Karlstad Business School
Handelshögskolan vid Karlstads universitet

Alexander Hansson

En jämförelse mellan ramverk för att utveckla hybridapplikationer

A comparison between the frameworks to develop hybrid applications

Informatik
C-uppsats

Termin: VT-15
Handledare: Peter Bellström
Examinator: John Sören Pettersson

Karlstad Business School
Karlstad University SE-651 88 Karlstad Sweden

Abstract

Syftet med undersökningen är att jag ska samla kunskap om olika ramverk för att ta fram hybridapplikationer och undersöka hur dessa ramverk presterar mot kriterier som tagits fram tillsammans med Bulldozer kommunikationsbyrå.

Med hjälp av en samling kriterier är målet med undersökningen att komma fram till vilket som är det "bästa" ramverket för att ta fram en hybridapplikation för en webbutvecklare som saknar kompetens för att utveckla plattformsspecifika applikationer. Resultatet ska vägleda målgruppen i vilket ramverk de ska använda sig av när de bestämmer sig för att börja utveckla applikationer.

Datainsamling för undersökningen skedde i två omgångar. Den första var vid urvalet av ramverk och då åtta stycken ramverk blev tre genom en dokumentanalys. Den andra omgången av datainsamling skedde på applikationerna som utvecklades utefter vilka tre ramverk som samlade på sig mest poäng under första datainsamlingen. Hur lätt det kommer att vara att underhålla och sätta sig in i koden undersöktes bara på de skrivna applikationerna och mättes med hjälp av cyklomatisk komplexitet och logiska rader kod.

De ramverk som samlade på sig mest poäng efter första urvalet var Meteor, Phonegap och Titanium. Titanium valde under undersökningens gång att bli kommersiellt och täckte därför inte längre kriterierna och därför utvecklades det ingen applikation för det ramverket. Det utvecklades istället en applikation på ramverket som samlade på sig fjärde mest poäng, Cordova. Av Meteor, Phonegap och Cordova är Phonegap att föredra även om det samlade på sig lika många poäng som Meteor. Detta för att Phonegap är mognare och koden är mindre komplex vilket gör den lättare för utvecklare att sätta sig in i och hantera.

Nyckelord: hybridapplikationer, cross-platform, cyklomatisk komplexitet, webbutveckling, JavaScript, ramverk

Omnämnade

Jag vill tacka Bulldozer kommunikationsbyrå för möjligheten att få samarbeta med dem i denna undersökning. Jag vill även tacka min handledare Peter Bellström som givit mig bra konstruktiv kritik under undersökningens gång.

Innehållsförteckning

Tabellförteckning

Figurförteckning

Begreppsförklaring

Cross-platform development

Begreppet syftar till en applikation som bara har en kodbas men fungerar att använda på flera olika plattformar. En webbapplikation är cross-platform för att den inte behöver skrivas till någon specifik plattform utan kan köras på både iOS- och Android-enheter. Motsatsen till cross-platform är plattformsspecifik, då behövs det skrivas en kodbas per plattform som det skall utvecklas en applikation för. Exempel på detta är en native-applikation.

Native-applikation

Begreppet är en översättning från engelskan "*native application*" som syftar till en applikation som specifikt är skriven för en viss plattform, dvs. plattformsspecifik. Till exempel om du utvecklar en native-applikation för iOS och Android behövs det skrivas en specifik applikation för iOS och en för Android i två olika programmeringsspråk då de ska köras på två olika plattformar.

API (Application Programming Interface)

API är en specifikation hur olika program kan kommunicera med programvara. Till exempel kan ett API ge en utvecklare tillgång till en sportklockas GPS-system.

Open source

Open source innebär att en utvecklare ger vem som helst tillgång till det som denne utvecklat med en licens som säger att vem som helst får använda koden och modifiera den.

GitHub

GitHub är en webbtjänst som erbjuder versionshantering för förvaltning av källkod. Denna förvaringsplats av kod används ofta för open source-projekt. Något som kommer återkomma frekvent i undersökningen är "issues". Ett "issue" är att en användare, som bidrar till projektet eller inte, rapporterar något som de anser är fel eller behöver uppdateras. Därefter så är det antingen upp till den som rapporterade att fixa det eller de som ligger bakom projektet.

Märkspråk

Märkspråk, översättning av engelskans "markup language", syftar till kod som består av taggar och element som inte syns när de renderats. Till exempel ger HTML-kod webbläsaren direktiv hur en webbsida skall renderas.

Stackoverflow

Stackoverflow är en plattform för att fråga frågor och få svar inom programmering. Aktiva medlemmar som svarar på många frågor blir tilldelade "ryktespoäng". Ryktespoäng innebär att om många andra användare anser att ett svar är bra skrivet och röstar på det blir författaren

tilldelad ryktespoäng. Stack overflow ligger idag enligt Alexa Top 500 Global Sites (2015) på plats 60 bland de mest besökta webbsidorna i världen.

Akronymförteckning

HTML	HyperText Markup Language
IDE	Integrated Development Environment
JS	JavaScript
CSS	Cascading StyleSheets
AJAX	Asynchronous JavaScript and XML
LOC	Lines of Code
LLOC	Logical Lines of Code
CK	Cyclomatic complexity/Cyklomatisk komplexitet
iOS	Apples mobila operativsystemet
API	Application Programming Interface
GPS	Global Positioning System
IEEE	Institute of Electrical and Electronics Engineers
MySQL	My Structured Query Language
PHP	Hypertext PreProcessor

1. Inledning

Detta kapitel kommer att gå igenom varför jag har valt att skriva om ämnet som behandlas i undersökningen. Kapitlet beskriver också hur jag gick tillväga i undersökningen.

1.1 Problembakgrund

Smartphone-försäljningen har fördubblats på bara de senaste tre åren. Enligt Statista (2015) såldes första kvartalet 2012 152.7 miljoner enheter och Q3 2014 hade det mer än fördubblats enligt Gartner (2015) och det såldes drygt 350 miljoner smartphones runt om i världen.

Att så många enheter säljs innebär att företag vill utveckla applikationer för att komma nära sina kunder och/eller få nya kunder. Att skriva en plattformspecifik applikation skulle innebära att det skulle behöva utvecklas två versioner av samma applikation om den ska stöjda till exempel både iOS och Android, som är Apples respektive Googles mobila operativsystem. Dessa applikationer är alltså byggda för att passa den enhet som den byggts för. Detta betyder alltså att ett val av vilka plattformar man vill stödja måste tas innan man börjar utveckla (Fling 2009). Även om iOS och Android är de största plattformarna finns det åtskilliga andra operativsystem som, beroende på användarbasen, skulle behöva utvecklas för; Microsofts Windows Mobile, Nokias Symbian, och kanske även RIM som är BlackBerrys operativsystem.

Idag kan man skriva något som kallas en hybridapplikation där man försöker att kombinera fördelarna från webben med fördelarna med de nativa applikationerna (Fling 2009). Dessa applikationer är ofta byggda i HTML, CSS och JavaScript. Fördelen med att skriva en hybridapplikation är att man inte behöver ta hänsyn till vilken plattform man skriver applikationen för. Webbvyn man skriver packas ihop med “native container” som gör att funktionaliteten som man annars saknar hos en webbsida men finns tillgängligt hos en smartphone går att använda. Som exempel så kan man använda sig av telefonens GPS, kamera och accelerometer (Xanthopoulos & Xinogalos 2013).

Det finns idag många olika ramverk som kan tas till hjälp för att ta fram en hybridapplikation och om man inte har jobbat med något innan kan det vara svårt att veta vart man ska börja.

Det finns många studier som undersöker hur webb-, hybrid- och native-applikationer står sig mot varandra eller hur olika ramverk för att ta fram hybridapplikationer skiljer sig åt i slutprodukten. Jag har inte kunnat hitta något studie som avser att jämföra olika typer av ramverk för att ta fram hybridapplikationer där man vill komma fram till vilket som är bäst för utvecklare.

1.1.1 I samarbete med Bulldozer kommunikationsbyrå

Undersökning är gjord för och i samarbete med Bulldozer kommunikationsbyrå. Bulldozer kommunikationsbyrå är ett företag från Karlstad med 14 anställda som håller på med olika typer

av kommunikation men har fokus på digitala medier och film. Bulldozer erbjuder inte idag att ta fram applikationer till sina kunder men det är något som de vill kunna göra i framtiden. Då de redan har anställda webbutvecklare på företaget vill de ta vara på kompetensen som redan finns och se över vilket ramverk för att ta fram hybridapplikationer som passar dem bäst.

Jag är idag anställd hos Bulldozer kommunikationsbyrå men är inte inblandad i något projekt som skulle kunna påverka undersökningen och får inte betalt för något av det arbete som jag utför i undersökningen.

1.2 Syfte

Syftet med undersökningen är att jag ska samla kunskap om olika ramverk för att ta fram hybridapplikationer och undersöka hur dessa ramverk presterar mot kriterier som tagits fram tillsammans med företaget. Kriterierna är:

- hur många stjärnor projektet har på GitHub
- hur många resultat som kom upp vid en sökning på ramverkets namn på Stackoverflow
- hur aktivt underhållet ramverket är (totala issues mot hur många lösta issues från GitHub)
- hur bra dokumentationen uppfyllde IEEE Computer Society (2001) punkter på vad en dokumentation bör innehålla
- logiska rader kod som krävdes för att skriva applikationen
- komplexiteten i koden

Med hjälp av kriterierna är målet att komma fram till hur väldokumenterat, underhållet, hur mycket hjälp det finns att hitta på internet, hur lätt det är att sätta sig in i koden och hur lätt det är att underhålla koden. Resultatet från detta är det som kommer motivera vilket ramverk som är "bäst" och detta ska vägleda målgruppen i vilket ramverk de ska använda sig av när de bestämmer sig för att börja utveckla applikationer

1.3 Målgrupp

Målgruppen för rapporten är företag och webbutvecklare som i dag är intresserade av att börja utveckla applikationer men inte har kompetensen att utveckla plattformsspecifikt utan vill nyttja dagens ramverk för att ta fram hyrbridapplikationer.

1.4 Undersökningsfråga

Vilket är det "bästa" ramverket för att ta fram en hybridapplikation för en webbutvecklare som saknar kompetens för att göra plattformsspecifika applikationer?

1.5 Avgränsning

För ramverken som ska utvärderas ska applikationerna skrivas i HyperText Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, Hypertext PreProcessor (PHP) och/eller

MySQL (Structured Query Language). NodeJS och MongoDB går också bra då det skrivs i JavaScript. Det ska använda sig av Cordova (ett API för att komma åt funktionaliteten i smartphonen), samt att det ska vara gratis och open source.

Då rapporten inriktar sig till företag och webbutvecklare som är intresserade att utveckla hybridapplikationer kommer undersökningen inte att behandla vilken av webb-, hybrid- och nativeapplikationer som är bäst lämpade för olika tillfällen utan endast vilket ramverk som är "bäst" för att ta fram en hybrilapplikation.

I undersökningen är det endast JavaScript-koden som kommer att utvärderas. Detta för att HTML och CSS endast representerar applikationen visuellt och har inget med själva funktionaliteten i applikationen att göra och därför kommer detta att inte mätas.

1.6 Metod

1.6.1 Första urvalet

Enligt kriterier som togs fram tillsammans med företaget började jag med att, av åtta ramverk som företaget valt ut, göra en urgallring av dessa åtta för att komma fram till tre stycken som sedan skulle jämföras. Att den första urgallringen skedde var för att jag inte hade möjlighet att skriva applikationer för alla åtta ramverk. De åtta som företaget valde ut har tagits fram efter kriterierna:

- applikationen skulle skrivas i HTML, CSS, JavaScript, PHP och/eller MySQL. *(NodeJS och MongoDB gick också bra då det skrivs i JavaScript)*
- använde sig av Cordova *(ett API för att komma åt funktionaliteten i telefonen)*
- gratis
- open source

De åtta som togs fram utvärderades enligt fyra punkter:

- hur många stjärnor projektet hade på GitHub (GitHub 2015a-g)
- hur många resultat som kom upp om man sökte på ramverkets namn på Stackoverflow[1]
- hur aktivt underhållet ramverket var (GitHub 2015a-g)
- hur bra dokumentationen uppfyllde IEEE Computer Society (2001) punkter på vad en dokumentation bör innehålla.

Punkt ett avsedde att se hur populärt ramverket är genom att mäta hur många stjärnor projektet hade på GitHub.

[1] Datan hämtades från en sökning på Stackoverflow där parametern för sökningen var namnet på ramverket. Exempel på hur en URL såg ut var: *http://stackoverflow.com/search?q=meteor*. Datan hämtades 2015-02-20.

Stackoverflow är en plattform för att fråga frågor och få svar inom datorprogrammering och ligger idag på på plats 60 av världens mest besökta webbsidor (Alexa Top 500 Global Sites 2015) och avsedde att se hur mycket hjälp det redan finns att hämta från Internet utöver ramverkets egen dokumentation.

Den tredje punkten är hur aktivt underhållet ramverket var och detta analyserades genom att se till hur aktiviteten på GitHub såg ut. Detta gjordes genom att se till hur många totala issues som fanns mot hur många issues som var lösta.

Sista och fjärde punkten är hur väldokumenterat ramverket var och detta utvärderades mot IEEE Computer Society (2001) punkter om vad som bör finnas i en dokumentation. Att dessa punkter används från IEEE är för att de är en organisation som är med och sätter industristandader för sådant som användardokumentation (IEEE-SA 2015).

1.6.2 Utveckling av applikationer

De tre ramverk som levde upp till dessa kriterier gjordes sedan ett projekt på. Jag tog fram en applikation som skulle kunna användas på iOS och Android. Att applikationen skulle kunna användas på iOS och Android är för att detta är de två plattformar som 2014 hade majoriteten av användare (Gartner 2014). Applikationen skulle utnyttja telefonens kamera, accelorometer och GPS. Att applikationen skulle utnyttja telefonens kamera, accelorometer och GPS är för att dessa tre funktioner är sådant som kan tillhandahållas av en hybridapplikation men inte av en webbapplikation (Xanthopoulos & Xinogalos 2013). Kriterierna som mättes på applikationerna är:

- komplexiteten i koden
- logiska rader kod som krävdes för att skriva applikationen

Applikationerna testkördes på en iPhone 6 som kör iOS 8.3 och en Motorola MotoG som kör KitKat 4.4.4.

1.6.3 Datainsamling

Datainsamling för undersökningen skedde i två omgångar. Den första omgången var vid urvalet av ramverk och då åtta stycken ramverk blev tre. Datan hämtades från det som Patel och Davidson (2011) benämner som “dokument”. Traditionellt har dokument använts som benämning för data som har nedtecknats eller tryckts. Tack vare den tekniska utvecklingen kan information lagras på så många andra olika sätt än nedtecknat eller tryckt och därför ingår nu även till exempel, det som jag kommer använda mig av, information från Internet (Patel & Davidson 2011). Då information från Internet är något som alltid ändras och uppdateras analyserades dokumenten under samma dag så att resultatet skulle bli så tillförlitligt som möjligt.

På dokumenten gjordes en innehållsanalys. Med hjälp av en innehållsanalys kan en kvantifiering av innehållet göras, vare sig det är på skrift, ljud eller bilder. I en innehållsanalys behövs det

utarbetas vilka enheter som ska mätas. Forskaren kan därefter räkna dessa utifrån hur många av enheten som förekommer (Denscombe 2009).

I denna undersökning analyserades dokument i form av webbsidor från Stackoverflow, Github samt respektive ramverks dokumentationswebbsida. På webbsidorna samlades data in i form av sökresultat, punkter som en dokumentation bör innehålla, antal issues och antal stjärnor på GitHub. När enheterna räknats eller samlats in ställdes de sedan upp i tabeller för att kunna jämföra de olika ramverkens enheter mot varandra.

Den andra omgången av datainsamling gjordes på applikationerna som utveckla utefter vilka tre ramverk som samlade på sig mest poäng under första datainsamlingen. Hur lätt det kommer att vara att underhålla och sätta sig in i koden undersöktes bara på de skrivna applikationerna och mättes med hjälp av cyklomatisk komplexitet och rader logisk kod.

Cyklomatisk komplexitet mäter hur många val som görs i logiken i skriven kod. Det används främst till två ändamål. Det första är att utifrån utvärderingen får ut hur många test som rekommenderas att skriva för koden. Det andra är att det kan användas för att göra utvärderingar kontinuerligt under hela utvecklingsprocessen vid utveckling av mjukvara för att säkerställa att koden alltid kommer att vara testbar och hanterbar (Watson & McCabe 1996).

Rader kod (LOC) kan användas som en indikator på underhållbarheten. Indikatorn baseras på antagandet att en applikation är enklare att underhålla när den har färre rader kod, t.ex. för att nya utvecklare kräver mindre träning och källkoden är enklare att läsa. Det finns andra mer sofistikerade utvärderingsmetoder men de är svåra att applicera på större, komplexa projekt där olika programmerings- och märkspråk används (Heitkötter, Hanschke & Majchrzak 2012).

Då jag skulle skriva all kod togs en kodstandard fram som visade hur funktioner och variabler deklareras vilket gör att LLOC kommer kunna att mätas på ett tillförlitligt sätt. Ett webbaserat verktyg[2] för att mäta CK och LLOC i JavaScript-kod användes men detta räknades även manuellt för att säkerställa att verktyget räknat korrekt.

1.6.4 Etiska reflektioner

Vetenskapligarådet har tagit fram principer som ska användas vid en konflikt mellan forskare och undersökningsdeltagare så att det kan göras en god avvägning mellan forskningskravet och individskyddskravet. Eftersom att problem, som konflikt, kan se olika ut från fall till fall har principerna skrivits så att de ska ses som vägvisare och inte detaljreglerade föreskrifter. Detta för att forskaren själv ska reflektera över principerna (Vetenskapsrådet 2015).

[2] http://jscomplexity.org/

Det finns även fyra konkreta huvudkrav som ska appliceras på svensk forskning. Dessa är:

- Informationskravet — Detta krav innebär att forskaren ska informera undersökningsdeltagarna om vad deras uppgift i projektet är och vilka villkor som finns. Viktigt här är att redogöra för vad för typer av obehag och risker som finns för undersökningsdeltagarna.
- Samtyckeskravet — Detta krav innebär att när undersökningar är av sådan karaktär att det kräver aktiv insats från undersökningsdeltagarna skall samtycke alltid inhämtas. Här tas även upp att undersökningsdeltagarna själva skall ha rätten att säga på vilka villkor de vill utföra undersökningen. Om undersökningsdeltagaren väljer att avbryta testet skall detta inte få några negativa följder för deltagaren.
- Konfidentialitetskravet — Detta krav innebär att undersökningsdeltagarna skall hållas anonyma och personuppgifter, om det finns, skall lagras så att utomstående, obehöriga personer inte kommer åt uppgifterna.
- Nyttjandekravet — Detta krav innebär att uppgifterna som samlats in från undersökningsdeltagarna endast får användas för forskning. Uppgifterna som samlats in får alltså inte användas till kommersiellt bruk. Uppgifter som samlats in får heller inte användas för att ta beslut om den enskilde undersökningsdeltagaren var det gäller till exempel vård eller tvångsintagning utan särskilt medgivande (Vetenskapsrådet 2015).

Då denna undersökning inte kommer att jobba mot några undersökningsdeltagare, utan bara handlar om dokumentanalys, och då är dessa krav inget jag kommer att behöva ta hänsyn till.

2. Teknisk bakgrund

Detta kapitel kommer att gå igenom vilka tekniker som används för att utveckla för webben. Kapitlet kommer även att gå igenom olika typer av applikationer och plattformar samt ge en introduktion till ramverken som valts ut för undersökningen.

2.1 Märk-, script- och programmeringsspråk

Detta avsnittet kommer vara en genomgång av de språk som används för att skriva en webb- eller hybridapplikation för att ge en introduktionen till tekniken som undersökningens applikationer kommer att utvecklas i.

2.1.1 HTML

HyperText Markup Language (HTML) är ett märkspråk använt för att skapa och visuellt presentera en webbsida. HTML stöder bilder och andra media. HTML är språket som beskriver hur strukturen på en webbsida är uppbyggd. Strukturen byggs upp med hjälp av taggar, HTML-element, som till exempel <img>, <h1>, <p> och (Mozilla Developer 2015a).

2.1.1.1 HTML5

HTML5 är designat att bli efterträdare till HTML4 som är den förra specifikationen. Istället för att enbart strukturera upp webbsidan, som HTML4 endast kunde göra, kan den göras till en robust webbapplikation.

HTML5 skapar intressanta nya möjligheter för mobila webbapplikationer med till exempel canvas-elementet, offline-lagring och medieuppspelning (Fling 2009). Canvas-elementet kan bland annat rendera grafer och användas vid spelutveckling (W3C 2015b).

2.1.2 CSS

CSS (Cascading Style Sheets) är ett deklarativt språk som kontrollerar hur en webbsida ser ut i webbläsaren. Webbläsaren lägger till CSS-deklarationerna till valda element och visar designen utvecklaren lagt till visuellt. En deklaration som beskrivs innan innehåller egenskaper med värden som sedan bestämmer hur webbsidan kommer att se ut (Mozilla Developer 2015b).

CSS är en av de tre kärnteknologierna, tillsammans med HTML och JavaScript. CSS används oftast för att sätta form på HTML-element men kan också användas med andra markup-språk som Scalable Vector Graphics (SVG) och Extensible Markup Language (XML) (Mozilla Developer 2015b).

SVG är ett märkspråk för att beskriva tvådimensionel grafik (W3C 2015a). XML är ett textbaserat format för att presentera strukturerad information: dokument, data, böcker,

transaktioner och liknande. XML används idag mest till att dela information mellan program och datorer både lokalt och över nätverk (W3C 2015c).

2.1.3 JavaScript

JavaScript är ett programmeringsspråk som mest är använt för att göra webbsidor dynamiska på klienten, alltså det som användaren ser (Mozilla Developer 2015c). Idag har JavaScript kommit tillbaka till serversidan med Node.js-plattformen (Node.js 2015).

JavaScript används mest i webbläsaren och gör det möjligt för utvecklare att exempelvis manipulera webbsidans innehåll genom Document Object Model (DOM), manipulara data med Asynchronous JavaScript + XML (AJAX), måla grafik med canvas med mera. JavaScript är ett av världens mest använda programmeringsspråk (Mozilla Developer 2015c).

DOM är ett plattform- och språksneutralt interface som tillåter program och skript att dynamisk få åtkomst att uppdatera innehållet, få åtkomst till strukturen på en webbsida och även att dynamiskt ändra hur webbsidan ser ut (W3C 2015d).

AJAX gör att webbsidor kan göra inkrementella uppdateringar till användargränssnittet utan att behöva ladda om hela sidan (Mozilla Developer 2015d).

2.1.4 PHP

PHP (Hypertext Preprocessor) är ett skriptspråk som speciellt är anpassat för webbutveckling och kan bäddas in i HTML.
Det som skiljer PHP från ett klientskriptspråk som JavaScript är att koden exekveras på servern och genererar HTML som sedan skickas till klienten. Klienten tar emot resultatet från PHP-skriptet utan att veta hur själva PHP-skripten såg ut innan det exekverades på servern (PHP.net 2015).

2.1.5 MySQL

MySQL är ett databashanteringssystem. En databas i sig är en strukturerad samling av data. MySQLs databaser är relationsdatabaser, vilket innebär att istället för att spara alla data tillsammans så delas datan upp i mindre delar där delarna har olika relationer till varandra. MySQL är open source vilket gör det möjligt för vem som helst att både modifiera och använda mjukvaran (MySQL 2015).

2.1.6 SQL

SQL-delen i MySQL är en förkortning för “Structured Query Language” SQL är det vanligaste språket som används för att få tillgång datan i databaser (MySQL 2015).

2.2 Olika typer av applikationer

Detta avsnitt avser att ge en redogörelse för vilka typer av applikationer som finns för att ge en bättre bild av vad de är samt ge några för- och nackdelar med de olika applikationerna.

2.2.1 Webbapplikation

En webbapplikation behöver inte installeras på användarens smartphone utan den körs direkt i webbläsaren. Med hjälp av HTML, CSS och JavaScript kan webbapplikationen utvecklas så att den känns som att vara som en "riktig" (läs: native-) applikation även fast den körs i webbläsaren.

När den första iPhonen kom var då det verkligen började hända saker på webbapplikations-fronten. Med hjälp av WebKit, en webbläsarmotor (The WebKit Open Source Project 2015), kunde det utvecklas mer komplexa applikationer som gav användaren en rikare upplevelse (Fling, 2009).

Fördelar med webbapplikationer är att de enligt Fling (2009) är enkla att utveckla om utvecklaren har kunskap i HTML, CSS och JavaScript och att det enda som användaren behöver för att komma åt applikationen är en webbläsare. En nackdel är att en webbapplikation saknar funktionalitet som en native-applikation gör, som GPS, kamera och accelerometer (Fling 2009).

2.2.2 Hybridapplikation

En hybridapplikation försöker att kombinera fördelarna från webben med native-applikationer. Hybridapplikationer är ofta byggda i HTML, CSS och JavaScript. Som en webbapplikation så exekveras källkoden av en webbläsare men för hybridapplikationen sparas källkoden i applikationen medan webbapplikationen behöver ladda ner källkoden från servern den ligger på varje gång.

En fördel med en hybridapplikation mot en webbapplikation är att hybridapplikationen kan bli installerad på smartphonen. En annan fördel är att hybridapplikationen har åtkomst till hårdvaran och kan därför komma åt funktionalitet som inte en webbapplikationen kan (Xanthopoulos & Xinogalos 2013).

En nackdel med hybridapplikationer enligt Xanthopoulos och Xinogalos (2013) är att det är skillnad i prestandan hos slutanvändaren av hybridapplikationen då dessa är webbaserade.

2.2.3 Native-applikation

Native-applikationerna skulle även kunna att kallas för plattformsspecifika applikationer eftersom att de måste utvecklas och kompileras för varje plattform man vill stödja. Dessa applikationer är byggda specifikt för enheter för plattformen i fråga.

Att applikationerna är byggda på plattformen de vill stödja innebär att den kommer åt hårdvaran och funktionaliteten som enheter tillhandahåller (Xanthopoulos & Xinogalos 2013)

Fördelar med native-applikationer är att de ger användaren den bästa användarupplevelsen med rik design och tillgång till enhetens funktionalitet. Nackdelar är att det är svårt att skriva om en native-applikation till andra plattformar. Det är dyrt att utveckla, testa och stödja flera plattformar (Fling, 2009).

2.3 Plattformar

Det finns idag många olika plattformar som bör utvecklas för när man skall ta fram en applikation. För att understryka hur olika plattformarna är kommer här nedan att göras en redogörelse för hur det ser ut för iOS och Android vid utveckling. Detta är intressant då dessa var de plattformar med flest sålda enheter under 2013. Android stod för 78.4% av alla sålda enheter och iOS stod för 15.6% (Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014 2014).

2.3.1 iOS

För att kunna utveckla en applikation för iOS behövs till att börja med en Apple-dator som kör OS X, Apples operativsystem. Applikationerna skrivs med hjälp av Xcode, en IDE som används för att skriva kod, debugga och göra layout. Språket som används för att skriva applikationerna är främst Objective-C men idag även i Apples egna programmeringsspråk Swift (Apple 2015).

Den mesta av funktionaliteten som du kan utveckla i Xcode kan också testas i Xcode för att det finns en simulator som kör applikationen. Dock går inte all funktionalitet från hårdvaran att testa i programmet utan måste göras på en fysisk enhet. Det som inte går att testas är bland annat GPS, kamera och accelerometer men dessa funktioner går att simulera i programmet. Detta innebär att för att kunna utveckla en applikation som kräver den funktionaliteten bör en riktig enhet finnas på utvecklingsplatsen (Goadrich & Rogers, 2011). Idag kostar den billigaste varianten av iPhone 6 6395 SEK och den billigaste varianten av iPhone 6 Plus 7395 SEK (Apple Store 2015)**.** Detta anses av Goadrich och Rogers (2011) vara mycket om smartphonen bara behövs som testenhet.

2.3.2 Android

Till skillnad mot iOS så kan Android utvecklas i vilket annat större operativsystem. Till exempel, Mac OS X, Windows (XP eller högre) eller Linux. Detta gör att det är mer tillgängligt än att utveckla för iOS då det kan göras från olika plattformar. Funktionaliteten för en Android-applikation skrivs i Java och layouten i XML (Goadrich & Rogers 2011).

Vid utveckling av applikationer till Android kan Android Emulator användas för att emulera applikationen i utvecklingsmiljön utan att behöva en extern testenhet. Med hjälp av Android

Emulator kan till exempel GPS, SMS/samtal, accelerometer och kompass simuleras under utveckling (Goadrich & Rogers 2011).

2.4 Ramverk

Även fast några av ramverken jag kommer att använda kallar sig själv för plattformar så kommer de att gå under termen ramverk i denna rapport. Detta för att enligt Fling (2009) så körs ett ramverk ovanpå ett operativsystem och delar tjänster med detta. Dessa skulle till exempel kunna vara säkerthet-, autentisering-, och grafiktjänster.

En plattforms främsta uppgift är att ge tillgång till enheten, i detta fall en smartphone. För att kunna köra mjukvara och andra tjänster på enheter behövs det en plattform, eller ett lågnivåspråk (“core programming language”) som all mjukvara är skriven i (Fling 2009). Exempel på en plattform är iOS eller Android. Det vi kommer att använda oss av för att skriva hybridapplikationerna är ramverk.

Härefter kommer var och ett av de åtta ramverken som kommer att vara med i studien få en kortare presentation.

2.4.1 Meteor

Meteor är ett ramverk för att bygga webb- och smartphoneapplikationer (Meteor 2015). Enligt Meteorpedia (2015) kan man bygga hyrbridapplikationer för webb, iOS och Android.

Meteor är ett klient-server-databas-ramverk helt skrivet i JavaScript. Meteors server körs på NodeJS och är ett av de få ramverk som både kan köras front- och back-end i. Databasen som Meteor använder sig av som standard är MongoDB. (Meteorpedia 2015).

2.4.2 AppGyver

Enligt AppGyver (2015) kan man skriva en applikation i vanlig HTML5 och CSS som ser ut och känns precis som ett native interface och också ha tillgång till, som nämnt innan, funktionalitet från smartphonen som en vanliga webbapplikation inte har åtkomst till.

2.4.3 Ionic

Ionic (2015) säger sig själva vara ett ramverk för att skapa hybridapplikationer med webbteknologier. Ionic tillhandahåller ett bibliotek med smartphone-optimerade HTML-, CSS- och JavaScript-komponenter som verktyg för att bygga interaktiva applikationer. Ionic är byggt för att prestera på de senaste mobila enheterna. (Ionic 2015).

2.4.4 MoSync
MoSync är ett ramverk för att snabbt kunna utveckla hybridapplikationer med hjälp av HTML5, CSS och JavaScript. (MoSync 2015)

2.4.5 Apache Cordova
Apache Cordova är en samling av API:er som tillhandahåller funktionalitet för utvecklare att komma åt en smartphones egenskaper som kamera och accelerometer med JavaScript. När man använder Cordovas APIer kan en applikation skrivas utan att utvecklare kan någon som helst plattformsspecifik kod, som Java eller Objective-C (Apache Cordova 2015).

Eftersom dessa JavaScript-APIer är konsekventa över olika plattformar och byggda på webbstandader kan applikationerna skrivas om till andra plattformar med några få eller inga ändringar alls (Apache Cordova 2015).

Som nämns i metoddelen är ett kriterium att ramverken ska ta hjälp av Cordovas API för att komma åt funktionaliteten i smartphonen men då det även går att skriva applikationer direkt med hjälp av dessa API:er är det också med som ett ramverk.

2.4.6 Phonegap
Med hjälp av Phonegap kan man skriva hyrbridapplikationer med HTML5, CSS och JavaScript. Phonegap utvecklas av företaget Nitobi som köptes upp av Adobe 2011 (Phonegap 2015).

2.4.7 Titanium
Titanium är ett ramverk för att skriva hybridapplikationer i HTML, CSS och JS. Detta ramverk tillhandahåller en egen IDE för utveckling av applikationer (Titanium 2015)

2.4.8 Enyo
Enyo-applikationer skrivs genom att man gör små komponenter skrivna i HTML, CSS och JS och dessa små komponenter tillsammans med andra komponenter blir andra större komponenter. Detta gör det enkelt att återanvända och underhålla applikationen (Enyo 2015).

3. Litteraturöversikt

Detta kapitel kommer att behandla böcker, artiklar och dokument som är relevanta för syfte samt problemområde för rapporten.

3.1 Tidigare forskning

Heitkötter et al. (2012): *Comparing cross-platform development approaches for mobile applications* — Denna artikel tar fram olika kriterier för hur man kan gå till väga vid utvärdering av olika ramverk för utveckling av hybridapplikationer.

Artiklen ger en heltäckande översikt av aktuella tillvägagångssätt vid utveckling av plattformsoberoende applikationer. Utifrån detta kommer Heitkötter, Hanschke & Majchrazk fram till ett ramverk med kritierer som kan användas vid utvärdering. Ramverket innehåller sju kriterier som rör utvecklarens perspektiv. Dessa är:

- *Utvecklingsmiljö* — Här utvärderas funktionaliteten av utvecklingsmiljön, speciellt då om ramverket har en egen IDE och om det finns automatiska tester för att testa koden. Här utvärderas även hur enkelt det är att sätta upp en fullt fungerande utvecklingsmiljö på den önskade plattformen.
- *Användargränssnitt* — Detta kriterium täcker processen av att skapa ett grafisk användargränssnitt. Att kunna testa och se användargränssnittet utan att behöva köra det på en extern enhet anses vara fördelaktigt.
- *Hur enkel utveckling är* — Detta kriterium summerar kvaliteten på dokumentation samt hur hur svårt ramverket är att lära sig, det vill säga inlärningskurvan. Det första mäts genom att det ska finnas tillgängliga kodexempel, länkar till liknande problem och användarkommentarer. Inlärningskurvan beskriver, subjektivt, framåtskridandet för en utvecklare under första undersökningen av ett ramverk. Här undersöks om ramverket använder sig av redan kända paradigmer för utvecklare. Detta är viktigt för att se hur det kan gå när medarbetare ska introduceras för ramverket och hur mycket ramverkspecifik kod utvecklaren behöver lära sig.
- *Underhållbarhet* — Rader kod kan användas som en indikator för att utvärdera underhållbarheten. Att man använder denna typ av kriterium är för att ju färre rader kod det finns desto enklare är det för nya utvecklare att sätta sig in i koden för att den är kort och lättare att läsa.
- *Skalbarhet* — Skalbarheten baseras på hur bra större utvecklingsgrupp arbete tillsammans i samma ramverk. Detta mäts genom att kolla på om projektet kan delas upp i flera moduler så att fler utvecklare kan arbeta på samma projekt samtidigt.
- *Möjligheter till vidareutveckling* — Bestämmer återanvändbarheten av koden över olika metoder, detta för att minska risken av låsa sig till en viss metod under utveckling.

- *Hastighet och kostnad för utveckling* — Utvärderar hastigheten av utvecklingsprocessen och faktorer som hindrar snabb och entydig utveckling. Kostnaden estimeras inte enskilt utan räknas utifrån hastigheten av utvecklingen.

Xanthopoulos et al. (2013): *A comparative analysis of cross-platform development approaches for mobile applications* — Denna artikel fokuserar på trender för utveckling av plattformsoberoende applikationer. Analysen sker på tre olika punkter. Det första Xanthopoulos och Xinogalos ser över är vilka typer av tekniker det finns för att utveckla plattformsoberoende applikationer. Det andra de ser över är för- och nackdelar för med de olika teknikerna och till sist identifierar de en lovande plattformsoberoende teknik och undersöker hur effektivt det är i praktiken.

Kriterierna Xanthopoulos och Xinogalos använde när de utvärderade vilken typ av teknik de ska testa var:

- *Distribution* — Utvärderar om det är enkelt och hur applikationer kan göras tillgängliga för användare genom t.ex. Google Play eller Apples AppStore.
- *Utbredda teknologier* — Utvärderar om applikationen kan skapas med utbredda teknologier, som JavaScript.
- *Hårdvaruåtkomst* — utvärderar om applikationen har åtkomst, begränsad eller full, till underliggande hårdvara.
- *Hur användaren uppfattar prestandan* — Utvärderar hur applikationen presterar utefter vad slutanvändaren tycker om laddningstid och exekveringshastighet jämfört mot en native-applikation.
- *"Look and feel"* — Utvärderar om applikationen kan ärva användargränssnittet från plattformar som iOS och Android eller om användargränssnittet behöver simuleras med hjälp av ramverk som jQuery Mobile.

3.2 Att utvärdera mjukvara

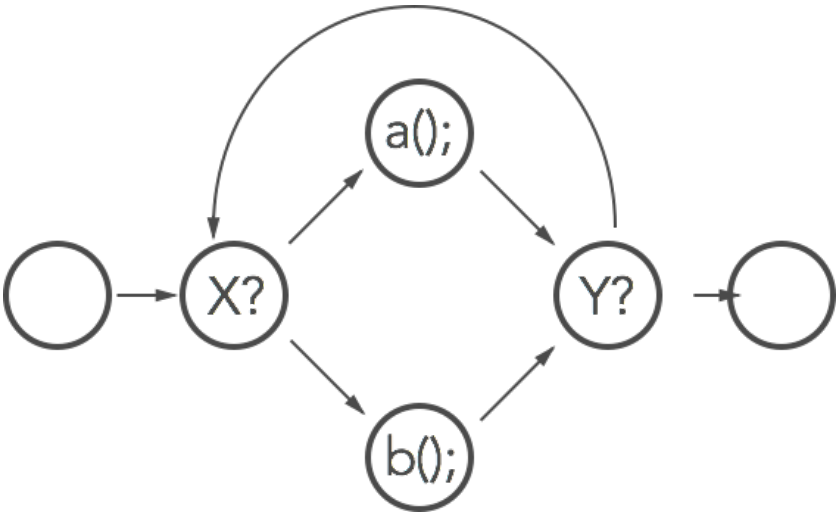
3.2.1 Cyklomatisk komplexitet (eng. cyclomatic complexity)

Cyklomatisk komplexitet mäter hur många val som görs i logiken i skriven kod. Det används främst till två ändamål. Det första är att utifrån utvärderingen få ut hur många test som rekommenderas att skriva för koden. Det andra är att det kan användas för att göra utvärderingar kontinuerligt under hela utvecklingsprocess vid utveckling av mjukvara för att säkerställa att koden alltid kommer att vara testbar och hanterbar (Watson & McCabe 1996).

Cyklomatisk komplexitet utgår från ett programs kontrollflödesgraf och mäter hur många val som görs i logiken i den skrivna koden. Kontrollflödesgrafen är en riktad graf som består av noder och kanter. Noderna är sammanhängande stycken av kod och kanterna hur många exekveringsvägar det finns mellan noderna (Watson & McCabe 1996). Exempelvis kan programkoden i figur 1 översättas till grafen i figur 2.

```
10
11  do {
12      if (x) {
13          A();
14      }
15      else {
16          B();
17      }
18  } while (y);
19
20
```

Figur 1 — kodexempel för cyklomatisk komplexitet



Figur 2 — kontrollflödesgraf från koden i figur 1

Cyklomatisk komplexitet V i en graf G med n noder, de olika sammanhängade styckena kod, och e kanter, exekveringsvägarna mellan noderna, skrivs ut som:

$$V(G) = e - n + 2$$

Koden i figur 1 har komplexiteten 3. e = 7, kanterna ses som pilarna i grafen. n = 6, noderna är de cirklarna i grafen. 7 - 6 + 2 = 3. Enligt Watson & McCabe (1996) är en lämplig övre gräns på komplexiteten 10.

3.2.2 Utvärdera dokumentation

Strukturen hos dokumentation för mjukvara inkluderar hur den är organiserad i olika segment och i vilken ordning dessa segment kommer. Strukturen hos användardokumentationen borde hjälpa användaren i uppgiften att hitta och förstå innehållet.

Listan nedan listar vilka punkter IEEE Computer Society (2001) som bör vara med i dokumentation:

1. *Identifikationsdata* — det första som skall visas när man ser dokumentationen skall vara vilken organisation som skrivit samt vilken version det är på dokumentationen.
2. *Innehållsförteckning* — innehållsförteckning innebär att det någonstans skall listas alla rubriker med antingen en referens till ett sidnummer eller en elektronisk länk.
3. *Lista över illustrationer* — lista med tabeller, illustrationer och figurer.
4. *Introduktion* — introduktionen skall innehålla en kort översikt av programvarans ändamål.
5. *Information om hur man använder dokumentationen* — något i dokumentationen som rekommenderar de olika delarna av målgruppen vilken sektion de ska besöka.
6. *Concept of operations (CONOPS)* — att dokumentationen skall beskriva hur systemet funkar ur ett användarperspektiv.
7. *Procedurer* — instruktioner för installation och komma igång.
8. *Information om programvarukommandon* — Dokumentation skall förklara parametrar som krävs, parametrar som är valdfria, standardalternativ och syntax. Exempel skall finnas för att illustrera hur man använder kommandon.
9. *Felmeddelanden och problemlösning* — dokumentationen skall innehålla information om vanliga problem och hur användaren själv kan lösa dessa.
10. *Ordlista* — dokumentationen ska innehålla en lista med alla vanligt förekommande ord i bokstavsordning.
11. *Relaterade informationskällor* — dokumentation kan innehålla relaterade källor, som relaterade webbsidor eller relaterad dokumentation.
12. *Navigationsfunktioner* — innebär att man ska kunna ta sig till nästa relaterade ämne.
13. *Index* — en lista med ord, figurer och illustrationer som är länkade till vart de är i texten.
14. *Sökmöjlighet* — om dokumentationen är elektronisk skall det finnas möjlighet att kunna söka efter ord eller fulltext.

De som är valbara är “lista över illustrationer”, “relaterade informationskällor” och “index” och de kommer inte att användas i undersökningen. De används inte för att de just är valbara och inte är något som behöver finnas i dokumentation.

3.2.3 LOC - Lines of Code

Rader kod (LOC) kan användas som en indikator på underhållbarheten. Indikatorn baseras på antagandet att en applikation är enklare att underhålla när den har färre rader kod, t.ex. för att nya utvecklare kräver mindre träning och källkoden är enklare att läsa. Det finns andra mer sofistikerade utvärderingsmetoder men de är svåra att applicera på större, komplexa projekt där olika programmerings- och märkspråk används (Heitkötter et al. 2012).

En kodstandard kommer att tas fram så att funktioner och variabler deklareras på samma sätt vilket gör att rader kod kommer att kunna mätas på ett tillförlitligt sätt.

4. Undersökningens upplägg och genomförande

Detta kapitel kommer att gå igenom hur undersökningens upplägg och genomförande såg ut.

4.1 Datainsamling

Datainsamling för undersökningen har skett i två omgångar. Den första omgången gjordes vid urvalet av ramverk och åtta stycken ramverk blev tre. Datan hämtades från det som Patel och Davidson (2011) benämner som "dokument". Traditionellt har dokument använts som benämning för data som har nedtecknats eller tryckts. Tack vare den tekniska utvecklingen kan information lagras på så många andra olika sätt än nedtecknat eller tryckt och därför ingår nu även till exempel, det som jag kommer använda mig av, information från Internet (Patel & Davidson 2011).

Den andra omgången av datainsamling skedde i samband med och efter utveckling av de tre applikationerna som testade de tre utvalda ramverken.

All data som kommer att samlas in från webbsidor som används i undersökningen kommer att vara tagna från samma datum för att numrena kan ändras från ena dagen till den andra.

4.2 Första urvalet

Utvärdering av de åtta ramverken skedde utefter följande kriterier:

- hur många stjärnor projektet har på GitHub
- hur många resultat som kom upp om man sökte på ramverkets namn på stackoverflow
- hur aktivt underhållet ramverket (totala issues mot hur många lösta issues från GitHub)
- hur bra dokumentationen uppfyllde IEEE Computer Society (2001) punkter på vad en dokumentation bör innehålla

Jag har gjort en analys, i procent, av hur ramverken förhåller sig till varandra på kritierierna där det endast är siffror som avgör förhållandet. Ett exempel på hur detta såg ut kan ses i tabell 1.

Tabell 1 — Exempel på utvärderingstabell

Ramverk	**GitHub-stjärnor**	**Procent**	**Poäng**
Ramverk 1	10000	66.666%	3
Ramverk 2	500	3.333%	1
Ramverk 3	4500	30%	2
Total	15000		

Som ovan i tabell 1 blev det enkelt att läsa ut vilket ramverk som är störst i förhållande till de andra. Exempel 1 har 66.6% av de sammanlagda stjärnorna vilket gör det till det största. Procenten gjordes sedan om till poäng för att kunna läggas ihop för att få en total för varje ramverk, vilket gjorde det lättare att ställa dem mot varandra när allt är uträknat. Ramverket med mest procent tilldelades åtta poäng och ramverket med minst procent tilldelades 1 poäng. Detta gjordes för att mätvärdena ska ge en rangordning men utan att säga något om avståndet mellan värdena. (Patel & Davidson 2011). Om flera ramverk fick samma resultat under en mätning fick ramverken samma poäng. Exempelvis om två ramverk presterade bäst tilldelades dessa åtta poäng vadera och ramverket med näst mest poäng tilldelades sju poäng. Poängen ska inte ses som en rangordning utom som poäng.

Dokumentation utvärderades efter IEEE Computer Society (2001) för hur mjukvara ska dokumenteras. Den ursprungliga listan innehåller 14 punkter men denna utvärdering kommer endast att använda sig av de punkter som är skrivna som nödvändiga enligt standarden.

Punkterna som ramverken fick poäng för om de innehöll det är:
- *Identifikationsdata*
- *Innehållsförteckning*
- *Introduktion*
- *Information om hur man använder dokumentationen*
- *Concept of operations (CONOPS)*
- *Procedurer*
- *Information om programvarukommandon*
- *Felmeddelanden och problemlösning*
- *Ordlista*
- *Navigationsfunktioner*
- *Sökmöjlighet*

4.3 Utveckla applikationer

Jag har utvecklat tre stycken applikationer som innehåller exakt samma funktionalitet, detta för att kunna göra en rättvis jämförelse som möjligt. Applikationerna utvecklades för att kunna mäta komplexiteten i koden samt logiska rader kod som är två av kriterierna som presenterades i 1.2.

Applikationen skulle kunna köras på iOS och Android samt kunna komma åt smartphonens position via geolocation, kunna ta ett kort med smartphonens kamera och använda accelerometer.

4.3.1 Applikationens funktionalitet

I applikationen fanns funktionalitet för att använda smartphonens kamera och geolocation där funktionaliteten anropades med ett knapptryck och en dialogruta kom upp som visade om allt gått bra eller om något problem uppstått. Accelerometerns funktionalitet startades så fort applikationen startades och gav en dialogruta med återkoppling när smartphonen skakades. Det fanns felhantering för funktionaliteten i koden för applikationen som gav en dialogruta med återkoppling ifall något gick fel.

4.3.2 Kodstandard

Kodstandard togs fram för att resultatet från LOC skulle bli trovärdigt. Om kodstandard inte sätts upp och följs skulle kod som har samma funktionalitet men annorlunda formattering ta upp fler rader. Raderna som räknades är logisk LOC (LLOC). LLOC innebär att man istället för att räkna varje rad räknar varje körbart kommando. I figur 3 nedan är det en LOC men två logiska LOC för att det är två körbara kommandon. För jämförelse se figur 4 där samma kod tar upp fysiska rader men fortfarande har samma LLOC.

```
52
53
54
55  for (i = 0; i < 100; i++) alert("hello");
56
57
58
59
```

Figur 3 — 1 LOC men 2 LLOC

```
60
61
62
63  for (i = 0; i < 100; i++)
64  {
65      alert("hello");
66  }
67
68
69
```

Figur 4 — 4 LOC men 2 LLOC

Funktionsdeklareringar hade klammerparentes på samma rad som deklarationen startade som kan ses i figur 5.

```
144
145
146
147
148  function funktionsNamn() {
149      // kod
150  }
151
152
153
154
```

Figur 5 — funktionsdeklarering

Vid if-sats med en else/elseif så startar else-satsen på raden efter if-satsens klammerparantes som kan ses i figur 6.

```
145
146
147
148  if(this.isTrue()) {
149    // kod
150  }
151  else {
152    // annan kod
153  }
154
155
156
157
```

Figur 6 — if- och else-satser

Variabler deklareras på en rad som kan ses i figur 7.

```
156
157
158
159
160  var variabelNamn = {};
161
162
163
164
```

Figur 7 — variabeldeklarering

4.4 Utvärdering av framtagna applikationer

Den slutgiltiga utvärderingen innefattade också den data som samlades in under urvalsprocessen. Här utvärderades ramverken även efter:

- logiska rader kod som krävdes för att skriva applikationen
- komplexiteten i koden

Här analyserades utfallet utifrån samma princip som vid första datainsamlingen som kommer ge de olika ramverken poäng från 1-3 där 3 är bäst.

Hur logiska rader och komplexiteten i koden mäts och vad den kan påvisa har tidigare beskrivs i 1.6.3 Datainsamling.

De tre ramverk som blev utvalda fick sina poäng omgjorda till 1, 2 eller 3 beroende på hur många poäng de samlade på sig under den första gallringen. Detta för att fortsätta på principen att mätvärdena ska ge en rangordning men utan att säga något om avståndet mellan värdena. (Patel & Davidson 2011).

5. Resultat

Detta kapitel kommer att redovisa resultatet av undersökningen med kortare förklaringar och i tabellform.

5.1 GitHub-stjärnor

Nedan i tabell 2 presenteras resultatet från hur många GitHub-stjärnor som ramverken hade. Det var framförallt två ramverk som stod ut på denna punkten: Meteor och Ionic, som hade 46.25% respektive 28.88% av de totala 50 190 stjärnorna. Nästefter dessa två kom Cordova och Phonegap med 8.81% respektive 7.93%. Enyo hade 3.66%, Titanium hade 3.56%, AppGyver 0.05% och sista MoSync med 0.04%.

Tabell 2 — resultat från GitHub-stjärnor

Ramverk	**Github-stjärnor**	**Procent**	**Poäng**
AppGyver	270	0.05%	2
Cordova	4,422	8.81%	6
Enyo	1,838	3.66%	4
Ionic	14,456	28.88%	7
Meteor	23,208	46.25%	8
MoSync	225	0.04%	1
Phonegap	3,983	7.93%	5
Titanium	1,788	3.56%	3
Totalt	50190 stjärnor		

5.2 Stackoverflow-resultat

Nedan i tabell 3 presenteras resultatet från hur många träffar som fanns på Stackoverflow efter att man sökt på ramverkets namn. Ramverket som presterade bäst här var Phonegap med 48.93% av träffarna följt av Meteor på 28.36%. Titanium fick 14.53%, Ionic 4.65%, Cordova 2.43%, Enyo 0.06%, MoSync 0.03% och AppGyver 0.02%.

Tabell 3 — resultat från träffar på Stackoverflow

Ramverk	**Stackoverflow-resultat**	**Procent**	**Poäng**
AppGyver	173	0.02%	1
Cordova	1,964	2.34%	4
Enyo	513	0.06%	3
Ionic	3,899	4.65%	5
Meteor	23,808	28.36%	7
MoSync	297	0.03%	2
Phonegap	41,066	48.93%	8
Titanium	12,201	14.53%	6
Total	83921 träffar		

5.3 GitHub-issues

Nedan i tabell 4 presenteras resultatet från hur många procent av alla issues är lösta mot totala issues. Titanium var ramverket som presterade bäst på denna punkten och hade 85.16% lösta issues följt av AppGyver med 47.67%. Sedan kom Phonegap med 34.84%, Cordova med 25.78%, Meteor med 16%, Ionic med 8.81% och sist MoSync och Enyo som inte lösta eller totala issues.

Tabell 4 — resultat lösta mot totala issues på GitHub

Ramverk	**Lösta issues**	**Öppna issues**	**Total**	**Procent**	**Poäng**
AppGyver	226	474	700	32.38%	7
Cordova	1557	6039	7596	20.49%	5
Enyo	0	0	0	—	2
Ionic	229	2599	2828	8.09%	3
Meteor	409	2555	2964	13.79%	4
MoSync	0	0	0	—	2
Phonegap	69	198	267	25.84%	6
Titanium	3565	4186	7751	45.99%	8

5.4 Dokumentation

Nedan i tabell 5 presenteras resultatet från hur många punkter ramverkens dokumentation samlade på sig av IEEE Computer Society (2001). Resultatet visar hur många punkter ramverken innehöll av de 11 obligatoriska punkterna. Här var det 6 ramverk som samlade på sig lika många poäng: Meteor, MoSync, Cordova, Phonegap, Titanium och Enyo och får därför 8 poäng vardera. Efter dessa kom Ionic och sist AppGyver. Denna data finns i bilaga 3.

Tabell 5 —resultat för dokumentation

Ramverk	**Dokumentation**	**Procent**	**Poäng**
AppGyver (AppGyver Documentation 2015)	2	4%	6
Cordova (Apache Cordova Documentation 2015)	7	14%	8
Enyo (Enyo Documentation 2015)	7	14%	8
Ionic (Ionic Documentation 2015)	6	12%	7
Meteor (Meteor Documentation 2015)	7	14%	8
MoSync (MoSync Documentation 2015)	7	14%	8
Phonegap (PhoneGap Documentation 2015)	7	14%	8
Titanium (Titanium Documentation 2015)	7	14%	8
Total	50		

5.5 Total från dokumentanalysen

Nedan i tabell 6 presenteras de sammanlagda poängen från tabell 2, tabell 3, tabell 4 och tabell 5. De som samlade ihop mest poäng var Meteor, Phonegap och Titanium men precis efter att dokumentanalysen var klar valde Titanium att gå och bli kommersiellt. Det vill säga att de valde att inte längre vara open source och gratis, som var ett av kriterierna hos ramverken.

Tanken var först att bygga en av applikationerna i Titanium för att undersökningen var så långt gången men efter att jag inte fick tag i plattformen som krävdes, och tillhandahölls av Titanium, i tid fick ramverket strykas från listan och den fjärde, Cordova, istället ta plats bland de tre som ska utvecklas applikation på. Då Titanium blev kommersiellt under tiden av undersökningen och dokumentanalysen skedde innan detta kommer dess resultat ändå få vara kvar i tabell 2-5. De tre ramverk som byggdes applikationer på är de som är fetstilta.

Tabell 6 —totalen från dokumentanalysen i första urvalet

Ramverk	**GitHub-stjärnor**	**Stackoverflow**	**Issues**	**Dokumentation**	**Total**
AppGyver	2	1	7	6	16
Cordova	**6**	**4**	**5**	**8**	**23**
Enyo	4	3	2	8	17
Ionic	7	5	3	7	22
Meteor	**8**	**7**	**4**	**8**	**27**
MoSync	1	2	2	8	13
Phonegap	**5**	**8**	**6**	**8**	**27**
~~Titanium~~	~~3~~	~~6~~	~~8~~	~~8~~	~~25~~

5.6 Resultat från applikationsutvärderingen

Tabell 7 visar resultatet av hur många logiska rader kod det krävdes för att skriva applikationerna. Meteor krävde minst logiska rader kod, 30 LLOC, och fick därför högst poäng. Cordova och Phonegap krävde 33 LLOC.

Tabell 7 — resultat från LLOC

Ramverk	**LLOC**	**Poäng**
Cordova	33	2
Meteor	30	3
Phonegap	33	2

Tabell 8 visar resultatet från den cyklomatiska komplexiteten. Meteors kod fick 6 i CK och Cordova och Phonegap fick 1. Cordova och Phonegaps kod var mindre komplex och fick därför 3 poäng och Meteor 2.

Tabell 8 — resultat från CK

Ramverk	**CK**	**Poäng**
Cordova	1	3
Meteor	6	2
Phonegap	1	3

5.7 Sammanfattning av resultatet

Tabell 9 visar sammanfattningen av resultatet från undersökningen. Totalt så slutade Meteor och Phonegap på 8 poäng och Cordova fick 7 poäng.

Tabell 9 —totalen av undersökningen

Ramverk	**Poäng från dokumentanalys**	**CK**	**LOC**	**Total poäng**
Cordova	2	3	2	7
Meteor	3	2	3	8
Phonegap	3	3	2	8

6. Analys och diskussion

Detta kapitel avser att analysera och diskutera resultatet av undersökningen.

Syftet med undersökningen var att samla kunskap om olika ramverk för att ta fram hybridapplikationer för att se hur dessa presterade mot kriterier som tagits fram tillsammans med Bulldozer. Detta skulle resultera i kunskap som skall vägleda dem, och andra webbutvecklare, till vilket ramverk som är "bäst" när målet är att börja utveckla applikationer men inte har kompetensen att göra plattformsspecifika applikationer.

Med hjälp av kriterierna var målet att komma fram till hur väldokumenterat och underhållet ramverket var, hur mycket hjälp det fanns att hitta på Stackoverflow, hur lätt det var att sätta sig in i koden och hur lätt det kommer vara att underhålla koden. Resultatet från detta var det som kommer motivera vilket ramverk som är "bäst".

IEEE Computer Society (2001) stod som grund för vad en dokumentation bör innehålla för att vara fullständig. Punkterna som togs upp användes för att ställa ramverkens dokumentationer emot varandra. Hur underhållet ramverket var undersöktes genom att se hur många totala issues mot stängda issues ramverket hade på GitHub. Detta användes för att se om personerna som ligger bakom ramverket aktivt underhåller och åtgärdar problem som rapporteras. Hur mycket hjälp det fanns att hitta på Internet mättes genom att undersöka hur mycket information som fanns att hämta på Stackoverflow. Stackoverflow är en plattform för att fråga frågor och få svar inom programmering och ligger idag på på plats 60 av världens mest besökta webbsidor. Ramverket popularitet mättes genom att analysera hur många stjärnor de hade på GitHub.

Hur lätt det kommer att vara att underhålla och sätta sig in i koden undersöktes bara på de tre ramverk som tog sig vidare från dokumentanalysen och mättes med hjälp av cyklomatisk komplexitet och logiska rader kod.

6.1 Meteor

6.1.1 Första urgallringen

Meteor var det ramverk som samlade på sig mest poäng under första urgallringen, som kan ses i tabell 6. Detta skedde främst för att Meteor är populärt på GitHub, har mycket information att hämta från Stackoverflow samt har en dokumenatation som fick högsta poäng utifrån IEEE Computer Society (2001) mall för vad som en dokumentation bör innehålla.

Enligt IEEE Computer Society (2001) bör en dokumentation innehålla 11 punkter:

- *Identifikationsdata*
- *Innehållsförteckning*
- *Introduktion*

- *Information om hur man använder dokumentationen*
- *Concept of operations (CONOPS)*
- *Procedurer*
- *Information om programvarukommandon*
- *Felmeddelanden och problemlösning*
- *Ordlista*
- *Navigationsfunktioner*
- *Sökmöjlighet*

Av dessa innehöll Meteors dokumentation sju punkter. Att Meteor hade 46.2% av stjärnorna av alla ramverk tyder på att många utvecklare som håller till på GitHub tycker om ramverket och vill följa dess utveckling. 46.2% gjorde att Meteor fick högsta poängen, åtta. Att ramverket är såpass populärt på GitHub kan bidra till att det finns mycket information att hämta från Stackoverflow, där ramverket fick 7 poäng.

På punkten hur väl underhållet ramverket är fick Meteor fyra poäng och kom efter båda Cordova, med fem poäng, och Phonegap, med sex poäng. Att Meteor tappade så mycket på denna punkten när ramverket tidigare fått antingen åtta eller sju poäng kan bero på att Meteor inte är lika moget som de andra två ramverken. Meteor är idag, 20/5-15, på version 1.1.0.2 medan Phonegap är på version 2.9.1 och Cordova på 3.9.0. Meteor har precis gått över 1.0 och kommit till sin första större version. Phonegap är snart på sin tredje större version och Cordova på sin fjärde. De två senare kan alltså ses som mognare än Meteor och det skulle kunna vara för detta som Phonegap och Cordova är bättre underhållet.

6.1.2 Applikationsutvärderingen

Meteor hade färre rader JavaScript-kod, se tabell 7, men koden var komplexare än Cordova och Phonegaps källkod, se tabell 8. Att Meteors kod var mer komplex än de andra två fastän källkoden hade färre logiska rader kod kan vara för att Meteor använder sig av en annan syntax än de andra två, detta kan ses och jämföras i bilaga 1 och bilaga 2. Som källkoden för Meteor i bilaga 1 visar så använder sig Meteor av egen funktionalitet för att hantera händelser (event) och för att tillgodose applikationens vy med data med hjälp-funktioner (helpers). Phonegap och Cordova använder sig av vanlig JavaScript-syntax med JavaScript-metoder för att lyssna på händelser och hantera dessa.

Heitkötter et al. (2012) beskriver att med hjälp av rader kod kan hanterbarheten av koden utvärderas. Det skiljde bara tre LLOC mellan Meteor och Phonegap och Cordova vilket betyder att det inte heller skiljer mycket i hur svårt det är att hantera koden. Heitkötter et al. (2012) kommer inte med någon gräns för när de anser att koden blir mindre hanterbar men jag själv anser att raderna som mina applikationer landade på är hanterbara. Meteor hade 30 LLOC och

Phonegap och Cordova hade 33 LLOC. Att det blev resultatet blev så likt kan bero på att applikationen som utvecklades inte innehöll mycket funktionalitet.

6.2 Cordova

6.2.2 Första urgallringen

Cordova var det ramverk som samlade på sig minst poäng efter första urgallringen av de ramverk som skulle utvecklas applikation för, som kan ses i tabell 6. Cordova samlade ihop 22 poäng mot Meteor och Phonegaps 27 poäng. Att det blev ett hopp på fyra poäng kan vara för att Cordova egentligen var ramverket med fjärde mest poäng men, som förklarats innan i 5.5, Titanium valde att bli kommersiellt under undersökningens gång och utvecklades därför ingen applikation på.

Punkterna som Cordova tappade poäng mot Phonegap och Meteor var Stackoverflow-resultat och hur underhållet det är där det fick 4 respektive 5 poäng. På punkten GitHub-stjärnor samlade ramverket på sig 6 poäng, vilket gjorde att det ändå var populärare än Titanium på den punkten. Ramverket samlade även på sig 8 poäng på dokumentationen. Att ramverket tappat på de andra punkterna men har en såpass bra dokumentation kan bero på att det är på en så sen version som det är och den då har kunnat genomarbetas.

6.2.2 Applikationsutvärderingen

Cordova samlade på sig lika många poäng som Phonegap. 3 poäng på CK och 2 poäng på LLOC. Att Cordova och Phonegap samlade på sig lika många poäng beror på, som diskuterades i 5.1.2, att de båda använder sig av vanlig JavaScript-syntax för att skriva funktionalitet och inte har någon ramverksspecifik syntax blir koden för de två exakt likadan.

Att Cordovas och Phonegaps källkod var mindre komplex kan bero på att funktionaliteten skrevs per funktion medan Meteor, med sin egen syntax, har mer nästlad kod som bakar in händelselyssnare och hjälpfunktioner. Detta gör Meteors kod mer komplex än Cordova och Phonegaps. Det kan även bero på att Cordova och Phonegaps kod krävde fler LLOC för att dessa två krävde fler funktioner för att utföra det som applikationen krävde, som kan ses i bilaga 2. Fler funktioner med låg komplexitet är enligt Watson & McCabe (1996) något som ökar testbarheten som i sin tur leder till en kod utan fel.

Det skulle vara intressant att se om större och komplexare applikationer än de som utvecklades för undersökningen hade skrivits och om dessa hade gjort att applikationerna fortfarande låg på ungefär samma LLOC men att Meteors kod fortfarande var desto mer komplex än Cordova och Phonegap.

6.3 Phonegap

6.3.2 Första urgallringen

Phonegap samlade på sig lika många poäng under första urgallringen som Meteor gjorde, 27 poäng, som kan ses i tabell 6. Punkten som Phonegap kom både efter Cordova och Meteor på var stjärnor på GitHub. Detta skulle kunna bero på att Phonegap inte är lika stort på GitHub som Meteor och Cordova är. Phonegap hade mindre stjärnor men även färre totala issues, som visas i resultatet i tabell 4. Meteor och Cordova hade 2964 respektive 7596 totala issues mot Phonegaps 267. Att ramverket inte har fler issues än så skulle kunna relateras till att det inte har lika många som följer det på GitHub som Meteor och Cordova. När det är mindre användare som följer ramverket på GitHub finns det mindre användare som är inne och rapporterar fel på GitHub.

Phonegap var det ramverket som hade mest poäng på punkten som gällde resultatet på Stackoverflow. Detta kan bero på, som diskuterades i 6.1.1, att Phonegap är på en senare version än Meteor och därför kunnat samla på sig en större skara följare som hjälps åt att svara på frågor om hur de har gått till väga när de stött på problem.

Phonehaps dokumentation samlade på sig lika många poäng som Cordova och Meteor. Som diskuterades i 6.1.1 så är Phonegap och Cordova mognare än Meteor men intressant att se att Meteors dokumentation trots att ramverket precis gått över på sin första större version.

6.3.2 Applikationsutvärderingen

Då Phonegap och Cordovas källkod blev likadan har den redan diskuterats i 6.2.2.

6.4 Totalanalys

Utifrån resultatet är Meteor och Phonegap ramverken som slutar på första plats med lika många poäng. Källkoden för de två ramverken kom nära varandra i LLOC, 30 LLOC respektive 33 LLOC, men ändå skiljde mycket i hur komplex koden var där Meteor hade 6 i CK mot 1 i Phonegaps CK. Med hänsyn till detta istället för poängen visar det att LLOC är lika varandra för alla tre applikationerna men det skiljer i CK. Vilket ändå bör tas hänsyn till.

Det Meteor ligger efter på är hur underhållet det är men att vara så nytt och "omoget" och ändå vara populärare på GitHub än både Phonegap och Cordova måste betyda någonting. Meteor ligger efter men har mognat snabbt.

Även om Meteor och Phonegap fick lika många poäng tycker jag personligen att Phonegap är att föredra då komplexiteten i koden skiljde sig såpass mycket även om raderna logiska kod inte gjorde det. Utöver det så är Phonegap bättre underhållet vilket kan komma från att det är på en senare version än Meteor och har hunnit mogna mer under längre utveckling. Tidigare forskning

av Heitkötter et al. (2012) samt Xanthopoulos et al. (2013) styrker mina teorier om att Phonegap är att föredra efter utvärdering där man inte ser till utseende och känsla på slutprodukten.

7. Slutsatser

Vilket är det “bästa” ramverket för att ta fram en hybridapplikation för en webbutvecklare som saknar kompetens för att göra plattformsspecifika applikationer?

Utifrån resultatet delar Meteor och Phonegap titeln “det ‘bästa’ ramverket för att ta fram en hybridapplikation för en webbutvecklare som saknar kompetens för att göra plattformsspecifika applikationer” utifrån kriterierna:

- hur många stjärnor projektet har på GitHub
- hur många resultat som kom upp vid en sökning på ramverkets namn på Stackoverflow
- hur aktivt underhållet ramverket är
- hur bra dokumentationen uppfyllde IEEE Computer Society (2001) punkter på vad en dokumentation bör innehålla
- logiska rader kod som krävdes för att skriva applikationen
- hur komplex koden är

Å andra sidan är komplexiteten i koden för Meteor och Phonegap stor, sex respektive ett, vilket är stor skillnad när logiska rader kod inte skiljer sig mer än tre rader. Detta betyder att Meteors kod är svårare för en utvecklare att sätta sig in i. Genom att också ta hänsyn till att Meteor idag är på version 1.1.0.2 medan Phonegap är på version 2.9.1 kan betyda att Phonegap har hunnit mogna lite mer än Meteor vilket spelar in på hur väl underhållet ramverket är.

Så även om vi slutade på två ramverk med lika många poäng tror jag att Phonegap ändå är att föredra, vilket också stöds av tidigare forskning av Heitkötter et al. (2012) samt Xanthopoulos et al. (2013).

Referenslista

Alexa Top 500 Global Sites (2015). *The top 500 sites on the web*. [Elektronisk]. Tillgänglig: http://www.alexa.com/topsites/global;2 [2015-04-29]

Apache Cordova (2015). *Apache Cordova.* [Elektronisk]. Tillgänglig: http://cordova.apache.org/ [2015-03-11]

Apache Cordova Documentation (2015). *Apache Cordova Documentation*. [Elektronisk]. Tillgänglig: http://cordova.apache.org/docs/en/4.0.0/ [2015-04-10]

AppGyver (2015). *AppGyver.* [Elektronisk]. Tillgänglig: http://www.appgyver.com/ [2015-03-16]

AppGyver Documentation (2015). *AppGyver Documentation.* [Elektronisk]. Tillgänglig: http://docs.appgyver.com [2015-04-10]

Apple Store (2015). *Apple iPhone.* [Elektronisk]. Tillgänglig: http://store.apple.com/us/buy-iphone/iphone6 [2015-04-16]

Denscombe, M. (2009) *Forskningshandboken - för småskaliga forskningsprojekt inom samhällsvetenskaperna.* Lund: Studentlitteratur.

EnyoJS (2015). *Enyo JavaScript Application Framework Test.* [Elektronisk]. Tillgänglig: http://enyojs.com/ [2015-03-10]
Enyo Documentation (2015). *ENYODOCS*. [Elektronisk]. Tillgänglig: http://enyojs.com/docs/latest [2015-04-10]

Fling, B. (2009). *Mobile Design and Development*. CA, Sebastopol: O'Reilly Media Inc.

Gartner (2015). *Gartner Says Smartphone Sales Surpassed One Billion Units in 2014*. [Elektronisk]. Tillgänglig: http://www.gartner.com/newsroom/id/2996817 [2015-02-20]

Gartner (2014). *Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014*. [Elektronisk]. Tillgänglig: http://www.gartner.com/newsroom/id/2944819 [2015-02-20]

GitHub (2015a). *AppGyver.* [Elektronisk]. Tillgänglig: https://github.com/AppGyver [2015-02-20]

GitHub (2015b). *We build Ionic.* [Elektronisk]. Tillgänglig: https://github.com/driftyco/ionic [2015-02-22]

GitHub (2015c). *MoSync.* [Elektronisk]. Tillgänglig: https://github.com/MoSync/MoSync [2015-02-23]

GitHub (2015d). *Meteor Development Group.* [Elektronisk]. Tillgänglig: https://github.com/meteor/meteor [2015-02-25]

GitHub (2015e). *Apache Cordova.* [Elektronisk]. Tillgänglig: https://github.com/apache/cordova-cli/ [2015-02-25]

GitHub (2015f). *PhoneGap.* [Elektronisk]. Tillgänglig: https://github.com/phonegap [2015-02-28]

GitHub (2015g). *Titanium.* [Elektronisk]. Tillgänglig: https://github.com/appcelerator/titanium [2015-02-28]

Goadrich, M. &. Rogers, M. (2011). *Smart smartphone development: iOS versus android. In Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, New York, NY, USA, 607-612.

Heitkötter, H., Hanschke, S., & Majchrzak, T. (2012). Comparing Cross-platform Development Approaches for Mobile Applications. In *Proceedings of the 8th International Conference on Web Information Systems and Technologies (WEBIST)*, Porto, Portugal, 299–311

IEEE-SA. (2015) *IEE STANDARDS ASSOCIATION.* [Elektronisk]. Tillgänglig: http://standards.ieee.org/ [2015-05-20]

IEEE Computer Society. (2001). *IEEE Standard for Software User Documentation.* IEEE Press, New York, NY.

Ionic (2015). *Ionic: Advanced HTML5 Hybrid Mobile App Framework.* [Elektronisk]. Tillgänglig: http://ionicframework.com [2015-03-16]

Ionic Documentation (2015). *Ionic Documentation.* [Elektronisk]. Tillgänglig: http://ionicframework.com/docs/ [2015-04-10]

Meteor (2015). *Meteor.* [Elektronisk]. Tillgänglig: https://www.meteor.com/ [2015-03-11]

Meteor Documentation (2015). *Meteor - Documentation.* [Elektronisk]. Tillgänglig: http://docs.meteor.com [2015-04-10]

MoSync (2015). *Creative native apps for multiple platforms.* [Elektronisk]. Tillgänglig: http://www.mosync.com/ [2015-03-16]

MoSync Documentation (2015). *MoSync Developers*. [Elektronisk]. Tillgänglig: http://www.mosync.com/docs/index.html [2015-04-10]

Mozilla Developer (2015a). *HTML (HyperText Markup Language).* [Elektronisk]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Web/HTML [2015-02-31]

Mozilla Developer (2015b) *CSS.* [Elektronisk]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Glossary/css [2015-02-31]

Mozilla Developer (2015c) *JavaScript* [Elektronisk]. Tillgänglig: https://developer.mozilla.org/en-US/docs/Glossary/JavaScript [2015-02-31]

Mozilla Developer (2015d). *AJAX* [Elektronisk]. Tillgänglig:. https://developer.mozilla.org/en/docs/AJAX [2015-04-16]

MySQL (2015). *What is MySQL?* [Elektronisk]. Tillgänglig: https://dev.mysql.com/doc/refman/4.1/en/what-is-mysql.html [2015-04-29]

Node.js (2015). *Node.js.* [Elektronisk]. Tillgänglig: https://nodejs.org/ [2015-04-16]

Patel, R., Davidson, B. (2011). *Forskningsmetodikens grunder: att planera, genomföra och rapportera en undersökning.* Lund: Studentlitteratur.

PhoneGap (2015). *PhoneGap.* [Elektronisk]. Tillgänglig: http://phonegap.com/ [2015-03-11]

PhoneGap Documentation (2015). *PhoneGap API Documentation.* [Elektronisk]. Tillgänglig: http://docs.phonegap.com/en/4.0.0/index.html [2015-04-10]

PHP.net (2015). *PHP: What is PHP?.* [Elektronisk]. Tillgänglig: *http://php.net/manual/en/intro-whatis.php.* [2015-02-31]

Statista (2015). *Global smartphone shipments by operating system from 1st quarter 2011 to 4th quarter 2014 (in million units).* [Elektronisk]. Tillgänglig: http://www.statista.com/statistics/235359/global-smartphone-operating-system-shipments-by-mobile-operating-system/ [2015-04-21]

The WebKit Open Source Project (2015). *The WebKit Open Source Project.* [Elektronisk]. Tillgänglig: https://www.webkit.org/ [2015-04-21]

Titanium (2015). *Titanium*. [Elektronisk]. Tillgänglig: http://www.appcelerator.com/titanium/ [2015-03-10]

Titanium Documentation (2015). *Titanium 3.X - Appcelerator Docs.* [Elektronisk]. Tillgänglig: http://docs.appcelerator.com/titanium/latest/ [2015-04-10]

Vetensksrådet (2015). *Forskningsetiska principer inom humanistisk-samhällsvetenskaplig forskning.* [Elektronisk]. Tillgänglig: http://www.codex.vr.se/texts/HSFR.pdf [2015-04-22]

W3C (2015a). *SVG.* [Elektronisk]. Tillgänglig: http://www.w3.org/Graphics/SVG/ [2015-04-16]

W3C (2015b). HTML, *The Web's Core Language.* [Elektronisk]. Tillgänglig: http://www.w3.org/html/ [2015-02-31]

W3C (2015c). *XML.* [Elektronisk]. Tillgänglig: http://www.w3.org/standards/xml/core.html [2015-04-16]

W3C (2015d). *DOM.* [Elektronisk]. Tillgänglig: http://www.w3.org/DOM/#what [2015-04-16]

Watson, A.H. & McCabe, T.J. (1996) *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. NIST Special Publication 500-235. Gaithersburg, 1-124.

Xanthopoulos, S. & Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. In *Proceedings of the 6th Balkan Conference in Informatics.* ACM, New York, NY, USA, 213-220.

Appendix

Bilaga 1 — Källkod Meteor

```
if (Meteor.isClient) {
  Session.setDefault('sensitivity', 15);

  onShake = _.debounce(function onShake() {
    alert("shake");
  }, 750, true);

  Template.hello.helpers({
    error: function() {
      return Geolocation.error;
    }
  });

  Template.hello.events({
    'click .photo': function () {
      var cameraOptions = { width: 800, height: 600 };

      MeteorCamera.getPicture(cameraOptions, function (err, data) {
          if (err) {
              console.log(err);
          }
          else {
              alert("Foto taget");
              Session.set("photo", data);
          }
      });
    },
    'click .getGeo': function(){
      alert(Geolocation.latLng());
    },
});

Meteor.startup(function () {
    if (shake && typeof shake.startWatch === 'function') {
      shake.startWatch(onShake, Session.get('sensitivity'));
      Session.set('watching', true);
    }
    else {
      alert('Shake not supported');
    }
  });
}
```

Figur 3 — källkod för Meteor

Bilaga 2 —Källkod Phonegap/Cordova

```
var app = {
    initialize: function() {
        this.bindEvents();
    },
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },
    onDeviceReady: function() {
        app.receivedEvent('deviceready');
        pictureSource=navigator.camera.PictureSourceType;
        destinationType=navigator.camera.DestinationType;
        shake.startWatch(onShake, 40, onError );
    },
    receivedEvent: function(id) {
        var takePhoto = parentElement.querySelector('.takePhoto');
        takePhoto.addEventListener('click', this.capturePhoto, false);

        var getLocation = parentElement.querySelector('.getLocation');
        getLocation.addEventListener('click', this.getLocation, false);

        var toggleShake = parentElement.querySelector('.toggleShake');
        toggleShake.addEventListener('click', this.shakeToggle, false);
    },
    getLocation: function() {
        navigator.geolocation.getCurrentPosition(disp);
    },
    disp: function(pos) {
        alert(pos.coords.latitude + " " + pos.coords.longitude);
    },
    onPhotoDataSuccess: function(imageData) {
      alert("Foto taget");
    },
    capturePhoto: function() {
      navigator.camera.getPicture(onPhotoDataSuccess, onFail, { quality: 50,
        destinationType: destinationType.DATA_URL });
    },
    onFail: function(message) {
      alert('Kunde inte ta foto');
    },
    onError: function(message) {
      alert('Shake failed');
    },
    onShake: function () {
        alert("shake");
    },
};
```

Figur 4 — källkod för Phonegap/Cordova

Bilaga 3 – Dokumentationsutvärdering

Tabell 10 — resultatet från dokumentationsutvärderingen

Ramverk	Innehålls-förteckning	Identifikat-ionsdata	Proced-urer	Introd u-ktion	Hur dokume n-tationen används	Programvar u-kommandon	Naviga-tion	Sök	Ordli-sta	CONOPS	Felmede lande
Meteor (Meteor Documentation 2015)	X		X	X		X	X	X	X		
AppGyver (AppGyver Documentation 2015)					X		X				
Ionic (Ionic Documentation 2015)	X		X		X	X	X	X			
MoSync (MoSync Documentation 2015)	X		X	X	X	X	X	X			
Cordova (Apache Cordova Documentation 2015)	X		X	X	X	X	X		X		
Phonegap (PhoneGap Documentation 2015)	X		X	X	X	X	X		X		
Titanium (Titanium Documentation 2015)	X		X	X	X	X	X	X			
Enyo (Enyo Documentation 2015)	X		X	X	X	X	X		X		