gränssnitt i Android SDK är relativt enkelt eftersom det finns ett flertal olika visuella verktyg för gränssnittsutformning av XML, t.ex. genom Eclipse. Genom att koppla aktiviteterna till olika vyer blir uppdelningen mellan logik och gränssnitt mycket överskådlig, vilket underlättar utvecklingen.

Liknande det vi gör med XML i Android SDK gör vi med HTML och CSS i PhoneGap. Det finns även en uppsjö olika HTML-utvecklingsverktyg vilket gör det lätt att skapa snygga gränssnitt. Genom användning av jQuery blir sidorna levande och interaktiva vilket är viktigt för en mobilapplikation. Förutom det tillhandahåller det även en CSS-mall som gör att de visuella objekten ser bra ut med minimalt med kod för utvecklaren, och även lätt kan ändra färgschema genom ett enkelt attributsbyte.

Till skillnad från de övriga finns det inga verktyg till Corona för designutveckling, så all grafik måste ritas upp med LUA-kod. Ett genomgående tema när gränssnittsutveckling görs här är att det krävs flera rader källkod för att göra vad som kan tyckas vara en enkel sak. Det blir snabbt rörigt i koden redan efter skapning av ett tomt skal med bakgrund och några knappar med text.

Sammanfattningsvis är min slutsats att gränssnittsutformning är lättast att göra i PhoneGap just p.g.a. möjligheterna med HTML och CSS. Besvärligast är det i Corona eftersom det inte finns några designverktyg, samt att det jämförelsevis är fler rader källkod som behöver skrivas för att skapa t.ex. en knapp jämfört med samma objekt i HTML.

3.3.5 Databaskoppling med Android SDK

Android använder sig av SQLite för att manipulera data. Då exemplet tillhandahåller en färdigskriven databasadapter var det onödigt att skriva en egen. Genom den sköts alla transaktioner vilket gör uppdelningen av databaslogik och annan logik lätt överskådlig. Nedan visas funktionen *createIt()* som skapar databasen genom *SQLiteDatabase* när applikationen körs:

```
private SQLiteDatabase mDb;
private static final String DATABASE_CREATE = "create table notes
(_id integer primary key autoincrement, " + "title text not null,
body text not null);";
public void createit() {
    mDb.execSQL("DROP TABLE IF EXISTS notes");
    mDb.execSQL(DATABASE_CREATE);
}
```

Därefter är det möjligt att nå de olika funktionerna genom klassen *mDbHelper*, som demonstreras här i funktionen *fillData()* där vi hämtar namnen på noterna och visar dessa i list-vyn:

```
private void fillData() {
        Cursor notesCursor = mDbHelper.fetchAllNotes();
        startManagingCursor(notesCursor);
        String[] from = new String[]{NotesDbAdapter.KEY_TITLE};
        int[] to = new int[]{R.id.text1};
        SimpleCursorAdapter notes =
               new SimpleCursorAdapter(this, R.layout.notes_row,
               notesCursor, from, to);
        setListAdapter(notes);
}
```

3.3.6 Databaskoppling med PhoneGap

Eftersom PhoneGap skapar webbapplikationer[5] så används en databas som kallas "Web SQL Database". Det är en databas baserad på SQLite som fungerar på de webbläsare som använder sig av renderingsmotorn WebKit [28].

Jag utgick från PhoneGaps egna databasexempel i JavaScript och skapade en *databaseAdapter.js* fil. Målet var att ha en plats där enbart all databaslogik skulle ske precis som i Androids SDK, men det visade sig att det blev enklare att urskilja och dessutom mindre kod genom att blanda logiken istället, särskilt p.g.a. jQuery. Nedan är ett exempel för *onLoad()* funktionen där både databasen och tabellerna skapas om den inte redan är skapad:

```
//Skapa DB Handtag
 db = openDatabase(shortName, version, displayName,maxSize);
 db.transaction(function(tx){
//Skapa tabell
   tx.executeSql( 'CREATE TABLE IF NOT EXISTS Notes(NotesId
   INTEGER NOT NULL PRIMARY KEY, NotesName TEXT NOT NULL,
   NotesText TEXT NOT NULL)',
   [],nullHandler,errorHandler);},errorHandler,successCallBack);
 ListDBValues();
 }
```

Genom användning av tidigare nämnda jQuery för bl.a. händelsehantering och interaktion uppdateras listan med noteringar. Exemplet nedan visar hur uppdateringsfunktionen fungerar genom att uppdatera *myNotes* elementet i list-vyn med resultatet från SQL-anropet *Select*:

```
db.transaction(function(transaction) {
   transaction.executeSql('SELECT * FROM Notes;', [],
     function(transaction, result) {
      if (result != null && result.rows != null) {
        for (var i = 0; i < result.rows.length; i++) {
              var row = result.rows.item(i);
              $(".myNotes").append('<li data-icon="false"
              onClick="gotoNote(id);" id="' + row.NotesId + '">
              <a href="#">' + row.NotesName + '</a></li>');
              }
         $(".myNotes").listview('refresh');
      }
     },errorHandler);
 },errorHandler,nullHandler);
 return;
}
```

3.3.7 Databaskoppling med Corona SDK

Corona använder SQLite som databas och LuaSQLite för att manipulera den. SQLite är inbyggt i både iPhone och Android vilket gör att det fungerar likadant. För att skapa databaskopplingen utgick jag ifrån exempelapplikationen som följde med installationen. En fördel med Corona vid utformningen av databaskopplingen är att det effektivt och smidigt går att separera den ifrån gränssnittet då vi kan nå funktioner och variabler globalt. Som vanligt måste databasen först skapas och definieras för att vara arbetbar:

[5] En applikation beroende av en webbläsare för att vara körbar.

```
//Inludera
require "sqlite3"
//Öppna eller skapa data.db samt tabeller
local path = system.pathForFile("data.db",
system.DocumentsDirectory)
db = sqlite3.open( path )
local tablesetup = [[CREATE TABLE IF NOT EXISTS notes (NotesID
INTEGER PRIMARY KEY, NotesName, NotesText);]]
//Kör transaktionen
db:exec(tablesetup)
```

För att därefter uppdatera list-vyn som vi tidigare gjorde i gränssnittsutformningen behöver vi bara ändra variabeln i *tableView.newList* med data som vi kallade *notesData*:

```
//Definiera variabler
local notesData = {}
local i = 0
//Loopa igenom notes och sätt data
for row in db:nrows("SELECT * FROM notes") do
notesData[i] = {}
    notesData[i].noteName = row.NotesName
    notesData[i].noteID = row.NotesID
    i = i+1
end
```

3.3.8 Databaskoppling - sammanfattning

Databaskopplingskoden är väldigt lik varandra i de olika verktygen. Ett handtag till databasklassen skapas och genom den utförs de olika transaktionerna. Web SQL kräver lite mer kod än vad de övriga gör, då transaktionskoden även innehåller funktioner för felmeddelningshantering osv.

Databasmanipulation genom LuaSQLite är också enkelt då det krävs minimalt med kod för att utföra transaktioner. Corona SDKs klass *SQLite3* liknar Android SDKs *SQLiteDatabase*, transaktionerna utförs med funktionen *:exec('SQL Statement')* vilken fungerar liknande *ExecSQL('SQL Statement')* i Android.

3.4 Prestandamätning

Prestandamätningar har gjorts genom att mäta tiden innan och efter exekveringen av en funktion, och därefter avläsa tiden i logcat. Alla verktyg använder olika funktioner för att åstadkomma loggningen, men den fungerar i princip likadant. Resultaten är gjorda på applikationen installerad på smartphonen och genom att före och efter varje mätning installera om applikationen samt tömma RAM-minnet på enheten. Detta för att det inte ska finnas kvar data från applikationen i minnet, vilket annars kan påverka mätningarna.

3.4.1 Notapplikation skapad med Android SDK

Genom funktionen *System.currentTimeMillis()* har mätningar gjorts i den Android skapade notapplikationen. Databasmätningen har gjorts genom att mäta tiden innan *SQLiteDatabase execSQL()* och sedan logga tiden efter funktionen körts.

Gränssnittet har mätts genom att starta mätning precis innan vyn byts med *Intent i = new Intent(this, NoteView.class)*. Därefter har tiden loggats efter *onCreate()* funktionen har körts klart i den startade aktiviteten.

Resultatet för mätningen av exekveringstider kan ses i tabellen i bilaga 1. De slutsatser vi kan dra ifrån tabellen är att databasfunktionerna är mycket snabba och exekveringstiderna ligger ungefär på samma nivåer under alla körningar. Gränssnittsmätningarna visar också på snabba resultat vid bytet av vy, vilket gör att applikationen känns responsiv.

3.4.2 Notapplikation skapad med PhoneGap

Applikationen skapad genom PhoneGap har mätts med hjälp av funktionen *+new Date()*. Databasmätningen har gjorts genom att mäta tiden innan *db.transaction()* körs och därefter vänta på ett lyckat svar och logga tiden genom *Console.log()*.

Gränssnittet har mätts genom att börja mätningen innan vyn byts med *$.mobile.changePage* funktionen och därefter har det med hjälp av jQuery bundit en funktion till *$('#noteView').live('pageshow')* som loggar tiden när sidan visas, liknande *onCreate()* för Android.

Resultatet för mätningen av exekveringstider kan ses i tabellen i bilaga 2. I databasmätningen kan vi se en liten skillnad på runt 100ms för den totala mätningen. Den post som utmärker sig här är ”Ta bort en not” som utförs avsevärt långsammare genom JavaScript med anrop till Web SQL databasen. Gränssnittsmätningen visar stora skillnader på exekveringstiden för navigeringen. Det tog cirka en halv sekund att byta vy vilket gör att applikationen inte känns lika responsiv som den som utvecklades i ren Android-miljö.

3.4.3 Notapplikation skapad med Corona SDK

Coronas tider mättes med funktionen *system.getTimer()*. Databasmätningen har därefter gjorts genom att logga tiden efter avslutad transaktion med *db:exec()*.

Gränssnittsmätningen är gjord genom att starta tidtagningen i *listButtonRelease()* respektive *createButtonPress()* med kopplingar till funktionen för byte av displaygrupp. Funktion skapar även displaygruppen och loggningen görs när den visas.

Resultatet för mätningen av exekveringstider kan ses i tabellen i bilaga 3. Exekveringstiden för databasmätningen är ungefär likadan som i PhoneGap. Precis som där är det ”Ta bort not” anropet som är långsammare. Gränssnittsmätningen visar att tiden för ändring av vy däremot är mycket snabb, vilket mer liknar Androids motsvarighet.

3.4.4 Prestandamätning - sammanfattning

Sammanfattningsvis har notapplikationerna mycket liknande resultat när det kommer till exekveringstiden för databastransaktionerna. Tabell 1 visar sammanfattning av resultaten. Mer detaljerade resultat finns i bilaga 1,2 och 3.

Det mest anmärkningsvärda är att tiden för att ta bort en not ur databasen skiljer sig markant då transaktionen i Android SDK tar cirka 100ms mindre än de övriga. Då de andra databastiderna är liknande för de olika applikationerna och genom försäkran på att noten är borttagen med testkod kan det konstateras att i just det här fallet är notapplikationen utvecklad i Android SDK snabbast på att ta bort en rad ur databasen. Utöver det kan vi även dra slutsatsen att Corona SDK är långsammast på att ”Skapa fyra noter”, då den exekveringstiden är snäppet högre än de andra.

I mätningen av gränssnittet visar det tydligt att PhoneGap är långsammare än de övriga när det laddar in ett nytt HTML-dokument genom *$mobile.changePage*. Tidsskillnaden mellan Corona SDK och Android SDK är väldigt liten och borde inte uppmärksammas av användaren av applikationen.

Medeltid (ms)	Android SDK	PhoneGap	Corona SDK
DB: Skapa tabell	56	71	65
DB: Skapa fyra noter	213	208	255
DB: Ta bort en not	24	123	127
UI: Notval	112	550	156
UI: Skapa not	67	558	59
DB+UI Totalt	**472**	**1510**	**662**

Tabell 1. Sammanfattning av prestandamätning.

3.5 Minnesanvändning

Då minnesanvändningen i Linuxliknande operativsystem som Android är ett mycket komplicerat område är resultaten bara en approximation. Däremot borde vi kunna dra slutsatser om vi kan se att skillnaderna mellan minnesanvändningen är stora. Se tabell 2 för resultat av Pss- och Uss-värden från de tre applikationerna under körning efter en not är skapad. Alla värden från respektive applikations minnesdump kan ses i bilaga 4,5 och 6.

Det vi kan se är att skillnaden mellan PhoneGap applikationen och de andra är mycket stor, samt att en applikation utvecklad i ren Android-miljö tar minst plats. PhoneGaps höga resultat kan delvis bero på WebView komponenterna samt användningen utav jQuery.

Genom att bara köra applikationen med jQuery utan JavaScript kan minnesanvändningen minskas till runt hälften, vilket fortfarande är mer än de övriga. Alltså är det inte troligt att det är den egna koden som skapar den stora skillnaden, även om det är möjligt att det finns en minnesläcka någonstans.

Minne (kB)	Android SDK	PhoneGap	Corona SDK
Pss	3716	26140	8094
Uss	2904	21556	5420

Tabell 2. Minnesanvändning.

4 Diskussion

Då ämnet täcker ett ganska stort område finns det en hel del att diskutera. Mina metodval är inte självklara, utvärderingen hade självfallet kunnat göras på andra sätt. Diskussionen besvarar de val jag gjort samt de frågor som uppstått under processen. Även en sammanfattning av mina åsikter efter utvecklingen om verktygens fördelar och nackdelar har tagits upp.

4.1 Android/iOS

Det fanns flera anledningar till att jag valde att bara fokusera på Android-plattformen. För flera av verktygen var man tvungen att använda Mac OS X som operativsystem och/eller inneha ett utvecklarkonto hos Apple, vilket kostar 99$ per år. Förutom det så fanns det helt enkelt inte tid till att sätta mig in i utveckling för båda miljöerna, eftersom det blev dubbelarbete med installationer och konfigurering. Genom att Android SDK är gratis vägde det tyngst i valet mellan dessa plattformar.

4.2 Applikationsval

Förutom att spara data lokalt i databasen på smartphonen ville jag även synka databasen till en server för att externt spara data liknande det som beskrivs i ”Smartphone Enterprise application integration” [19]. Men eftersom det kan bli ett andra steg avgränsade jag mig till att för tillfället enbart arbeta med en lokaldatabas utan synkning. Valet föll på ”Android developers” Notepad-exempel [16] eftersom det var det bästa databasexemplet jag hittade samt även hade en rimlig omfattning.

För att vara säker på att det inte skulle bli några problem att replikera den i de andra verktygen valde jag att ta bort funktioner som teoretiskt kan bli ett problem (t.ex. användningen av menyknappen). Detta gjordes även för att applikationen skulle bli multiplattformsvänlig genom att enbart kräva en tryckkänslig skärm.

4.3 Prestandamätning och minnesanvändning

Båda snabba databastransaktioner och ett responsivt gränssnitt är viktigt då de hänger ihop med varandra. De mätpunkter jag gjorde kan diskuteras, men valet föll på att mäta tiden för att initiera databasen och skapa fyra rader i databasen, samt sedan ta bort en rad då jag ansåg det mer representerar ett verkligt användningsfall.

Det hade varit intressant att även göra en användarstudieundersökning på hur applikationerna upplevs av användare, då det skulle ge möjlighet att se om resultaten från gränssnittsmätningen också speglar deras uppfattning. Eftersom det inte fanns tid för det så har jag nöjt mig med att konstatera att enligt min subjektiva upplevelse så finns det ett samband.

Metoden för mätning av minnesanvändning är baserad på svaret till frågan ”How to discover memory usage of my application in Android” från Stackoverflow [24]. Det framkommer dock att det är svårt att tyda minnesvärderna, och att det inte är helt klart hur mycket minne en process tar upp då en del av minnet är delat med andra processer. Jag tolkar svaret som att det mest korrekta värdet att kolla på är Uss vilket även förespråkas på både Texas Instrument [27] och Elinux [25], och därefter dragit mina slutsatser utifrån det.

4.4 Kartläggning och verktygsval för utvärdering

Kartläggningen av verktygen gjordes i två omgångar, vilket inte var optimalt då detta är ett område under snabb utveckling. AirPlay SDK ändrade t.ex. namn till Marmalade under kartläggningen och deras regler för licens blev generösare vilket möjligtvis hade resulterat i att AirPlay SDK utvärderats istället för Corona SDK om kartläggningen utförts senare. Jag fick även igång Rhodes RhoMobile efter att ha laddat hem en nyare version och installerat om Ruby, vilket även det kunde ändrat utgången av val av verktyg för utvärdering.

Många av verktygen är intressanta, särskilt moSync då jag gillade deras IDE för utveckling samt den interna emulatorn. Tyvärr hade det inte någon dokumenterad databas med exempel. Då PhoneGap och Titanium Appcelerator fungerar liknande varandra genom att göra en applikation av HTML-kod var det känslan som avgjorde, och den föll på PhoneGap eftersom integrationen i Eclipse IDE var bra. Jag tror utvärderingen av Titanium hade blivit mycket lik PhoneGaps när det kommer till gränssnittsutformning och databaskoppling.

4.5 Gränssnittsutformning och prestandamätningar

Det är möjligt att åstadkomma mer eller mindre likadana visuella gränssnittsresultat på olika sätt i de olika verktygen. I PhoneGap baserade jag applikationen på ett exempel med jQuery som använde ett annat HTML-dokument för att generera en ny vy med *changePage*. Det hade varit möjligt att få samma resultat med endast ett dokument genom att dölja/visa objekt och detta hade möjligtvis resulterat i snabbare mättider.

PhoneGap källkoden bör kunna kompileras i Titanium med några små modifieringar. Det kunde varit intressant att undersöka prestanda- och minnesmätningen i Titanium också, eftersom jag är osäker på om den kommer vara samma.

I Corona SDK baserade jag applikationen på exemplet ”List-View”. Här är jag däremot osäker på om det är möjligt att få en större skillnad på mättiderna genom användning av ett annat designalternativ. För mycket ändringar gör även att applikationen inte liknar den ursprungliga Notepad applikationen.

4.6 Verktygens lämplighet för utveckling av liknande applikationer

PhoneGap var absolut enklast att arbeta med då det är enkelt att designa gränssnitt samt att det finns mycket information om HTML och JavaScript. jQuery gjorde det enkelt att göra applikationen interaktiv med få rader källkod. Genom att integrera PhoneGap med Eclipse IDE så kunde applikationen testas precis som vanligt. Nackdelen med PhoneGap var att notapplikationen var slöare än de andra, vilket märktes främst vid navigationen till not-vyn.

Sammanfattningsvis anser jag att PhoneGap lämpar sig bra till att utveckla liknande applikationer då det går snabbt att få ett resultat genom HTML och JavaScript. Om prestanda däremot är det viktigaste för utvecklaren rekommenderar jag Corona SDK.

Corona SDK saknar dock en egen eller integrerad IDE, vilket resulterade i att källkoden skrevs i tredjepartsprogrammet Notepad++. Coronas egna hemsida erbjöd många ingående exempel vilket gjorde att det var lätt att påbörja utvecklingen, men utöver det finns det väldigt få andra källor att tillgå för information. En nackdel med Corona är att det tar längre tid att utforma gränssnittet än med PhoneGap, eftersom det krävs fler rader källkod samt att inga tredjepartsverktyg för designutveckling finns att tillgå. Testningen gick enkelt då den tillhandahåller en egen emulator, dock så fungerade vissa funktioner inte i den som t.ex. en editbox. Detta var en stor nackdel då

applikationen först var tvungen att kompileras för att testa dessa funktioner vilket tog längre tid än i PhoneGap.

Sammanfattningsvis så anser jag att även Corona SDK lämpar sig för dessa typer av applikationer, då verktyget producerar en applikation med liknande prestanda som en utvecklad i ren Android-miljö. Men eftersom PhoneGap är det utvecklingsverktyg som kräver minst tid för att utveckla en liknande applikation anser jag att det är bäst lämpat för enklare applikationer som använder sig av en databas. Dessutom är det gratis.

4.7 Rekommendation för fortsatt utvärdering

Eftersom applikationerna kan kompileras till flera olika plattformar genom verktygen rekommenderas det att göra detta även till iOS, samt att konstruera en liknande applikation genom iOS SDK och utvärdera dessa. Kommer resultaten att vara liknande, dvs. kommer applikationen byggd i iOS SDK vara snabbare än de övriga?

Det rekommenderas även att göra en användarstudieundersökning på de olika applikationerna, vad tycker användarna? Speglar resultaten prestandamätningarna?

Eftersom PhoneGap och Titanium liknar varandra genom att de båda tillverkar applikationer av HTML hade det varit intressant att jämföra deras prestanda.

Utöver detta kan applikationen utvecklas med t.ex. en funktion för att synka den lokala databasen med en central, eller mer avancerade funktioner, för att undersöka verktygens begränsningar.

5 Slutsatser

Slutsatser jag dragit från resultaten är att det stämmer att apparna som utvecklats genom multiplattformsutvecklingsverktygen överlag har sämre prestanda än den som utvecklades med Android SDK. Exekveringstiden för databaslogiken visar att alla tre verktyg har liknande resultat, förutom på mätpunkten ”Ta bort en not”, där är Android SDKs applikation snäppet snabbare. Resultaten av gränssnittets prestanda visar att PhoneGaps applikation är väsentligt långsammare än de övriga verktygens, och Android SDKs och Coronas SDKs motsvarighet har jämförbar prestanda.

Även på minnesområdet är Androids SDKs applikation bättre genom att använda mindre minne än de övriga. Om vi bara analyserar det minne som garanterat frigörs om processen avslutas så använder Corona SDKs applikation ungefär dubbelt så mycket som Androids SDKs motsvarighet. PhoneGaps notapplikation använder sju gånger så mycket minne som Androids SDKs.

Verktygens utvecklingsprocess för databasimplementationen är lik varandra genom att ett handtag till databasklassen skapas och genom den utförs transaktionerna. Gränssnittsutformningen skiljer sig åt väsentligt då de i PhoneGap (HTML/CSS) och Android SDK (XML) är möjligt att använda tredjeparts program för att designa gränssnittet. Det är inte möjligt i Corona SDK, vilket gör gränssnittsutformning tidskrävande.

Kartläggningen resulterade i att valet av verktyg föll på PhoneGap och Corona SDK, då de passade bäst in på kriterierna i kapitel 2.4. Båda verktygen visade sig även vara väl lämpade till dessa typer av applikationer, dock var de inte perfekta. PhoneGaps stora fördel är att utvecklingsprocessen är snabb då gränssnittsutformningen är enkel med HTML och CSS. Nackdelen var dålig prestanda. För Corona SDK var resultat det omvända.

6 Referenser

[1] Canalys, “Google’s Android becomes the world’s leading smart phone platform” (Jan 2011) [Online] Available: http://www.canalys.com/pr/2011/r2011013.html

[2] DN.se, ”Alla villa ha appar” (2011) [Online] Available: http://www.dn.se/ekonomi/alla-vill-ha-appar

[3] A. I. Wasserman, "Software engineering issues for mobile application development," in *Proc. of the FSE/SDP workshop on Future of software engineering research*, Santa Fe, New Mexico, USA, 2010, pp. 397-400.

[4] A. Puder, "Cross-compiling Android applications to the iPhone," in *Proc. of the 8th International Conference on the Principles and Practice of Programming in Java*, Vienna, Austria, 2010, pp. 69-77.

[5] P. Biao, X. Kun, and L. Lei, "Component-based Mobile Web Application of Cross-platform," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference*, Bradford, 2010, pp. 2072-2077.

[6] M. D. Bloice, F. Wotawa, and A. Holzinger, "Java's alternatives and the limitations of Java when writing cross-platform applications for mobile devices in the medical domain," in *Proc. of the ITI 2009 31st International Conference*, Dubronvik, 2009, pp. 47-54.

[7] Faruk Ateş, ”Native vs. Web Apps” (2011) [Online] Available: http://farukat.es/journal/2011/03/537-native-vs-web-apps

[8] Martin Fowler, “CrosPlatformMobile” (2011) [Online] Available: http://martinfowler.com/bliki/CrossPlatformMobile.html

[9] RhoMobile’s Rhodes (2011) http://www.rhomobile.com

[11] PhoneGap (2011) http://www.phonegap.com

[10] Appcelerator (2011) http://www.appcelerator.com

[12] moSync (2011) http://www.moSync.com

[13] Ansca Mobile’s Corona SDK (2011) http://www.anscamobile.com/corona/

[14] AirPlay SDK (2011) http://www.airplaysdk.com

[15] Mashable, “5 Cross-Platform mobile development tools“ (2011) [Online] Available: http://mashable.com/2010/08/11/cross-platform-mobile-development-tools/

[16] Android Developers, “Notepad Tutorial“ (2011) [Online] Available: http://developer.android.com/resources/tutorials/notepad/index.html

[17] IEEE Explore (2011)
http://ieeexplore.ieee.org

[18] ACM Digital Library (2011)
http://portal.acm.org

[19] "Smartphone enterprise application integration", white paper, Rhomobile inc., Mars 2011

[20] A. Smith, "Mobile Database Applications – Importance" (2011) [Online] Available:
http://ezinearticles.com/?Mobile-Database-Applications---Importance&id=6129124

[21] Sun Microsystems, "Java performance first edition" [Online] Available:
http://java.sun.com/docs/books/performance/1st_edition/html/JPMeasurement.fm.html

[22] Android developer, "logcat" (2011) [Online] Available:
http://developer.android.com/guide/developing/tools/logcat.html

[23] Hoccer, "Measuring performance in the Android SDK" (2011) [Online] Available:
http://hoccer.com/2010/09/measuring-performance-with-android/

[24] Stackoverflow, "How to discover memory usage of my application in Android" (2011) [Online] Available:
http://stackoverflow.com/questions/2298208/how-to-discover-memory-usage-of-my-application-in-android

[25] ELinux, "Android memory usage" (2011) [Online] Available:
http://elinux.org/Android_Memory_Usage

[26] Android developer, "Android Debug Bridge (ADB)" (2011) [Online] Available:
http://developer.android.com/guide/developing/tools/adb.html

[27] Texas Instruments, "Android memory analysis" (2011) [Online] Available:
http://processors.wiki.ti.com/index.php/Android_Memory_Analysis

[28] The WebKit Opensource project (2011)
http://www.webkit.org/

Bilaga 1. Exekveringstid för Android SDKs notapplikation

Funktion:	K. 1	K. 2	K. 3	K. 4	K. 5	K. 6	K. 7	K. 8	K. 9	K. 10	Medel
DB: Skapa tabell	62	53	46	51	65	52	70	69	44	46	56
DB: Skapa fyra noter	220	224	156	199	196	241	248	208	206	230	213
DB: Ta bort en not	27	26	17	19	33	28	26	19	21	28	24
DB: Total	**309**	**303**	**219**	**269**	**294**	**321**	**344**	**296**	**271**	**304**	**293**
UI: Notval	125	116	123	115	84	113	81	128	117	120	112
UI: Skapa not	81	81	77	48	38	37	76	77	77	77	67
UI: Total	**206**	**197**	**200**	**163**	**122**	**150**	**157**	**205**	**194**	**197**	**179**
DB+UI Total:	**515**	**500**	**419**	**432**	**416**	**471**	**501**	**501**	**465**	**501**	**472**

K = Körning
DB = Databas
UI = Gränssnitt
Alla resultat i ms

Bilaga 2. Exekveringstid för PhoneGaps notapplikation

Funktion:	K. 1	K. 2	K. 3	K. 4	K. 5	K. 6	K. 7	K. 8	K. 9	K. 10	Medel
DB: Skapa tabell	87	77	87	64	53	54	52	92	80	62	71
DB: Skapa fyra noter	204	188	237	208	192	186	189	229	218	227	208
DB: Ta bort en not	122	117	120	124	129	111	117	134	141	114	123
DB: Total	**413**	**382**	**444**	**396**	**374**	**351**	**358**	**455**	**439**	**403**	**402**
UI: Notval	564	573	555	535	546	552	541	552	541	540	550
UI: Skapa not	538	553	561	571	567	598	547	532	568	548	558
UI: Total	**1102**	**1126**	**1116**	**1106**	**1113**	**1150**	**1088**	**1084**	**1109**	**1088**	**1108**
DB+UI Total:	**1515**	**1508**	**1560**	**1502**	**1487**	**1501**	**1446**	**1539**	**1548**	**1491**	**1510**

K = Körning
DB = Databas
UI = Gränssnitt
Alla resultat i ms

Bilaga 3. Exekveringstid för Corona SDKs notapplikation

Funktion:	K. 1	K. 2	K. 3	K. 4	K. 5	K. 6	K. 7	K. 8	K. 9	K. 10	Medel
DB: Skapa tabell	89	72	69	70	63	67	49	48	63	61	65
DB: Skapa fyra noter	266	277	275	261	264	261	201	239	263	245	255
DB: Ta bort en not	133	120	103	129	136	127	100	143	133	142	127
DB: Total	**487**	**468**	**447**	**460**	**463**	**455**	**350**	**430**	**459**	**448**	**447**
UI: Notval	131	166	116	164	175	130	164	174	169	171	156
UI: Skapa not	66	59	59	65	52	51	54	65	54	62	59
UI: Total	**198**	**225**	**175**	**229**	**227**	**181**	**218**	**239**	**223**	**233**	**215**
DB+UI Total:	**685**	**694**	**622**	**689**	**690**	**636**	**568**	**669**	**682**	**681**	**662**

K = Körning
DB = Databas
UI = Gränssnitt
Alla resultat i ms

Bilaga 4. Minnesdump av Android SDKs notapplikation

```
Applications Memory Usage (kB):
Uptime: 36730512 Realtime: 169087508

** MEMINFO in pid 26950 [com.android.demo.notepad3] **
                   native   dalvik    other    total
             size:   3796     5379      N/A     9175
        allocated:   3758     2661      N/A     6419
             free:     21     2718      N/A     2739
           (Pss):   1143      136     2437     3716
  (shared dirty):    916     1968     5344     8228
    (priv dirty):   1108       52     1744     2904

 Objects
           Views:        0        ViewRoots:        0
     AppContexts:        0      Activities:        0
          Assets:        3    AssetManagers:        3
   Local Binders:        5    Proxy Binders:       13
Death Recipients:        0
 OpenSSL Sockets:        0
```

Bilaga 5. Minnesdump av PhoneGaps notapplikation

```
Applications Memory Usage (kB):
Uptime: 1513046 Realtime: 1513039

** MEMINFO in pid 7367 [com.phonegap.testApp] **
                   native   dalvik    other    total
             size:  13560     5959      N/A    19519
        allocated:  10391     3104      N/A    13495
             free:   2216     2855      N/A     5071
           (Pss):    6869      290    18981    26140
  (shared dirty):     840     1956     5000     7796
    (priv dirty):    6852      148    14556    21556

 Objects
           Views:        0        ViewRoots:        0
     AppContexts:        0      Activities:         0
          Assets:        3    AssetManagers:        3
   Local Binders:        6    Proxy Binders:       16
Death Recipients:        1
 OpenSSL Sockets:        0
```

Bilaga 6. Minnesdump av Corona SDKs notapplikation

```
Applications Memory Usage (kB):
Uptime: 57244879 Realtime: 170161284

** MEMINFO in pid 27237 [se.hig.thopeh.corona.notepad] **
                   native   dalvik    other    total
             size:     4636     5959      N/A    10595
        allocated:     4380     3198      N/A     7578
             free:      135     2761      N/A     2896
           (Pss):     2267      859     4968     8094
  (shared dirty):      764     1940     5100     7804
    (priv dirty):     2252      228     2940     5420

 Objects
           Views:        0        ViewRoots:        0
     AppContexts:        0      Activities:         0
          Assets:        3    AssetManagers:        3
   Local Binders:       18    Proxy Binders:       17
Death Recipients:        1
 OpenSSL Sockets:        0
```