Santeri Rusila

Alustariippumattomien mobiilisovellusten kehitystavat

Tietotekniikan kandidaatintutkielma

19. huhtikuuta 2016

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Santeri Rusila

Yhteystiedot: sajukaru@student.jyu.fi

Työn nimi: Alustariippumattomien mobiilisovellusten kehitystavat

Title in English: Cross-platform mobile application development approaches

Työ: Kandidaatintutkielma

Sivumäärä: 31+0

Tiivistelmä: Mobiilisovellukset ovat nykyään erittäin suosittuja, mutta ne eivät ole aina saatavilla kaikille mobiilialustoille, sillä mobiilisovellusten kehittäminen erikseen kaikille suosituille mobiilialustoille vaatii paljon ammattitaitoa ja resursseja. On kuitenkin olemassa erilaisia kehitystapoja alustariippumattomien mobiilisovellusten kehittämiseen, ja tämän tutkielman tarkoituksena on selvittää mitä ne ovat ja kuinka ne toimivat. Tutkielmassa tarkastellaan myös kuinka kehitystavan valitseminen vaikuttaa kehitysprosessiin ja mobiilisovelluksen ominaisuuksiin.

Avainsanat: alustariippumaton, mobiilisovellus

Abstract: Mobile applications are very popular nowadays but not all of them are available for all mobile platforms. Developing mobile applications separately for all different mobile platforms requires lots of resources and expertise. There are, however, various different approaches for developing cross-platform mobile applications and the purpose of this thesis is to explain what they are and how they work. This thesis also examines how the selection of a cross-platform development approach impacts on the development process and the end product.

Keywords: cross-platform, application

Kuviot

Sisältö

1 Johdanto

Mobiilisovellukset ovat nykyään suuressa suosiossa, ja siksi sovellukset halutaan saada käytettäväksi kaikille mobiilialustoille, esimerkiksi Android-, iOS- ja Windows-laitteille. Ongelmana on kuitenkin se, että kaikki nämä alustat käyttävät eri kehitysympäristöjä ja ohjelmointikieliä. Olisi siis osattava ohjelmoida useilla eri ohjelmointikielillä, jotta olisi mahdollista julkaista sovellukset kaikille eri mobiilialustoille. Sovellusten kehittäminen ja ylläpitäminen monilla eri alustoille vaatisi siis suuret määrät resursseja ja ammattitaitoa. Heitkötter, Majchrzak ja Kuchen (2013) uskovat, että merkittävät erot ohjelmointirajapinnoissa, kirjastoissa ja ohjelmointikielissä aiheuttavat lähes jopa lineaarisen työmäärän kasvun eri alustoille mobiilisovelluksia kehitettäessä. Tämä on myös johtanut alustafragmentaatioon (engl. *platform fragmentation*), josta muodostuu suurempi ongelma kun sovellukset eivät toimi kohdelustan eri versioissa, varsinkin kun eri alustoille on omat koodipohjat (Appiah ym. 2015, 14). On kuitenkin olemassa kehitysympäristöjä ja -työkaluja, joilla on mahdollista rakentaa sovelluksia eri alustoille vain yhden koodin pohjalta.

Kehittäjät ovatkin alkaneet omaksumaan ohjelmistokehyksiä alustariippumattomien mobiilisovellusten kehittämiseen (engl. *cross-platform mobile app development framework*). Nämä ohjelmistokehykset antavat mahdollisuuden ohjelmoida mobiilisovellusten logiikka kerran korkeatasoisella ohjelmointikielellä ja ne tarjoavat työkalut ohjelmakoodin kääntämiseen eri alustoille. (Boushehrinejadmoradi ym. 2015, 441.) Dalmasso ym. (2013) päättelevät testituloksistaan, että työkalujen valitsemisella voi olla suuri vaikutus mobiilisovelluksen kehittämiseen, osassa työkaluista on suorituskyvyllisiä ongelmia ja osa teknologioista ei tarjoa tarpeeksi kyvykkyyttä. Kaikilla kehitystavoilla on paljon hyviä ja huonoja puolia sekä kehittäjän että sovelluksen käyttäjän näkökulmasta (Ciman ja Gaggi 2014, 429).

Nykyään kehitystyökalut voidaan selkeästi jakaa kahteen eri luokkaan. Ensimmäinen luokka, jota esimerkiksi Boushehrinejadmoradi ym. (2015) kutsuvat web-pohjautuvaksi ohjelmistokehykseksi (engl. *web-based framework*), mahdollistaa alustariippumattomien sovellusten kehittämisen suosittujen web-teknologioiden avulla. Näihin lukeutuvat muun muassa HTML5, JavaScript ja CSS. Tällaisia ohjelmistokehyksiä ovat esimerkiksi Adobe PhoneGap/Cordova, Sencha ja IBM MobileFirst. Toiseen luokkaan lukeutuvat natiivit ohjelmisto-

kehykset (engl. *native frameworks*), joissa yleensä ohjelmoidaan yhdelle alustalle sovellus, joka tämän jälkeen käännetään muille alustoille.

Tässä tutkielmassa kuvaillaan ja vertaillaan erilaisia kehitystapoja alustariippumattomien sovellusten kehittämiseen, ja selvitetään, mitä eroja eri työkaluilla on. Tarkoituksena on myös selvittää, miten toteustustapa vaikuttaa lopputuotokseen ja mitä kriteerejä on otettava huomioon toteutustapaa valittaessa. Tämä aihe on nykyään ja tulevaisuudessa varmasti tärkeä, sillä mobiilisovellusten rooli ihmisten arjessa ei ole ainakaan vähenemässä. Siksi olisi hyvä tuntea erilaisia tapoja toteuttaa sovelluksia helposti monille eri mobiilialustoille. Tutkimuskysymys on: ”Mitä alustariippumattomia ohjelmistokehyksiä on olemassa, mitä eroja niissä on, kuinka ohjelmistokehyksen valinta vaikuttaa lopputuotokseen ja mitä valintakriteerejä on otettava huomioon?”.

Toisessa luvussa käydään läpi kirjallisuuskartoituksen vaiheet, mitä tutkimusmenetelmää käytetään ja mitä lähteitä on alun perin otettu tutkielmaan mukaan. Kolmannessa luvussa käsitellään web-pohjaisten ja natiivien mobiilisovellusten perustavanlaatuisia eroja. Neljännessä luvussa selitetään mitä tarkoitetaan web-pohjaisilla ja natiiveilla ohjelmistokehyksillä ja esitellään molemmista luokista kaksi ohjelmistokehystä esimerkkeinä. Viidennessä luvussa käydään läpi kriteerejä valittaessa kehitystapaa mobiilisovellukselle ja kuudennessa luvussa kootaan yhteen tutkielman pääkohdat ja tehdään niistä johtopäätöksiä.

2 Kirjallisuuskartoitus

Lähteiden etsimiseen on käytetty seuraavia hakusanoja: cross-platform development, cross-platform comparison, cross-platform mobile applications, cross-platform tools. Aiheesta ei ole tehty tutkimusta kovin paljoa, sillä aiheeseen liittyviä artikkeleita tai tutkimuksia on hyvin rajallinen määrä. Hakukoneena on pääasiassa käytetty Google Scholaria. Monien tutkimusten lähteet perustuvat voimakkaasti eri työkalujen dokumentaatioihin. Esimerkiksi Xanthopoulos ja Xinogalos 2013 kuvailevat tutkimuksessaan eri työkalujen teknisiä ominaisuuksia ja vertailevat niitä keskenään. Lisäksi on jonkin verran testattu esimerkiksi eri alustoilla toteutettujen sovellusten muistinkäyttöä (Dalmasso et al. 2013).

Löydetty kirjallisuus

A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications	Xanthopoulos ja Xinogalos (2013)
Testing Cross-Platform Mobile App Development Frameworks	Boushehrinejadmoradi, Ganapathy, Nagarakatte ja Iftode (2015)
Evaluation of cross-platform frameworks for mobile applications	Sommer ja Krusche (2014)
Comparison of cross-platform mobile development tools	Palmieri, Singh ja Cicchetti (2012)
Cross platform app: a comparative study	Andrade, Albuquerque (2015)
Evaluating Cross-Platform Development Approaches for Mobile Applications	Heitkötter, Hanschke, Faily ja Majchrzak (2013)
A comparative analysis of cross-platform development approaches for mobile applications	Xanthopoulos ja Xinogalos (2013)
Mobile development tools and cross-platform solutions	Smutný (2012)

An evaluation framework for cross-platform mobile application development tools	Dhillon ja Mahmoud (2015)
Survey, comparison and evaluation of cross platform mobile application development tools	Dalmasso, Katti, Bonnet ja Nikaein (2013)

Koska mobiilimaailma muuttuu ja kehittyy nopeaa tahtia, on hauissa rajattu vuosilukua siten, että vain vuoden 2012 jälkeen julkaistut artikkelit on valittu mukaan. Tutkimuksia on yritetty etsiä myös rajaamalla vain vuoden 2015 ja sen jälkeen julkaistuja artikkeleita, mutta hakutulokset ovat jääneet vähäisiksi eivätkä tällä kriteerillä löytyneet artikkelit ole vastanneet tutkimusaihettani. Alkuperäisten lähteiden haun jälkeen on löydetty muutamia hyviä artikkeleita forward ja backward searchin avulla. Myös lähteiden lukemisen jälkeen aihe on tarkentunut ja hakuperusteet ovat hieman muuttuneet.

Artikkeleita, joissa on testattu erilaisia alustariippumattomia kehitystapoja on hyvin vähän. Esimerkiksi Boushehrinejadmoradi ym. (2015) väittävät artikkelissaan, että he olisivat ensimmäisiä, jotka ovat testanneet alustariippumattomia ohjelmistokehyksiä. Tämäkin artikkeli on vuodelta 2015, mikä osoittaa, että tutkimusta tästä aihepiiristä ei ole tehty paljoa. Suurin osa artikkeleista keskittyy vertailemaan eri ohjelmistokehysten eroja teknisestä näkökulmasta, eikä teknisiä testejä ole paljoa tehty.

Tutkimusmetodina käytetään kuvailevia ja kvalitatiivisia metodeja, sillä ne sopivat hyvin tähän tutkielmaan. Tutkielmassa kuvaillaan ja vertaillaan eri työkaluja ja teknologioita teknisestä näkökulmasta. Tarkoitus ei ole etsiä parasta vaihtoehtoa alustariippumattomien mobiilisovellusten kehittämiseen, vaan vertailla mitkä ovat kunkin tavan hyvät ja huonot puolet.

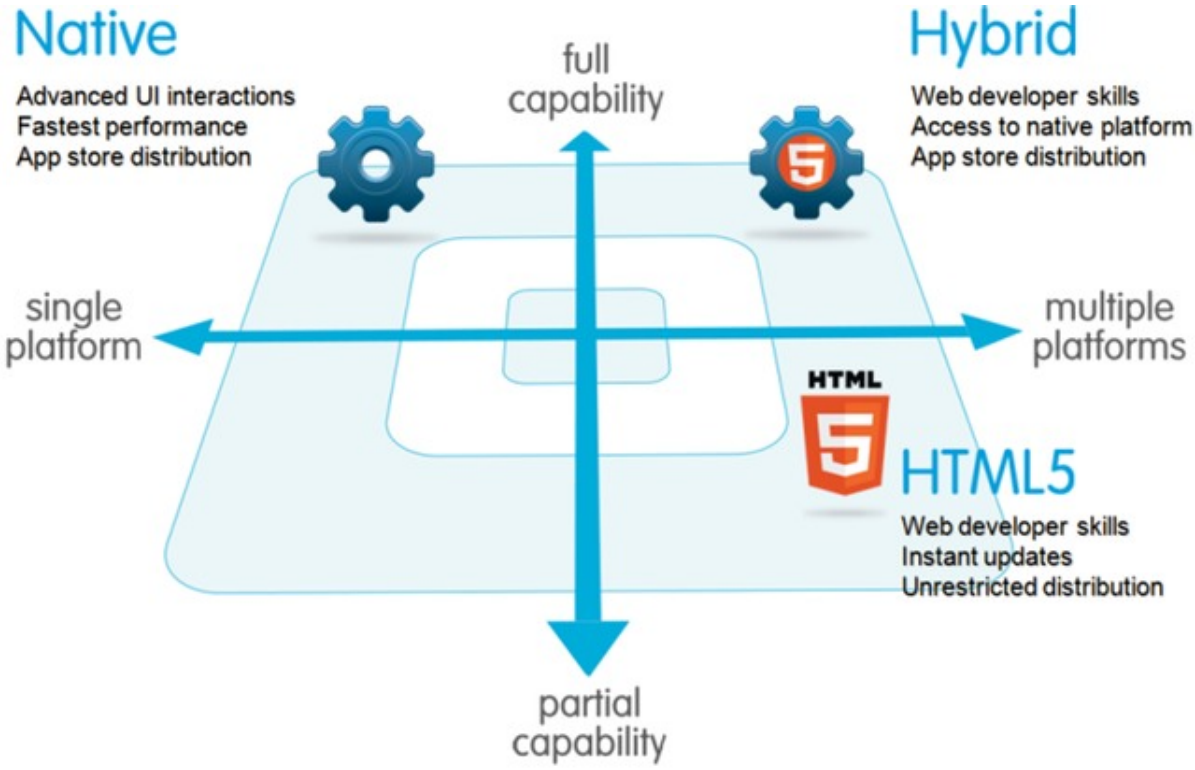
3 Web vs. natiivi

Natiivien teknologioiden ja web-teknologien välinen kuilu on jatkuvasti pienenemässä. Nykyään on mahdollista päästä esimerkiksi PhoneGapin avulla käsiksi älypuhelimien natiiveihin ominaisuuksiin, mikä ei ennen ollut mahdollista. Lisäksi on olemassa kirjastoja, joiden avulla voidaan päästä lähelle natiivia ulkoasua ja käyttökokemusta. Natiivien sovellusten kehittämisestä monimutkaista tekee se, että kaikilla kehitysympäristöillä on omat ohjelmistotyökalunsa (engl. *software development kit, SDK*), rajapintansa (engl. *application programming interface, API*) ja laitteensa, joilla on kaikilla omat kyvykkyytensä. Periaatteessa voidaan sanoa, että mobiilialustoja yhdistää vain mobiiliselain, johon pääsee käsiksi ohjelmallisesti natiivikoodista. Jokainen alusta antaa mahdollisuuden luoda ilmentymän selaimesta, johon natiivilla koodilla ollaan vuorovaikutuksessa, ja jonka sisältä voidaan kutsua natiivia koodia JavaScriptilla. Esimerkiksi PhoneGap hyödyntää tätä tekniikkaa. (Charland ja Leroux 2011, 50.)

Yleensä natiivit koodit ovat valmiiksi käännettyjä, ja siksi ne ovat nopeampia kuin tulkattavat komentosarjakielet kuten JavaScript. WebView ja selain käyttävät HTML:ää ja CSS:ää käyttöliittymien muodostamiseen vaihtelevalla menestyksellä, joten näiden teknologioiden avulla hyvien käyttöliittymien rakentaminen on hankalaa. Natiivilla koodilla pikselit piirretään ruudulle suoraan alustan omien rajapintojen ja abstraktioiden avulla. JavaScript on kuitenkin nopeutumassa jatkuvasti ja siksi esimerkiksi webOS 2.0 kirjoitti palvelukerroksensa uudestaan Javasta suositulle node.js-alustalle. (Charland ja Leroux 2011, 50.)

Kuviossa 1 (Korf ja Oksman 2015) näkyy kuinka natiivin, hybridin ja puhtaan HTML5-pohjaisen web-sovelluksen ominaisuudet eroavat toisistaan. Natiiveilla sovelluksilla on paras suorituskyky ja kehittyneimmät ominaisuudet käyttöliittymän interaktioon, ja niillä pääsee myös käsiksi laitteen sisäisiin ominaisuuksiin varmemmin. Hybridejä sovelluksia kehitettäessä saadaan sovellukset kerralla ladattavaksi useammalle alustalle ja tällöin on myös mahdollista päästä käsiksi laitteen sisäisiin ominaisuuksiin. Tällöin on kuitenkin oltava olemassa plugin, johon on koodattu valmiiksi JavaScript-rajapinta. Hybridisovellus jää kuitenkin käyttöliittymän ulkoasussa ja suorituskyvyssä jälkeen natiiveista sovelluksista. Puhtaat HTML5-sovellukset ovat palvelimella ja niitä voi yleensä käyttää vain jos on yhteydessä In-

ternetiin. Ne myös toimivat kaikilla alustoilla ja päivitykset sovelluksessa näkyvät käyttäjälle heti.



Kuvio 1. Natiivisovelluksen, hybridisovelluksen ja web-sovelluksen ominaisuudet havainnollistettuna (Korf ja Oksman 2015).

3.1 Käyttöliittymä

Monilla natiiveilla alustoilla on omat käyttöliittymänsä ja niihin liittyvät komponentit, eli mikään mobiilikäyttöjärjestelmä ei ole samanlainen, kun puhutaan käyttökokemuksesta. Web-alusta on suurelta osin luotettava, mutta se ei tarjoa samoja mahdollisuuksia kuin natiivi sovellus. Pienet erot selaimien moottoreissa saattavat olla suureksi haitaksi sovelluksien ulkonäössä, mutta nykyään nämä moottorit ovat kuitenkin hyvin lähellä toisiaan ja jäljellä on vain pieniä eroja. Nykyään on tarjolla myös paljon kirjastoja, joiden avulla web-sovellusten käyttöliittymät on helpompi saada näyttämään hyvältä ja toimimaan monille eri alustoille. Tällaisia kirjastoja ovat esimerkiksi jQuery ja jQuery Mobi. Hyvän käyttökokemuksen tavoittelu ei ole pelkästään alustariippumaton ongelma, sillä natiiveja sovelluksia kehitettäessä

on myös saman alustan omaavilla puhelimilla suuria eroja. Web-kehittäjät voivat kuitenkin tukeutua tuttuihin paradigmoihin web-teknologioiden maailmassa. (Charland ja Leroux 2011, 51.)

Web-teknologiat elävät niin sanotussa hiekkalaatikossa, josta on vaikea päästä käsiksi laitteiden sisäisiin ominaisuuksiin ja toimintoihin. Tätä kuilua ollaan kuitenkin jatkuvasti kaventamassa, ja esimerkiksi geolokaatiota (engl. *Geolocation*) on mahdollista käyttää suoraan selaimesta. Muita ominaisuuksia ovat esimerkiksi laitteen värinät ja asennot (engl. *Device Orientation*) (West 2014). Tällä hetkellä PhoneGap tarjoaa rajapinnat käytännössä kaikkiin laitteen sisäisiin ominaisuuksiin, mutta tulevaisuudessa on mahdollista, että nämä ominaisuudet olisivat myös saatavilla suoraan selaimesta. Jos selain ei tue jotain natiivia ominaisuutta, se ei tarkoita etteikö se olisi mahdollista, vaan sitä ei ole vielä tehty mahdolliseksi (Charland ja Leroux 2011, 51).

3.2 Suorituskyky

Sovelluksen suorituskyky on yksi tärkeimmistä asioista käytettävyyden kannalta. Mobiilisovelluksen suorituskykyyn vaikuttaa kaksi tekijää, joista ensimmäinen on latenssi, eli kuinka kauan pakettien lähettäminen Internetin yli kestää, ja toinen on normaalien operaatioiden suoritusaika. Mobiilimaailmassa latenssi on suuri huomion kohde, sillä web-sovelluksen lataaminen Internetin yli vie aikaa, ja natiiveissakin sovelluksissa liikkuu paljon dataa Internetin välityksellä. JSON-formaatissa (*JavaScript Object Notation*) olevan datan on todettu aiheuttavan pienemmän datakuorman verrattuna esimerkiksi XML:n (*Extensible Markup Language*) aiheuttavaan kuormaan. (Charland ja Leroux 2011, 52.)

Yksi aikaa vievistä ongelmista on ladatun koodin parsiminen, jolla voi olla selkeitä vaikutuksia suorituskykyyn. Web-maailmassa ongelmana onkin se, että sovelluksen latauksen jälkeen koodi on tulkattava JavaScriptin luonteesta johtuen. Mitä enemmän on tulkattavaa koodia, sitä kauemmin ohjelman suoritus kestää, jolloin JavaScriptilla jää kirittävää natiiveihin kieliin nähden. On helpompaa ja nopeampaa kehittää web-sovellus, joka toimii monissa eri mobiililaitteissa, mutta se kärsii paljon suorituskyvyssä. Natiiveissa mobiilisovelluksissa suorituskyky on omassa luokassaan, mutta monelle eri alustalle ohjelman koodaaminen vie

paljon aikaa ja resursseja verrattuna web-sovellukseen. (Charland ja Leroux 2011, 52.)

Charland ja Leroux (2011) toteavat, että web-teknologiat eivät ole saavuttaneet sitä suorituskyvyn tasoa, joka on mahdollista saavuttaa natiivilla koodilla. He kuitenkin uskovat, että web-teknologioiden avulla toteutettujen sovellusten ja natiivien sovellusten käyttökokemuksien väliset erot tulevat katoamaan tulevaisuudessa. Sillä välin kun web-teknologiat yrittävät saavuttaa natiivien sovellusten suorituskykyä ja ominaisuuksia, PhoneGap toimii hyvänä siltana näiden teknologioiden välillä.

4 Ohjelmistokehykset

Ohjelmistokehykset voidaan jakaa kahteen eri luokkaan: web-pohjaisiin ja natiiveihin ohjelmistokehyksiin. Tässä luvussa käydään läpi nämä luokat ja selitetään, mitä näillä luokilla tarkoitetaan. Esimerkkeinä molemmille luokille kuvataan kahden eri ohjelmistokehyksen toimintalogiikka. Ohjelmistokehykset on valittu sen perusteella, kuinka hyvin niistä on löytynyt tietoa kirjallisuudesta.

4.1 Web-pohjaiset ohjelmistokehykset

Web-pohjaisiin ohjelmistokehyksiin kuuluvat sellaiset teknologiat, joilla alustariippumattomia sovelluksia toteutetaan tunnettujen web-teknologioiden, kuten esimerkiksi HTML:n, CSS:n ja JavaScriptin avulla. Kehittäjät määrittävät sovelluksen logiikan ja käyttöliittymän käyttäen yhtä tai useampaa web-pohjaista ohjelmointikieltä. Näillä teknologioilla ei kuitenkaan pääse suoraan käsiksi älypuhelimen sisäisiin ominaisuuksiin. Esimerkiksi puhelimen kamera, mikrofoni ja muut ominaisuudet vaativat erillisen ajonaikaisen kirjaston (engl. *runtime library*). Tällaisia sovelluksia kutsutaan yleisesti nimityksellä hybridimobiilisovellus (engl. *hybrid mobile app*). Web-pohjaisilla ohjelmistokehyksillä on helppo toteuttaa prototyyppejä mobiilisovelluksista, mutta ne ovat sopimattomia suorituskykyä vaativille sovelluksille. (Boushehrinejadmoradi ym. 2015, 441.)

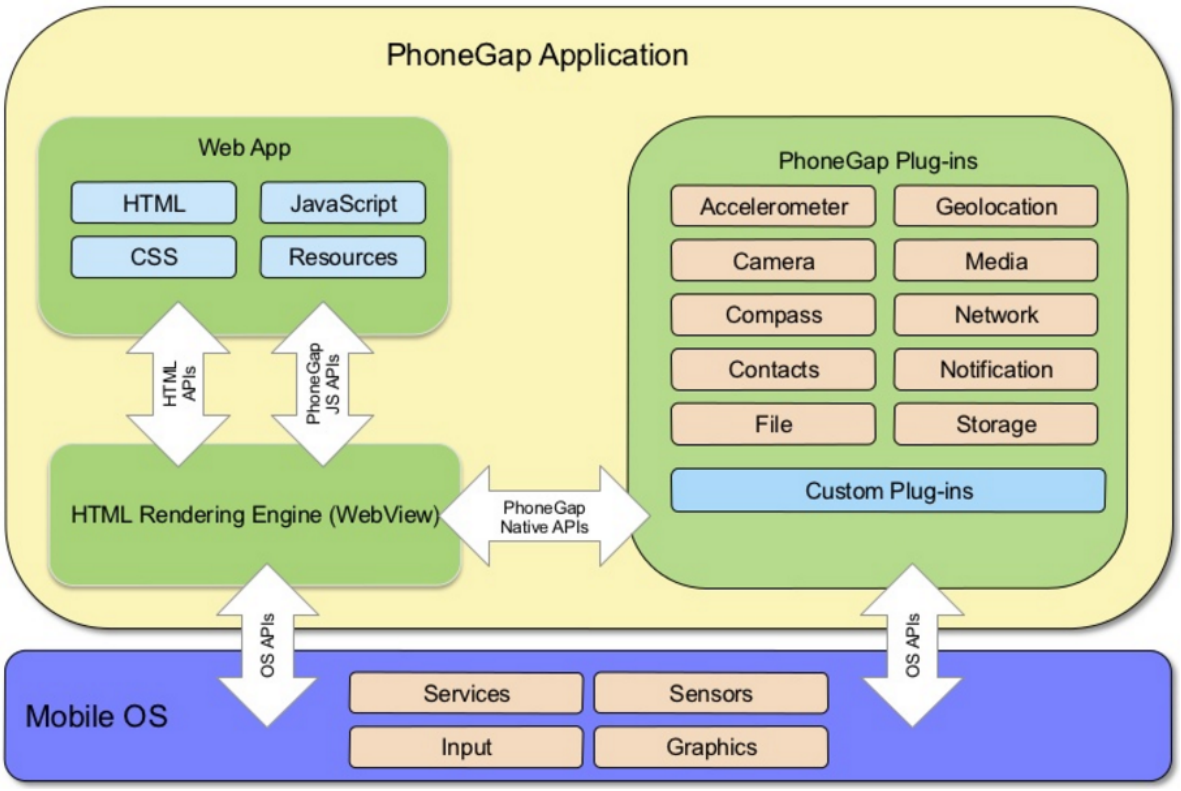
Näiden ohjelmistokehysten vahvuus on siinä, että web-teknologiat ovat suurimmalle osalle kehittäjistä jo ennalta tuttuja. Kehittäjien ei siis tarvitse opetella uusia teknologioita vaan he voivat nopeasti tuottaa alustariippumattomia sovelluksia jo ennalta tunnettujen ohjelmointikielien ja -kirjastojen avulla. Näihin ohjelmistokehyksiin lukeutuvat esimerkiksi Rhodes, PhoneGap, Sencha Touch ja IBM Mobile First. (Boushehrinejadmoradi ym. 2015, 441.) Web-pohjaiset mobiilisovellukset näyttävät kuitenkin yleensä enemmän tai vähemmän normaaleilta mobiilille suunnitelluilta web-sivuilta. Ne voivat olla hyvinkin kehittyneitä, mutta ne eivät yleensä tunnu natiiveilta sovelluksilta (Heitkötter, Majchrzak ja Kuchen 2013, 526).

4.1.1 PhoneGap

PhoneGap on sovelluksen säilöntäteknologia (engl. *application container technology*), joka mahdollistaa natiivisti asennettujen sovellusten tuottamisen web-teknologioiden avulla. Sen ydinmoottori (engl. *Core Engine*) perustuu täysin avoimeen lähdekoodiin ja toimii Apache Cordova -projektin alla (Trice 2012). PhoneGap on käytännöllinen ratkaisu, kun ohjelmoidaan sovelluksia käyttäen moderneja web-ohjelmointikieliä kuten HTML:ää, CSS3:ää ja JavaScriptia sen sijaan, että opeteltaisiin vähemmän tunnettuja ohjelmointikieliä kuten esimerkiksi Objective-C:tä. PhoneGapilla tuotetut sovellukset ovat hybrid-sovelluksia, mikä tarkoittaa sitä, että ne eivät ole puhtaasti natiiveja tai web-pohjaisia. Nämä sovellukset käyttävät hyödykseen WebView-teknologiaa, joka renderöi ulkoasua. (Palmieri, Singh ja Cicchetti 2012, 51.)

PhoneGapin arkkitehtuuri muodostuu pääosin kolmesta kerroksesta: web-sovelluksesta, PhoneGapista, käyttöjärjestelmästä ja natiiveista rajapinnoista. Keskimmäinen kerros on vastuussa web-sovelluksen ja PhoneGap-kerrosten välisestä rajapinnasta, ja se myös hoitaa rajapinnan JavaScript-APIen välillä. Tämän rajapinnan tarkoitus on pitää yllä JavaScript-APIen ja natiivien APIen välisistä relaatioista eri alustojen kanssa. PhoneGap tarjoaa kehittäjille JavaScript-rajapinnan, joka mahdollistaa pääsyn laitteen sisäisiin ominaisuuksiin, joihin lukeutuu esimerkiksi kiihtyvyysanturi, viivakoodinlukija, bluetooth, kamera ja GPS-paikannus. Kuviossa 2 (Palmieri, Singh ja Cicchetti 2012, 182) arkkitehtuuri on näkyvissä tarkemmin. Rajapintoja, joilla päästään käsiksi laitteen sisäisiin ominaisuuksiin kutsutaan plugineiksi (engl. *plug-ins*). Plugineita tehdään ohjelmoimalla natiiveja luokkia ja niitä vastaavia JavaScript-rajapintoja, joita käytetään PhoneGap-sovelluksessa (Trice 2012).

Andrade ym. (2015) tutkivat, kuinka alustariippumaton ja natiivi mobiilisovellus eroavat toisistaan käyttökokemuksessa. Tutkimuksessa yritys kehitti alustariippumattoman sovelluksen PhoneGapin ja The Intel Appin avulla. The Intel App on Intelin tuottama alustariippumaton ohjelmistokehys, joka käyttää hyödykseen HTML5-teknologioita. PhoneGapia käytettiin ytimen komentotulkkina ja The Intel Appia ulkoasun pohjana. Kokonaisuudessaan hybrid-sovelluksen kehitys kesti 12 viikkoa ja vastaavan natiivin Android-sovelluksen 10 viikkoa. Hybrid-sovellus oli kuitenkin tiimin ensimmäinen ja Java-kehitystiimillä oli jo aikaisempaa kokemusta sovellusten kehityksestä. Yrityksen asiakkaille jaettiin sekä puhelimia että tablet-



Kuvio 2. PhoneGap-arkkitehtuuri (Curtis 2011)

teja, joista osassa oli natiivi sovellus ja osassa hybrid-sovellus. Tuloksista kävi ilmi, että vain 13.33 % käyttäjistä huomasi eron natiivin ja hybrid-sovelluksen välillä. Hybrid-sovellus on siis hyvä vaihtoehto kehitettäessä sovelluksia eri alustoille. (Andrade ym. 2015, 38-39.)

4.1.2 Rhodes

Rhodes on Motorola Solutionin kehittämä alustariippumaton ohjelmistokehys. Sillä on mahdollista kehittää sovelluksia iOS:lle, Androidille ja Windows Phonelle käyttämällä ohjelmointikielenä Rubya. Rhodesin teknologioiden ansiosta voidaan rakentaa mobiilisovelluksia monille eri alustoille yhdellä koodipohjalla. Rhodes käyttää MVC-mallia (engl. *Model View Controller*) mobiilisovellusten kehittämiseen. Siinä sovellus on jaettuna kolmeen osa-alueeseen: malli (engl. *Model*) määrittää datarakenteen, näkymä (engl. *View*) määrittää käyttäjälle näkyvän osan, ja kontrolleri (engl. *Controller*) hoitaa ohjelman loogisen toiminnan. Käytäjälle näkyvään osaan käytetään web-teknologioita (HTML, CSS, JavaScript) ja sovelluksen logiikka ohjelmoidaan Ruby:lla. Koska Rhodes perustuu MVC-malliin, sillä on mah-

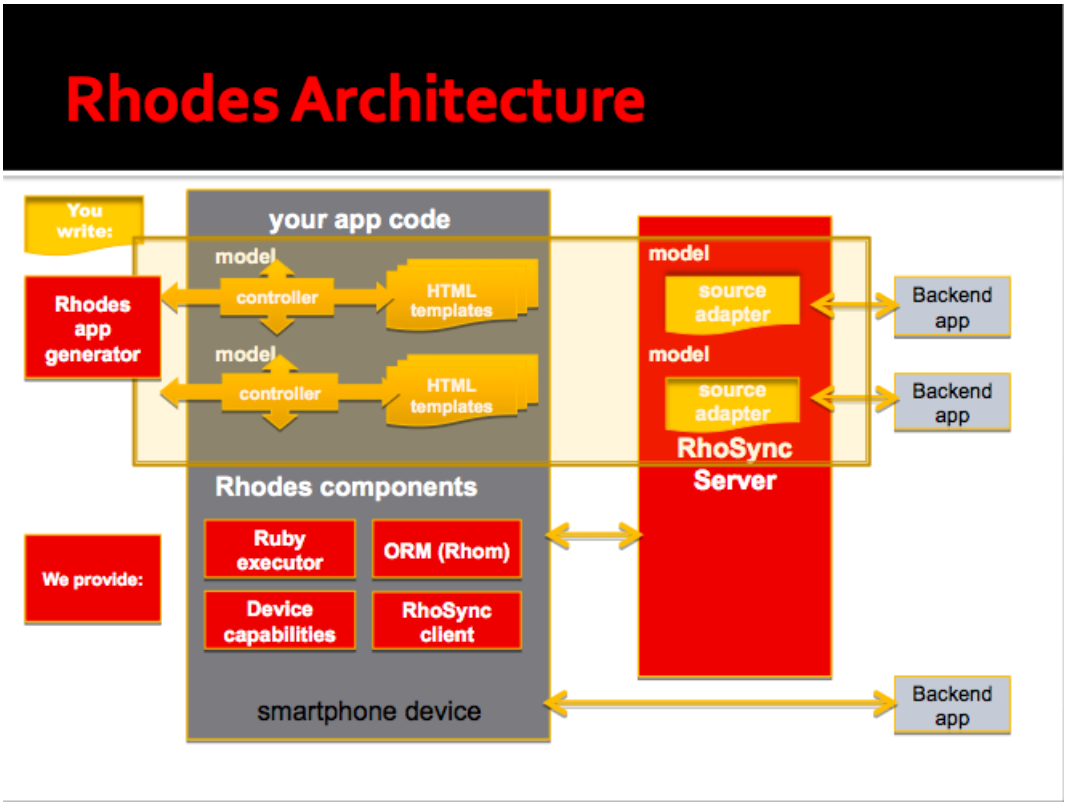
dollista tuottaa mobiilisovelluksia käyttäen pelkästään web-teknologioita, joilla ei ole kuitenkaan mahdollista saada aikaiseksi kovinkaan monipuolista tai monimutkaista sovellusta. (Palmieri, Singh ja Cicchetti 2012, 181.)

Rhodesin rakenne on kuvattuna kuviossa 3 (Leckylao 2010). Kehittäjän on itse ohjelmoitava HTML-templatet, kontrolleri ja source adapteri. Muut komponentit Rhodes tarjoaa kehittäjille valmiina. Muihin komponentteihin kuuluvat Rhodes App Generator, Ruby Excecutor, APIt ja RHOM. Rhodes App Generator on Rhodesin tarjoama kehitysympäristö (engl. *Intergarted Development Environment, IDE*), Ruby Excecutor suorittaa Ruby-koodin, API:t ovat rajapintoja laitteen ominaisuuksiin ja RHOM on pieni tietokanta-ORM (*Object Relational Mapper*). RHOM tarjoaa korkeatasoisen käyttöliittymän tietokantojen suunnittelemiseen ja toteuttamiseen. RhoSync-palvelin on kirjasto, joka lisää mahdollisuuden synkronointiin sovelluksissa, ja se yksinkertaistaa yhteyden muodostamisen palvelinsovelluksiin. Rhodesin kehitystiedostot käännetään natiiveiksi sovelluksiksi, joita voidaan ajaa virtuaalilaitteissa, ja Rhodes itsessään tarjoaa myös työpöytäsimulaattorin, jossa on mahdollista suorittaa sovelluksia. Sovellukset käännetään Javan tavukoodiksi BlackBerry-laitteita varten ja Ruby-tavukoodiksi kaikille muille alustoille. (Palmieri, Singh ja Cicchetti 2012, 181.)

4.1.3 Appcelerator Titanium

Myös Appcelerator Titaniumilla kehitetään sovelluksia web-teknologioien avulla, mutta sillä on tuki teknologioille kuten PHP, Ruby ja Python. Se tarjoaa JavaScript-rajapinnat, jotka käyttävät natiiveja käyttöliittymä -ja alustarajapintoja pääsyssä käyttöliittymäkomponentteihin ja natiiveihin laiteominaisuuksiin. Titanium sisältää Eclipse-pohjaisen käyttöliittymän nimeltä Titanium Studio, joka tarjoaa muun muassa mahdollisuuden sovellusten suorittamiseen tietokoneella. (Ribeiro ja Silva 2012, 257.)

Titanium linkittää JavaScriptin natiiveihin kirjastoihin, kääntää sen tavukoodiksi, ja sitten alustan ohjelmistotyökalu (Android, iOS) rakentaa paketin (engl. *package*) halutulle kohdealustalle. Tämän jälkeen sovellusta voidaan suorittaa joko natiivissa simulaattorissa tai itse laitteessa testausta varten. Lopullinen sovellus sisältää suurelta osin natiivia koodia ja renderöinti suoritetaan natiivisti, lisäksi sovellus sisältää ajonaikaisen JavaScript-tulkin ja Webkit



Kuvio 3. Rhodes-arkkitehtuuri (Leckylao 2010)

renderöintimoottorin. Sovelluksen suoriituksen aikana latauksien suorityskyky on heikompi, koska lähdekoodin tulkkaus laitteessa täytyy tehdä joka kerta, kun sovellus käynnistetään. (Ribeiro ja Silva 2012, 257.)

Titaniumin käytössä on ymmärrettävä kaksi tärkeää käsitettä: ydintoiminnot, jotka voivat olla samanlaisia eri alustojen välillä, jolloin saman koodin uudelleenkäyttö on tärkeää, ja alustakohtaiset ominaisuudet kuten esimerkiksi käyttöliittymät, joihin ohjelmoidaan koodia erikseen alustakohtaisesti. Titanium ottaa JavaScript-koodin ja yhdistää sen Titanium API:iin, joka on kirjoitettu kohdealustojen natiiveilla kielillä, ja JavaScript-tulkki evaluoi koodia sovelluksen ajon aikana. Kun sovellus käynnistetään, luodaan Javascript-suoritin natiiviin koodiin, ja näin Titanium API toimii siltana laitteen natiiveihin ominaisuuksiin oikeiden natiivien rajapintojen kautta. Titanium ei siis käytä samaa WebView-lähestymistapaa, jota PhoneGap hyödyntää, siksi Titanium pystyy tarjoamaan natiiveja UI-kontrolleja ja animaatioita sen sijaan, että se hyödyntäisi CSS:ää. (Korf ja Oksman 2015, .)

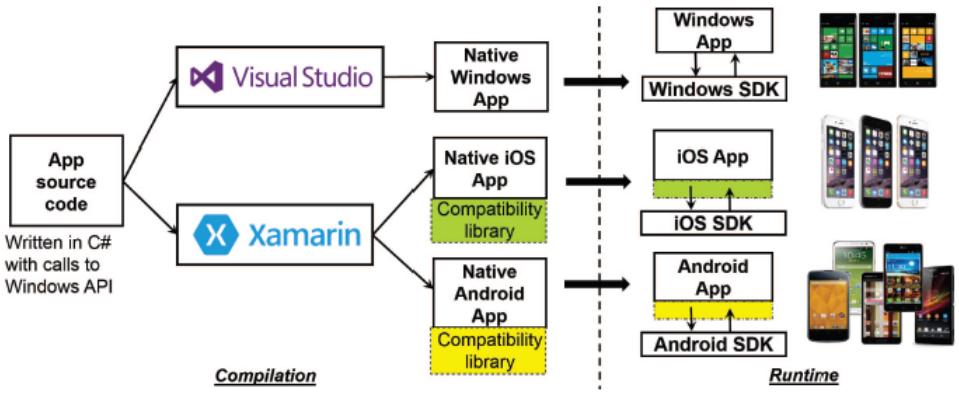
4.2 Natiivit ohjelmistokehykset

Toiseen luokkaan alustariippumattomista ohjelmistokehyksistä kuuluvat natiivit ohjelmistokehykset. Natiiveilla mobiilisovelluksilla tarkoitetaan sitä, että sovellus on kehitetty käyttäen sen omia kehitystyökaluja (engl. *Software Development Kit, SDK*) ja rajapintoja, ja se hyödyntää kyseisen käyttöjärjestelmän sekä laitteiston ominaisuuksia. Natiiveihin ohjelmistokehyksiin lukeutuvat esimerkiksi Xamarin, Apportable ja MD^2. Nämä ohjelmistokehykset tukevat kotialustaa (engl. *home platform*) ja yhtä tai useampaa kohdealustaa (engl. *target platform*). Näiden ohjelmistokehysten idea on siinä, että kehittäjät ohjelmoivat normaalisti sovelluksen kotialustalle, ja tämän jälkeen ne pystyvät tuottamaan sovelluksen kohdealustoille käyttäen kotialustan ohjelmointikieltä. Natiivien ohjelmistokehysten toiminta ei kuitenkaan ole aina näin suoraviivaista, esimerkiksi Xamarinilla tuotetuissa sovelluksissa on tietyt asiat ohjelmoitava kohdealustoille erikseen, koodikielenä kuitenkin jokaisessa käytetään C#:a. (Boushehrinejadmoradi ym. 2015, 441-442.)

Käytettäessä natiivia ohjelmistokehystä mobiilisovelluksen kehittämiseen tavoitteena on, että sovellus toimisi yhdenmukaisesti kotialustalla ja kohdealustoilla. Se riippuu kuitenkin suuresti siitä, kuinka täsmällisesti ohjelmistokehys kääntää kotialustan rajapinta-kutsut SDK:lle verrattuna kohdealustan SDK:hon. Tämä käännös on kuitenkin monimutkainen tehtävä, koska alustan täytyy koodata (engl. *encode*) semantiikat sekä kotialustan että kohdealustan SDK:lle ja relaatiot niiden välille. Tästä monimutkaisuudesta johtuen ohjelmistokehyksissä esiintyy paljon virheitä. (Boushehrinejadmoradi ym. 2015, 442.)

4.2.1 Xamarin

Xamarinin kotialusta on Windows Phone ja kehittäjät ohjelmoivat sovelluksen C#:lla ja Windows Phone SDK:n rajapinnalla. Se mahdollistaa koodipohjan kääntämisen automaattisesti iOS- ja Android-alustoille. Kuviossa 4 (Boushehrinejadmoradi ym. 2015, 442) on havainnollistettuna kuinka tyypillinen työnkulku (engl. *workflow*) toimii Xamarinissa. Boushehrinejadmoradi ym. (2015) käyttivät vain Xamarinia konkreettisena esimerkkinä testeissään, mutta sama yleinen työnkulku on käytössä kaikissa samankaltaisissa natiiveissa ohjelmistokehyksissä.



Kuvio 4. Xamarinin työnkulku (Boushehrinejadmoradi ym. 2015)

Käytettäessä Xamarinia voidaan käyttää paljon samaa koodia kun kehitetään natiiveja sovelluksia iOS:lle ja Androidille. Xamarinin tarjoamia ominaisuuksia ovat esimerkiksi generics, Linq ja parallel task library. Kehittäjä jakaa sovelluksen kahteen loogiseen osaan: sovellusytimeen, joka koodaa toimintalogiikan ja säilöö kaikille alustoille yhteisen koodin, ja käyttöliittymään, joka on kirjoitettu jokaiselle alustalle erikseen ja joka käyttää natiivin alustan käyttöliittymän ominaisuuksia, kuten esimerkiksi nappuloita ja widgettejä. Xamarin pyrkii minimoimaan kohdealustalle porttaamiseen tarvittavan ohjelmoinnin määrän. Xamarin-pohjaisten sovellusten ytimien pääkomponenteista tärkeimpiä ovat alustariippumattomat yhteensopivuuskirjastot (engl. *cross-platform compatibility libraries/portable class libraries*). Esimerkiksi Visual Studiolla ja muilla Microsoftin kehitysympäristöillä on mahdollista tuottaa kirjastoja, joita voidaan jakaa eri alustojen välillä. (Boushehrinejadmoradi ym. 2015, 443.)

Boushehrinejadmoradi ym. (2015) käyttivät testeissään X-Checkeriä, jolla generoitiin 22465 testiä, jotka kutsuivat 4758:aa metodia 354:sta luokasta. Testeissä löytyi 47 uniikkia esiintymää epäjohdonmukaisesta käyttäytymisestä Windows Phonen ja Androidin välillä. Kävi ilmi, että nämä epäjohdonmukaisuudet johtuivat bugeista Xamarinissa. Ongelmat liittyivät esimerkiksi matemaattisten kirjastojen tarkkuuteen ja poikkeuksien semantiikkoihin, eli samat epäonnistuneet metodikutsut antoivat erilaisia poikkeusilmoituksia. Boushehrinejadmoradi ym. (2015) sanovat kuitenkin, ettei voida olla varmoja johtuvatko virheet juuri Xamarinissa esiintyvistä bugeista vai jostain muusta tekijästä. Heillä on suunnitelmissa tarkastella tulevaisuudessa tekniikoita, joilla voisi kutsua metodeita satunnaisilla mutta tarkoituksenmu-

kaisilla parametreilla.

4.2.2 MD^2

Heitkötter, Majchrzak ja Kuchen (2013) esittelevät artikkelissaan MD^2:sta, joka on heidän kehittämänsä ohjelmistokehys alustariippumattomien mobiilisovellusten tuottamiseen, ja joka soveltuu erityisesti liiketoimintasovelluksille. Sen avulla kehittäjät voivat määritellä sovelluksen korkeatasoisella kielellä, joka on suunniteltu kuvaamaan sovellus ytimekkäästi. MD^2 on kehitetty läheisessä yhteistyössä teollisuusalan ammattilaisten kanssa, ja se tarjoaa tapoja kehittää datapohjaisia mobiilisovelluksia, jotka näyttävät ja tuntuvat natiiveilta.

MD^2 käyttää mallipohjaista ohjelmistokehitystä (engl. *model-driven software development, MDSD*), jonka ideana on kuvata ongelma ja generoida sen pohjalta ohjelmisto. MD^2 kehitettiin prototyyppisenä ohjelmistokehyksenä mallipohjaiselle alustariippumattomalle mobiiliohjelmistokehitykselle. Sovellukset määritellään MD^2:lle räätälöidyllä täsmäkielellä (engl. *domain-specific language*) ja ne käyvät läpi muunnosaskeleita kohti natiivia alustakohtaista koodia. Lopulta ne otetaan käyttöön puhtaasti natiiveina sovelluksina kohdealustoilla. MD^2 käyttää MVC-mallia. (Heitkötter, Majchrzak ja Kuchen 2013, 527.)

Heitkötter, Majchrzak ja Kuchen (2013) toteavat, että mallipohjainen kehittäminen sopii hyvin alustariippumattomiin ongelmiin mobiilikehityksessä. Automaattiset muuntamiset generoivat lähdekoodin kehittäjän luomien mallien pohjalta ja muutosvaiheita voi myös olla enemmän, jos koodia generoidaan monelle eri kohdealustalle. Generoidut sovellukset ovat aidosti natiiveja, eivätkä ne kärsi niistä ongelmista, joita web-pohjaisissa lähestymistavoissa esiintyy.

Sovellusten kehittäminen MD^2:lla etenee kolmessa vaiheessa, joista vain ensimmäinen vaatii oikeata työtä kehittäjältä. Aluksi kehittäjä kuvailee ongelman mallina, seuraavaksi koodigeneraattori muuntaa koodin jokaiselle tuetulle kohdealustalle lähdekoodiksi ottaen huomioon myös rakenteelliset elementit, kuten esimerkiksi projektin tiedostot. MD^2 hoitaa kaksi ensimmäistä vaihetta, ja kolmannessa vaiheessa turvaudutaan kohdealustojen tarjoamiin työkaluihin. Ohjelmistokehyksen toinen vaihe on täysin automatisoitu, ja se tapahtuu aina kun kehittäjä tallentaa mallinsa. Jäsennin kokoaa mallista käsitteellisen syntaksin ja tarjo-

aa elementit ja niiden väliset suhteet seuraaville vaiheille koodin generoinnissa. Seuraavaksi Android- ja iOS-koodigeneraattorit käyvät koodin läpi ja generoivat yksilölliset sovellukset kääntämällä mallin kohdealustan lähdekoodiksi. Androidille generoitu koodi on Javaa ja iOS:lle Objective-C:tä, näiden lisäksi MD^2 generoi XML-tiedostot, jotka määrittävät esimerkiksi Androdille käyttöliittymän ja iOS:lle datamallin. Projektitiedostot ja asetukset generoidaan kohdealustan normien mukaisesti, Androidille Eclipseen ja iOS:lle Xcodeen. (Heitkötter, Majchrzak ja Kuchen 2013, 527.)

4.3 Vertailua

Angulo ja Ferre (2014) vertailivat Titaniumilla tuotettua mobiilisovellusta natiiviin sovellukseen He muun muassa testasivat sovellusten nopeuksia ja käyttökokemusta. Tuloksista käy ilmi, että suurin osa testiin osallistuneista kertoi Titaniumilla toteutetun sovelluksen toimivan yleisesti samalla tavalla kuin natiivi sovellus, mutta nopeustesteissä natiivi sovellus oli Titaniumia nopeampi kolmessa viidestä tehtävästä. Natiivien -ja alustariippumattomien sovellusten välillä ei aina ole merkittävää eroa. Alustojen välillä voi käyttäjäkokemuksissa olla eroja, koska esimerkiksi IOS:n natiivit sovellukset ovat toisiinsa nähden homogeenisempia kuin Androidin sovellukset. Angulo ja Ferre (2014) sanovat, että on mahdollista saada sovellukseen aikaiseksi hyvä käyttökokemus, jos ohjelmistokehys valitaan siten, että sillä on mahdollista mukailla kohdealustan käyttöliittymiä. (Angulo ja Ferre 2014, 7-8.)

Ciman ja Gaggi (2014) testaavat tutkimuksessaan eri alustariippumattomillla ohjelmistokehyksillä tuotettujen sovellusten energiankäyttöä, joka on heidän mielestään hyvin tärkeä kriteeri ohjelmistokehystä valittaessa. Tutkimuksessaan he vertailevat PhoneGapia, Titaniumia ja natiivia kehitystapaa. Titanium suoriutuu PhoneGapia paremmin datan visualisoinnissa ja se voi olla hyvä vaihtoehto yksinkertaisissa sovelluksissa, joissa dataa ei haeta laitteen sensoreista. Vaikkakin Titanium näyttää hyvältä vaihtoehdolta niin Ciman ja Gaggi (2014) huomasivat testeissään, että kun Titaniumilla tuotettu sovellus hakee dataa kiihtyvyysanturin sensorista, se kuluttaa 60 % enemmän akkua kuin PhoneGapilla tuotettu sovellus. Yleisesti testituloksissa näkyy, että alustariippumaton kehitystapa lisää suuresti laitteen virrankulutusta, erityisesti kun dataa haetaan laitteen omista sensoreista. Siksi olisi tärkeä miettiä myös sovelluksen energiankuutusta valittaessa ohjelmistokehystä (Ciman ja Gaggi 2014, 375.)

Web-teknologioihin perustuvat Rhodes ja PhoneGap eivät tarvitse koodiin muutoksia eri alustojen välillä, kun taas esimerkiks Titaniumi vaatii pieniä muutoksia ulkoasuun. Web-pohjaisilla ohjelmistokehyksillä on se etu, ne eivät vaadi uuden opettelua kokeneilta web-kehittäjiltä, lukuun ottamatta plugini-rajapintojen käyttöä. Sommer ja Krusche (2013) ovat sitä mieltä, että PhoneGap käyttää parempaa lähestymistapaa kuin Rhodes, koska PhoneGapin plugineilla pääsee käsiksi laitteen natiiveihin ominaisuuksiin. (Sommer ja Krusche 2013, 374-375.)

Appiah ym. (2015) testasivat alustariippumatomien ohjelmistokehysten ominaisuuksia, kuten esimerkiksi kyvykkyyttä, kehitysnopeutta, suorituskykyä ja ulkonäköä. Testissä mukana olleet ohjelmistokehykset olivat Titanium, Xamarin ja PhoneGap. Tuloksista käy ilmi, että PhoneGap on paras vaihtoehto tuottamaan kyvykkäitä sovelluksia, ja se oli käytettävyydeltään helpoin ohjelmistokehys. PhoneGap jää kuitenkin jälkeen muista, kun verrataan sovellusten ulkonäköä.

5 Ohjelmistokehysten valintakriteerit

Heitkötter, Hanschke ja Majchrzak (2013) jakavat tutkimuksessaan ohjelmistokehyksen arviointikriteerit kahteen eri luokkaan: infrastruktuuri -ja kehitysnäkökulmaan. Molemmissa luokissa on seitsemän kriteeriä. Infrastruktuurinäkökulma yhdistää kriteerit liittyen sovelluksen elinkaareen, käyttöön, operaatioihin ja toiminnallisuuteen. Kehitysnäkökulmaan kuuluvat kriteerit, joita ovat esimerkiksi testaaminen, debuggaus ja kehitystyökalut. Heitkötter, Hanschke ja Majchrzak (2013) arvioivat näiden kriteerien perusteella neljää erilaista kehitystapaa: web-sovellusta, PhoneGapia, Titanium Mobilea ja natiivia sovellusta. He antavat jokaiselle kehitystavalle ja jokaista kriteeriä kohden arvosanan väliltä 1-6, 1 ollessa paras ja 6 huonoin.

5.1 Infrastruktuurinäkökulma

Heitkötter, Hanschke ja Majchrzak (2013) luettelevat seitsemän infrastruktuurinäkökulman kriteeriä, jotks koostuvat enemmän sovelluksen käyttäjään vaikuttavita tekijöistä:

1. Lisenssit ja maksut (engl. *License and Cost*)
2. Tuetut alustat (engl. *Supported Platforms*)
3. Pääsy alustakohtaisiin ominaisuuksiin (engl. *Access to platform-specific features*)
4. Pitkäaikainen soveltuvuus (engl. *Long-term feasibility*)
5. Ulkonäkö ja tuntuma (engl. *Look and feel*)
6. Sovelluksen suorituskyky (engl. *Application Speed*)
7. Jakelu (engl. *Distribution*).

Lisenssit ja maksut -kriteerissä otetaan huomioon se, onko ohjelmistokehys jakelussa ilmaisena tai jopa avoimena lähdekoodina ja minkä lisenssin alla se on julkaistu. Tuetuilla alustoilla tarkoitetaan sitä, kuinka monelle alustalle ohjelmistokehyksellä on tuki, ja tukeeko se kaikkia alustoja yhtälaisesti. Kolmannessa kriteerissä otetaan huomioon, kuinka hyvä pääsy ohjelmistokehyksellä on laitteen sisäisiin ominaisuuksiin. Neljännessä kriteerissä otetaan huomioon ohjelmistokehyksen valinta sen perusteella, onko siihen investointi järkevää pitkällä tähtäimellä, huomioiden esimerkiksi kuinka nopeat bugikorjaukset ja kuinka suuri ke-

hittäjäyhteisö ohjelmistokehyksellä on. Ulkonäöllä ja tuntumalla tarkoitetaan sitä, kuinka hyvin ohjelmistokehyksellä tuotettu sovellus vastaa natiivin sovelluksen ulkonäköä ja tuntumaa. Sovelluksen suorituskykyyn lukeutuvat kuinka nopeasti sovellus käynnistyy, ja kuinka responsiivinen se on käyttäjän vuorovaikutukseen. Jakelulla tarkoitetaan kuinka helppoa sovellusta on jakaa, eli onko sovellusta mahdollista esimerkiksi viedä sovelluskauppaan, ja saako sitä helposti pävitettyä. (Heitkötter, Hanschke ja Majchrzak 2013, 125.)

Infrastuktuurinäkökulmasta edellä mainuttujen kriteerien pohjalta Heitkötter, Hanschke ja Majchrzak (2013) arvioivat web-sovellusten pärjäävän parhaiten tuetuissa alustoissa ja pitkäaikaisessa soveltuvuudessa. Kaikissa älypuhelimissa on selain, joten kaikilla niillä on mahdollista päästä käyttämään web-sovellusta. HTML, CSS ja JavaScript ovat hyvin vakiinnettuja teknologioita, joihin tulee päivityksiä tasaisesti. Web-sovellus pärjää huonoiten ulkonäössä ja tuntumassa sekä pääsyssä alustakohtaisiin ominaisuuksiin. Webin kautta ei ole mahdollista käyttää natiiveja käyttöliittymäelementtejä, joten web-sovellusten ulkonäkö riippuu pelkästään CSS:stä. Tähän ongelmaan on kuitenkin olemassa CSS-kirjastoja, ja on mahdollista saada aikaiseksi natiivia muistuttavia käyttöliittymiä. PhoneGap pärjää hyvin siinä missä web-sovelluskin, mutta esimerkiksi Heitkötter, Hanschke ja Majchrzak (2013) ovat arvioineet nopeuden olevan PhoneGapissa hyvä. PhoneGap voittaa web-sovelluksen selvästi pääsyssä alustakohtaisiin ominaisuuksiin pluginien avulla. Titanium Mobilen pääsyä alustakohtaisiin ominaisuuksiin voi pitkälti verrata PhoneGapiin, mutta esimerkiksi ulkonäössä ja tuntumassa se on hieman parempi kuin PhoneGap. (Heitkötter, Hanschke ja Majchrzak 2013, 128-133.)

Natiivien sovellusten vahvuus on kaikissa kriteereissä, paitsi tuetuissa alustoissa. Jos sovelluksia kehitetään erikseen kaikille alustoille niiden omilla kirjastoilla ja rajapinnoilla, vie se paljon aikaa. Heitkötter, Hanschke ja Majchrzak (2013) sanovat, että tämä periaate ei tue alustariippumatonta kehitystä, mutta he eivät ota huomioon natiiveja ohjelmistokehyksiä, kuten esimerkiksi Xamarinia. (Heitkötter, Hanschke ja Majchrzak 2013, 134-135.)

5.2 Kehitysnäkökulma

Kehitysnäkökulman kriteeriluokkaan kuuluvat kehittäjää koskevat arviointikriteerit, johon Heitkötter, Hanschke ja Majchrzak (2013) luettelevat seuraavat:

1. Kehitysympäristö (engl. *Development environment*)
2. Käyttöliittymän suunnittelu (engl. *GUI Design*)
3. Kehityksen helppous (engl. *Ease of development*)
4. Ylläpidettävyys (engl. *Maintainability*)
5. Skaalautuvuus (engl. *Scalability*)
6. Mahdollisuudet jatkokehitykselle (engl. *Opportunities for further development*)
7. Kehityksen nopeus ja hinta (engl. *Speed and Cost Development*).

Kehitysympäristöllä tarkoitetaan sitä, kuinka hyvät ominaisuudet kehitysympäristöllä on, esimerkiksi minkälaisen käyttöliittymän, debuggerin ja emulaattorin se sisältää. Käyttöliittymän suunnittelulla arvioidaan kuinka helppoa ohjelmistokehyksellä on suunnitella mobiilisovelluksen käyttöliittymä. Kehityksen helppouteen huomioidaan se, kuinka hyvä dokumentaatio ja oppimiskäyrä ohjelmistokehyksellä on. Ylläpidettävyys katsotaan suoraan sen perusteella, kuinka monta koodiriviä toimivan mobiilisovelluksen aikaansaamiseksi vaaditaan, tällöin esimerkiksi ohjelmistokehys, joka vaati vain vähän koodia on uuden työntekijän helppo ymmärtää. Skaalautuvuudella tarkoitetaan sitä, kuinka hyvin isommat kehitystiimit voivat jakaa sovelluksen kehityksen eri osiin. Mahdollisuudella jatkokehitykseen -kriteerissä kiinnitetään huomiota siihen, kuinka hyvin ohjelmistokehyksessä tuotettua koodia voidaan käyttää toisessa ohjelmistokehyksessä. Viimeinen kriteeri ottaa huomioon kehityksen nopeuden ja sen suhteen kehittäjän palkkaan, tässä voidaan huomioida esimerkiksi se, onko JavaScript- tai Javakehittäjällä erisuuruiset palkat. (Heitkötter, Hanschke ja Majchrzak 2013, 126.)

Kehitysnäkökulman kriteerien pohjalta Heitkötter, Hanschke ja Majchrzak (2013) arvostelevat web-sovelluksen hyväksi vaihtoehdoksi kehittäjän kannalta, ja se saa kaikista kriteereistä hyvän arvosanan. Esimerkiksi web-teknologioilla kehittämiseen on tarjolla paljon hyviä kehitysympäristöjä, jotka mahdollistavat lyhyen ja elegantin koodin kirjoittamisen. PhoneGap pärjää myös siinä missä web-sovelluskin mutta jatkokehityksen mahdollisuudet eivät ole niin

hyvät, kuitenkin monia samoja teknologioita voi käyttää vastaavassa web-sovelluksessa. Testeissä Titanium saa selkeästi PhoneGapia ja web-sovellusta huonommat arvosanat, sen käyttäminen vaatii paljon opettelua, eikä sillä kirjoitettu ohjelmakoodi ole kovin uudelleenkäytettävää, sillä Titaniumilla on oma sisäinen logiikkansa. Titaniumilla on kuitenkin hyvät edellytykset toimia isommissa projekteissa, koska sen modularisointi on helppoa, esimerkiksi sillä on mahdollista liittää erillisiä tiedostoja projektiin ja siks se voidaan helposti pilkkoa osiin. (Heitkötter, Hanschke ja Majchrzak 2013, 128-133.).

Natiivien sovellusten kohdalla Heitkötter, Hanschke ja Majchrzak (2013) antavat kaikille paitsi jatkokehitykselle, nopeudelle ja hinnalle hyvät arvosanat (1-3). Natiivien sovellusten kehityksen huonot puolet ovat huonot mahdollisuudet jatkokehitykselle ja kallis hinta. Natiiveja mobiilisovelluksia ohjelmoidessa Heitkötter, Hanschke ja Majchrzak (2013) sanovat, että niitä ei yleisesti voida siirtää toiselle alustalle. Tätä ongelmaa varten onkin olemassa natiiveja ohjelmistokehyksiä, joilla porttaaminen toiselle alustalle on mahdollista, mutta he eivät ota sitä tutkimuksessaan huomioon. Natiiveja sovelluksia kehitettäessä kehittäjältä vaaditaan hyvää osaamista ja kokemusta, varsinkin jos sovellus halutaan julkaista natiivina monelle eri alustalle. (Heitkötter, Hanschke ja Majchrzak 2013, 134-135.).

6 Yhteenveto ja johtopäätökset

Tässä tutkielmassa oli tarkoituksena selvittää, mitä kaikkia erilaisia tapoja alustariippumattomien sovellusten kehittämiseen on, kuinka ne eroavat toisistaan ja mitä valintakriteerejä on otettava huomioon kehitystapaa valittaessa. Tutkimusten perusteella voidaan sanoa, että kehitystavan valinnalla voi olla suuri merkitys mobiilisovellusten kehitysprosessiin, ja siihen millainen sovellus sillä voidaan saadaan aikaiseksi. Kehitystavan valitseminen perustuu suurelta osin siihen, halutaanko päästä helpolla, jolloin mobiilisovellus kärsii suorituskyvyssä ja ulkonäössä, vai valitaanko tapa, joka vaatii enemmän ammattitaitoa ja resursseja, mutta jonka lopputuloksena on mahdollista saada aikaan kauniimpi ja tehokkaampi mobiilisovellus.

Alustariippumattomien mobiilisovellusten kehitystavat jakautuvat pääpiirteittäin kahteen luokkaan: web-pohjaisiin ohjelmistokehyksiin ja natiiveihin ohjelmistokehyksiin. Web-pohjaiset mobiilisovellukset voidaan kehittää Internetin välityksellä toimivina normaaleina web-sovelluksina tai mobiilissa lokaalisti suoritettavina sovelluksina. Esimerkiksi PhoneGap mahdollistaa tällaisen toimintaperiaatteen. Web-pohjaisten ohjelmistokehysten vahvuus on siinä, että sovellusten kehittäminen on mahdollista käyttäen pelkästää web-teknologioita kuten HTML:ää, CSS:ää ja JavaScriptia, joita pidetään yleisesti tunnetumpina kuin natiiveja teknologioita. Natiiveilla ohjelmistokehyksillä sovellus ohjelmoidaan kotialustalle, jonka jälkeen se käännetään yhteensopivuuskirjastojen avulla kaikille tuetuille kohdealustoille. Joitain osia on ohjelmoitava erikseen kaikille kohdealustoille, mutta voidaan silti käyttää kotialustan ohjelmointikieltä. Näin saadaan aikaiseksi puhtaasti natiiveja sovelluksia monelle eri alustalle.

Web-teknologioilla kehitetyt sovellukset kärsivät yleensä suorituskyvyssä ja ulkonäössä. JavaScriptin luonteesta johtuen suorituskyky ei yllä natiivien sovellusten tasolle, ja ulkonäkö on toteutettava pelkästään CSS:n ja sille tehtyjen kirjastojen, kuten esimerkiksi jQuery Mobilen avulla. Natiiveilla ohjelmistokehyksillä tuotetut sovellukset saadaan käännettyä kohdealustan natiivikoodiksi, jolloin suorituskyky on hyvä ja pystytään mukailemaan kohdealustalle ominaisia tyylillisiä seikkoja. Natiiveista ohjelmistokehyksistä esimerkiksi Xamarinissa on todettu kuitenkin esiintyvän jonkin verran virheitä käännettäessä kotialustan koodia kohdealustalle (Boushehrinejadmoradi ym. 2015). Vaikka web-teknologioilla tuotetuissa alustariippumattomissa mobiilisovelluksissa käyttökokemus ei ole samalla tasolla kuin natii-

veissa mobiilisovelluksissa, saadaan niillä kuitenkin julkaistua kerralla monelle eri alustalle (Dalmasso ym. 2013, 328).

Heitkötter, Hanschke ja Majchrzak (2013) jakavat artikkelissaan ohjelmistokehyksen valintakriteerit kahteen luokkaan sen perusteella, vaikutavatko ne kehittäjiin vai sovelluksen käyttäjiin. Tärkeitä kriteerejä ohjelmistokehyksen valinnassa ovat esimerkiksi sovelluksen jakomahdollisuudet, ylläpidettävyys ja pääsy laitteen sisäisiin ominaisuuksiin. He toteavat, että PhoneGap on kyvykäs vaihtoehto mobiilisovellusten tuottamiseen, koska HTML, CSS ja JavaScript ovat hyvin vakiintuneita teknologioita. Heitkötter, Hanschke ja Majchrzak (2013) eivät kuitenkaan ota tutkimuksessaan huomioon natiiveja ohjelmistokehyksiä.

Alustariippumattomasta mobiilisovelluskehityksestä ei ole paljoa tehty tutkimuksia, joissa testattaisiin ohjelmistokehysten toimintaa. Esimerkiksi Boushehrinejadmoradi ym. (2015) kertovat olevansa ensimmäisiä, jotka ovat testanneet kunnolla natiivia ohjelmistokehystä. Tulevaisuudessa olisi tärkeää tehdä enemmän tutkimusta natiiveista ohjelmistokehyksistä, esimerkiksi siitä, kuinka hyvin ne kääntävät koodin kohdealustalle ja mitä ongelmia tästä voi syntyä.

Lähteet

West, Matt. 2014. *Exploring Javascript Device Apis.* Saatavilla WWW-muodossa, http://blog.teamtreehouse.com/exploring-javascript-device-apis, viitattu 2.4.2016.

Andrade, Paulo RM de, Adriano B Albuquerque, Otávio F Frota, Robson V Silveira ja Fátima A da Silva. 2015. "Cross platform app: a comparative study". *arXiv preprint arXiv:1503.03511.*

Angulo, Esteban, ja Xavier Ferre. 2014. "A Case Study on Cross-Platform Development Frameworks for Mobile Applications and UX". Teoksessa *Proceedings of the XV International Conference on Human Computer Interaction,* 27:1–27:8. Interacción '14. Puerto de la Cruz, Tenerife, Spain: ACM. ISBN: 978-1-4503-2880-7. doi:10.1145/2662253.2662280. http://doi.acm.org/10.1145/2662253.2662280.

Appiah, Felix, J. B. Hayfron-Acquah, Joseph K. Panford ja Frimpong Twum. 2015. "Article: A Tool Selection Framework for Cross Platform Mobile App Development". Published by Foundation of Computer Science (FCS), NY, USA, *International Journal of Computer Applications* 123, numero 2 (elokuu): 14–19.

Boushehrinejadmoradi, N., V. Ganapathy, S. Nagarakatte ja L. Iftode. 2015. "Testing Cross-Platform Mobile App Development Frameworks (T)". Teoksessa *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on,* 441–451. Marraskuu. doi:10.1109/ASE.2015.21.

Charland, Andre, ja Brian Leroux. 2011. "Mobile application development: web vs. native". *Communications of the ACM* 54 (5): 49–53.

Ciman, Matteo, ja Ombretta Gaggi. 2014. "Evaluating Impact of Cross-platform Frameworks in Energy Consumption of Mobile Applications." Teoksessa *WEBIST (1),* 423–431.

Curtis, Bryce. 2011. *PhoneGap Day - IBM, PhoneGap and the Enterprise.* Saatavilla WWW-muodossa, http://www.slideshare.net/drbac/phonegap-day-ibm-phonegap-and-the-enterprise, viitattu 29.2.2016.

Dalmasso, I., S.K. Datta, C. Bonnet ja N. Nikaein. 2013. “Survey, comparison and evaluation of cross platform mobile application development tools”. Teoksessa *Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International,* 323–328. Heinäkuu. doi:10.1109/IWCMC.2013.6583580.

Heitkötter, Henning, Sebastian Hanschke ja Tim A. Majchrzak. 2013. “Web Information Systems and Technologies: 8th International Conference, WEBIST 2012, Porto, Portugal, April 18-21, 2012, Revised Selected Papers”. Luku Evaluating Cross-Platform Development Approaches for Mobile Applications, toimittanut José Cordeiro ja Karl-Heinz Krempels, 120–138. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN: 978-3-642-36608-6. doi:10.1007/978-3-642-36608-6_8. http://dx.doi.org/10.1007/978-3-642-36608-6_8.

Heitkötter, Henning, Tim A. Majchrzak ja Herbert Kuchen. 2013. “Cross-platform Model-driven Development of Mobile Applications with Md2”. Teoksessa *Proceedings of the 28th Annual ACM Symposium on Applied Computing,* 526–533. SAC ’13. Coimbra, Portugal: ACM. ISBN: 978-1-4503-1656-9. doi:10.1145/2480362.2480464. http://doi.acm.org/10.1145/2480362.2480464.

Korf, Mario, ja Eugene Oksman. 2015. *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options.* Saatavilla WWW-muodossa, https://developer.salesforce.com/page/Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options, viitattu 9.4.2016.

Leckylao. 2010. *Agile mobile web development.* Saatavilla WWW-muodossa, http://leckylao.com/2010/06/12/rhodes-framework-agile-mobile-web-development/, viitattu 31.3.2016.

Palmieri, Manuel, Inderjeet Singh ja Antonio Cicchetti. 2012. “Comparison of cross-platform mobile development tools”. Teoksessa *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on,* 179–186. IEEE.

Ribeiro, Andre, ja Alberto Rodrigues da Silva. 2012. “Survey on Cross-Platforms and Languages for Mobile Apps”. *2012 Eighth International Conference on the Quality of Information and Communications Technology* (Los Alamitos, CA, USA): 255–260. doi:http://doi.ieeecomputersociety.org/10.1109/QUATIC.2012.56.

Sommer, Andreas, ja Stephan Krusche. 2013. “Evaluation of Cross-Platform Frameworks for Mobile Applications.” Teoksessa *Software Engineering (Workshops),* 363–376.

Trice, Andrew. 2012. *PhoneGap Explained Visually.* Saatavilla WWW-muodossa, http://www.tricedesigns.com/2012/03/22/phonegap-explained-visually/, viitattu 29.2.2016.