Joni Tuhkanen

React native ja Xamarin alustariippumattomassa mobiilikehityksessä

Tietotekniikan kandidaatin tutkielma

26. toukokuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joni Tuhkanen

Yhteystiedot: joni.s.tuhkanen@student.jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: React native ja Xamarin alustariippumattomassa mobiilikehityksessä

Title in English: React native and Xamarin in cross-platform mobile development

Työ: Kandidaatin tutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 20+0

Tiivistelmä: Kaksi suosituimmista alustariippumattomista sovelluskehyksistä on Xamarin ja React native, jotka molemmat pyrkivät yhdistämään eri mobiilialustojen koodipohjan. Yhteisestä tarkoituksestaan huolimatta Xamarin ja React native toimivat hyvin eri tavoin ja perustuvat täysin eri teknologioihin. Tämä tutkielma käy vuoroittain läpi Xamarin-kehyksen ja React native -kehyksen hyvät sekä huonot puolet, sekä lopuksi vertaa näitä sovelluskehyksiä keskenään. Verrattaviin ominaisuuksiin lukeutuu tärkeimpinä kehyksien kehittämisympäristöt, suorituskyky, suosio ja hinnoittelumalli.

Avainsanat: Xamarin, React native, alustariippumaton mobiilikehitys, mobiilikehitys

Abstract: Two of the most popular cross-platform mobile frameworks are Xamarin and React Native, which both are made to combine the codebase of different mobile platforms. Regardless of their common goal, Xamarin and React Native work very differently and are based on very different technologies. This thesis goes through both the good sides and bad of both Xamarin and React Native, after which it compares these frameworks. The most relevant compared features are development platforms, performance, popularity and pricing model.

Keywords: Xamarin, React native, cross-platform mobile development, mobile development

Sisältö

1 Johdanto

Mobiilialustan jakautuminen kahtia Androidiin ja IOS:iin tuo esiin ongelman kehittää sama ohjelma molemmille alustoille. Tätä helpottamaan on luotu erilaisia teknologioita, joista kaksi ovat Xamarin ja React Native. Xamarin on vuonna 2011 julkaista ohjelmistokehys, joka perustuu C#-kieleen. React native on puolestaan 2015 julkaistu ohjelmistokehys, joka perustuu suosittuun, Javascript-kielen ympärille luotuun React-kehykseen. Molemmat teknologioista pyrkivät ratkaisemaan ongelmaa saman koodipohjan kirjoittamisesta uudelleen useammalle alustalle.

Tutkielma toteutettiin yhdessä Firstbeat-yrityksen kanssa. Motivaatio tutkimukselle lähti heidän tarpeestaan verrata Xamarin-kehystä ja React native -kehystä alustojen välisessä kehityksessä ja löytää heidän käyttöönsä sopiva vaihtoehto.

Tutkielma vertaa Xamarin-kehystä ja React native -kehystä alustariippumattomassa kehityksessä. Tutkielma käy läpi React native -kehyksen ja Xamarin-kehyksen hyvät ja huonot puolet, sekä vertaa Xamarin-kehystä ja React native -kehystä keskenään.

Alustariippumattomat sovelluskehykset mahdollistavat saman lähdekoodin käyttämisen useammalla mobiilialustalla. Tämä erityisesti säästää aikaa kehittämiseltä, kun samaa koodia ei tarvitse tehdä kokonaan uudestaan eri alustoille. Aikaa säästyy myös tuotteen ylläpitovaiheessa, kun uudistuksia ja bugikorjauksia ei välttämättä tarvitse toteuttaa kuin yhteen lähdekoodiin. Ajansäästymisen lisäksi alustariippumattoman mobiilikehityksen sovelluskehykset vähentävät mobiilikehitykseen tarvittavaa taitotasoa, kun suurin osa koodista voidaan kirjoittaa yhdellä koodikielellä (Palmieri, Singh ja Cicchetti 2012). Kehittäjien ajan säästyminen säästää myös oletettavasti rahaa kehittämiskustannuksista. Comentum-yrityksen toteuttamassa tutkimuksessa todettiin, että toteuttamalla sovelluksen alustariippumattomalla sovelluskehyksellä verrattuna natiiviin kehitykseen rahaa säästyi kokonaisuudessaan yli kolmannes. Tutkimuksessa toteutettiin molemmat, Android- ja IOS-versio. Myös Mounaimin toteuttamassa tutkimuksessa todettiin alustariippumattomien sovelluskehyksien käytön ehdottomasti säästävän aikaa ja rahaa (Mounaim ym. 2016).

Alustariippumattomissa sovelluskehyksissä on myös huonoja puolia, kuten niiden suurem-

pi alttius bugeille verrattuna natiiviin kehitykseen (Boushehrinejadmoradi ym. 2015). Alustariippumattomien sovelluskehyksien mukana tulevat lisäkerrokset ohjelmistokehitykseen saattavat myös hankaloittaa virheenjäljitystä verrattuna natiiviin kehitykseen (Eisenman 2017). Mobiilialustojen päivittyminen on myös ongelma alustariippumattomille sovelluskehyksille, sillä sovelluskehykset täytyy aina päivittää vastaamaan uusia muutoksia. Alustojen päivittymisen välillä ja sovelluskehyksien päivittymisen välillä on viivettä, jonka aikana sovelluskehys voi toimia puutteellisesti tai virheellisesti.

Sovelluskehyksien vertaamiseen tässä tutkielmassa käytetään apuna Mounaimin (Mounaim ym. 2016) luomia kriteerejä. Näihin kriteereihin kuuluu ylläpidettävyys, pääsy alustan natiiveihin rajapintoihin, resurssien käytön tehokkuus ja kehittämisympäristöt. Tämän lisäksi kiinnitetään huomiota myös ohjelmistokehyksien maksullisuuteen, koodin jakamiseen alustojen välillä ja sovelluskehyksien suosioon.

Ensimmäisessä käsittelyluvussa käsitellään React native -ohjelmistokehyksen hyviä ja huonoja puolia alustojen välisessä mobiilikehityksessä. Toisessa käsittelyluvussa puolestaan käsitellään Xamarin-ohjelmistokehyksen hyviä ja huonoja puolia. Kolmannessa käsittelyluvussa verrataan React native- ja Xamarin-ohjelmistokehyksiä keskenään.

2 React native

Tässä kappaleessa käsitellään ensiksi React native -sovelluskehyksen perusteet. Tämä jälkeen käydään läpi React native -kehyksen hyvät sekä huonot puolet.

2.1 React native -sovelluskehyksen perusteet

React native on alustojen välisen kehityksen mahdollistava sovelluskehys. React native julkaistiin maaliskuussa 2015 IOS-käyttöjärjestelmälle kehittämistä varten. Syyskuusta 2015 lähtien sovelluskehyksellä on pystynyt myös kehittämään Android-käyttöjärjestelmälle.

React native -komponentit on tarkoitettu jakaa pienempiin komponentteihin. Jokaisella React native -komponentilla täytyy olla render-funktio, jonka kautta komponentti päivittyy. Komponentti pystyy hallitsemaan omaa tilaansa sen state-tietueen avulla. Komponentin vanhempi puolestaan pystyy hallitsemaan lapsikomponenttia sen props-tietueen avulla.

React native perustuu verkkokehitykseen luotuun Reactjs-sovelluskehykseen. React native on hyvin samanlainen Reactjs-sovelluskehyksyn kanssa. React native ja Reactjs käyttävät molemmat Javascript-syntaksia laajentavaa JSX-syntaksia. JSX-syntaksi muistuttaa Html-syntaksia ja sitä käytetään React-koodissa komponenttien kirjoittamiseen. JSX-syntaksi ei ole ainoastaan tarkoitettu tyylittelyyn, vaan siihen pystyy sisällyttämään myös rajatta Javascript-koodia käyttämällä aaltosulkeita (“Introducing JSX” n.d.).

2.2 React native -sovelluskehyksen hyvät puolet

React native kääntää Javascript-koodin natiiviksi koodiksi Javascriptcore-kirjaston sekä eri kielillä kirjoitetun sillan avulla. Javascriptcore on natiivisti osa IOS-alustan Safari-selainta. Android-alustalla Javascriptcore on pakattu yhteen React native -kehyksen kanssa. Silta on oleellinen osa rakennetta, sillä se mahdollistaa kutsujen tekemisen eri alustoilla. Silta myös mahdollistaa alustojen natiivien rajapintojen kutsumisen suoraan Javascript-koodilla.

React native -kehyksestä poiketen Javascript-pohjaiset alustariippumattomat sovelluskehyk-

set käyttävät yleensä apunaan webview-teknologiaa lähdekoodin tulkitsemiseen, sen sijaan että ne ajaisivat natiivia koodia. Webview-teknologian avulla ohjelma ajetaan ohjelmaan sisälletyn selaimen kautta. Webview-teknologia käyttää verkkokehityksen tavoin Html-, Css- ja Javascript-kielillä tehtyä koodia. Koodin ajaminen selaimen kautta käyttää aikaa verrattuna koodin ajamiseen natiivisti. (Eisenman 2017)

Webview-teknologiaan perustuvat sovelluskehykset eivät yleensä pääse käsiksi kaikkiin natiiviin rajapintoihin, vaan niitä rajoittaa samat saatavilla olevat rajapinnat kuin selaimia. Tämän vuoksi kehittäjä ei pääse käsiksi esimerkiksi puhelimen kameraan. Samaten kehittäjän täytyy itse luoda käyttöliittymäkuvakkeet tai käyttää jotakin saatavilla olevaa käyttöliittymäkirjastoa. Tästä seuraa ongelma, sillä käyttöjärjestelmän päivitettyä kuvakkeet ja animaatiot täytyy aina päivittää uusien muutoksien perässä. Jäljiteltyihin kuvakkeisiin ja animaatioihin voi myös jäädä pieniä eroavaisuuksia oikeista natiiveista vastaavista, minkä vuoksi ne voivat tuntua käyttäjästä oudoilta. (Eisenman 2017)

Perustuessaan Reactjs-verkkokehitysteknologiaan, React native -kehyksellä on se etu, että näiden verkkokehittäjien on hyvin helppo oppia sen syntaksi (Eisenman 2017). React-kehyksen suosiosta kertoo se, että yli 30% Stackoverflow-sivun kyselyyn vastanneista verkkokehittäjistä kertoo käyttäneensä React-kehystä (“Developer Survey Results 2019” 2019). Tämän vuoksi React native voi tutun syntaksinsa vuoksi olla helppo valinta verkkokehityksestä mobiilikehitykseen siirtyville.

React native perustuu avoimeen lähdekoodiin, minkä vuoksi sen yhteisö voi kehittää edelleen React native -kehystä. Erityisesti React native -kehyksen kohdalla sen suuri yhteisö on kehittänyt merkittävän määrän ominaisuuksia Facebook-yrityksen kehittämän koodin lisäksi. React native -kehyksellä on Github-sivustolla kuudenneksi eniten kehittäjiä (“The state of the Octoverse” n.d.). Suuren yhteisönsä ja avoimen lähdekoodinsa vuoksi React native pärjääkin markkinoilla kilpailussa maksullisten kilpailijoiden kanssa. Facebook-yrityksen ei tarvitse itse kehittää koodia käyttäjien varoilla, vaan käyttäjät itse kehittävät React native -kehystä. React native kehityksen ollessa käyttäjäjohtoinen se myös vastaa hyvin sitä mitä käyttäjät sen kehitykseltä haluavat. React native myös mahdollistaa käyttäjien kehittää tuen alustojen uusille päivityksille, ennen kuin React native -kehykselle on julkaistu virallinen päivitys (Eisenman 2017).

React native -kehyksellä on kuitenkin myös suuri tukija, Facebook. React native -ohjelmistokehyksestä ei ole olemassa maksullista versiota, vaan se tarjoaa kaikille käyttäjille yhtä kattavat ominaisuudet. React native -kehitystä ei ole myöskään suljettu mihinkään tiettyihin kehittämisympäristöihin, vaan sen kehitykseen voi käyttää vapaavalintaisesti esimerkiksi Visual code- ja Atom-kehittämisympäristöjä.

Axelsson ja Carlström toteuttivat tutkielmassaan kokeen, jossa käyttäjät kokeilivat natiivisti ja React native -kehyksellä toteutettua versiota samasta sovelluksesta. Käyttäjien arviot siitä kumpi sovellus oli toteutettu natiivisti ja kumpi React native -kehyksellä olivat täysin sattumanvaraisia, kunnes sovellukset näytettiin käyttäjille vierekkäin (Axelsson ja Carlström 2016). Tämä tutkimustulos puoltaa React native -kehyksen ja natiivin kehityksen nopeuseron olevan suhteellisen merkityksetön. Jonas Anderssonin toteuttamassa tutkielmassa React native -kehyksellä toteutettu ohjelma todettiin Android-käyttöjärjestelmän osalta hitaammaksi kuin natiivi koodi, mutta nopeuseroa pidettiin merkityksettömänä käyttökokemuksen kannalta (Andersson 2017). Puolestaan jokaisessa Johanssonin ja Söderbergin toteuttamassa testissä React native on Android-alustalla hitaampi kuin natiivi koodi, mutta tutkimus ei kuitenkaan totea React native -kehyksen olevan liian hidas ammattimaiseen kehitykseen (Johansson ja Söderberg 2018).

React native -kehyksen koodin virheenjäljityksen apuna käytetään Expo-sovellusta. Expo-sovellusta ajetaan joko Android- tai IOS-alustalla, samalla kun lähdekoodi ajetaan tietokoneella. Koodi näkyy mobiilialustalla Expo-sovelluksen avulla ajettuna, ja se päivittyy reaaliaikaisesti tallentaessa lähdekoodin muutokset. Reaaliaikainen päivittäminen on suuri etu React native -kehykselle, sillä se säästää paljon aikaa verrattuna uudelleen kääntämisen vaativaan kehitykseen.

React native pystyy jakamaan suuren osan lähdekoodista eri alustojen välillä. Ainakin Facebook on julkistanut sen Ads manager -sovelluksen jakavan 87% lähdekoodista Android- ja IOS-alustojen välillä (Eisenman 2017).

React native -koodi ajetaan alustan pääsäikeen ulkopuolella. Tämän vuoksi React native koodi ei voi jäädyttää käyttöliittymää, vaikka itse React native koodi toimisikin ajoittain hitaasti.

React native -kehykselle on olemassa monia suosittuja käyttöliittymäkirjastoja, kuten Nati-

vebase sekä React native elements. Valmiiden käyttöliittymäkirjastojen avulla kehittäjät voivat säästää aikaa käyttöliittymäkehittämisessä.

2.3 React native -sovelluskehyksen huonot puolet

React native julkaistiin vuonna 2015, eli se on ohjelmistokehykseksi vielä uusi. Lyhyen olemassaolonsa vuoksi sen kokeneimmatkaan kehittäjät eivät ole vielä erittäin kokeneita.

React native on Rasmus Eskolan toteuttamien testien perusteella joissakin tehtävissä selvästi natiivia koodia hitaampi. Erityisesti alustava lataus voi testien mukaan olla jopa kolme kertaa suurempi React native -kehyksellä toteutettuna verrattuna natiiviin koodiin. Toinen oleellisesti enemmän aikaa vievä tapaus on pitkät listat, jotka hahmottuvat huomattavasti hitaammin React native -kehyksellä toteutettuna kuin natiivilla koodilla toteutettuna (Eskola 2018). Syynä tähän on mahdollisesti React native -kehyksen yksisäikeisyys Javascript-kielen tavoin. Yksisäikeisyydestä johtuen React native voi ajaa vain yhtä komentoa kerrallaan, eikä se siis pysty esimerkiksi ajamaan useampaa animaatiota samanaikaisesti.

React native ei myöskään voi jakaa muistia suoraan natiivin koodin kanssa, vaan muistin jakaminen toimii React native -kehyksen sillan kautta. Tämä voi myös osaltaan hidastaa React native -sovelluksen toimintaa, varsinkin jos suuri osa sovelluksesta on luotu natiivilla koodilla. (Alpert 2018)

Mobiilialustojen päivittymisen on ongelma myötä React native ei päivity välittömästi. Alustojen päivittymistä seuraa viive, että React native tukee uusia päivityksiä täysin. Tiiviistä päivitystahdista seuraa toisaalta myös toinen ongelma, sillä avoimen lähdekoodin kirjastojen pitää päivittyä React native -kehyksen perässä.

3 Xamarin

Tässä kappaleessa käsitellään ensiksi Xamarin-sovelluskehyksen perusteet. Tämä jälkeen käydään läpi Xamarin-kehyksen hyvät sekä huonot puolet.

3.1 Xamarin-sovelluskehyksen perusteet

Xamarin-koodi kirjoitetaan C#-kielellä, joka käännetään kokoamisen myötä alustakohtaiseksi natiiviksi koodiksi. C#-kielestä puuttuvien ominaisuuksien osalta Xamarin muistuttaa vahvasti Dotnet-kehystä (“What is Xamarin” n.d.).

Xamarin toimii molemmilla, Android- ja IOS-alustoilla, avoimenlähdekoodin Mono-ajoympäristön päällä. Android-käyttöjärjestelmällä Xamarin-koodi ajetaan yhdessä Mono-ajoympäristön ja Android runtime (ART) -virtuaalikoneen avulla dynaamisesti. IOS-alustalla koodi ajetaan Mono-ajoympäristön ja Objective C runtime -ajoympäristön päällä. IOS-alustalla koodi käännetään ennen ohjelman käynnistämistä C#-kielestä IOS-alustan ymmärtämälle
Assembly-kielelle, sillä IOS ei salli dynaamisesti tuotetun koodin ajamista (“iOS App Architecture” 2017).

Xamarin forms on oleellinen osa Xamarin-kehyksen ominaisuuksia kehittäessä alustojen välisiä sovelluksia. Xamarin forms on kokoelma käyttöliittymäelementtejä, jotka ovat erilaisia alustakohtaisesti. Xamarin forms -kirjaston avulla kehittäjän ei tarvitse itse kutsua alustojen yleisiä natiiveja komponentteja, vaan sen sijaan sijoittaa vain yhden Xamarin forms -komponentin koodiin.

3.2 Xamarin-sovelluskehyksen hyvät puolet

Xamarin on yksi ensimmäisistä alustojen välisen kehityksen mahdollistavista sovelluskehyksistä. Tämän vuoksi sen kehittäjät ovat muiden alustariippumattomien sovelluskehyksien kehittäjiin verrattuna enemmän kokeneita.

Xamarin on Cristian Contaselin toteuttamassa tutkimuksessa noin kymmenen prosenttia hitaampi kuin natiivi koodi (Contasel ym. 2018). Tutkimuksen sovellus keskittyy langattoman kommunikoinnin toteuttamiseen. Xamarin suoriutui tutkimuksessa huomattavasti paremmin kuin verkkopohjaiset kehykset. Lisandro Delían toteuttamassa tutkimuksessa Xamarin suoriutui testin lukuisia muita sovelluskehyksiä paremmin IOS-alustan osalta, mutta toisaalta suoriutui Android-alustan osalta huonommin kuin esimerkiksi verkkopohjaiset alustojen väliset sovelluskehykset (Delia ym. 2015).

Xamarin-kehyksen tehokkuuteen vaikuttaa mahdollisesti sen multisäikeisyys, eli Xamarin pystyy ajamaan useampaa toimintoa samaan aikaan. Xamarin-kehyksen nopeuteen IOS-alustalla vaikuttaa myös sen AOT-kokoaminen (ahead of time), eli Xamarin kokoaa koodin jo ennen ohjelman tämän ajoa. Näin kokoamiselta säästyy aikaa ohjelmaa ajaessa.

Microsoft tukee Xamarin-kehyksellä kehittämistä vahvasti. Xamarin-kehyksen dokumentointi on hyvin perusteellinen, minkä lisäksi dokumentoinnin tueksi on luotu paljon opetusmateriaalia. Microsoft on myös luonut Xamarin-kehykselle ilmaisen App center -järjestelmän, josta voi seurata esimerkiksi sovelluksen kaatumisdataa (“App center crashes” n.d.). Xamarin-kehyksellä on mahdollista jäljittää virheitä IOS-alustalla myös kehittäessä Windows-käyttöjärjestelmällä Microsoft-yrityksen luoman lisäosan avulla. Lisäosa vaatii silti toimiakseen Mac-tietokoneen, sillä lisäosa toteuttaa virheenjäljityksen Mac-tietokoneen kautta (“Remoted iOS Simulator for Windows” n.d.).

Suuri osa Xamarin-koodista voidaan jakaa alustojen välillä. Alustojen välillä jaettavan koodin määrä vaihtelee lähteittäin, Iflexion-artikkelin mukaan jaettavan koodin määrä on parhaimmillaan noin 80% (Vartanova 2019). Xamarin-kehyksen omassa dokumentaatiossa toisaalta sanotaan jaettavan koodin määrän olevan keskimäärin noin 90%.

3.3 Xamarin-sovelluskehyksen huonot puolet

Xamarin-kehyksellä kehitetyn sovelluksen koko voi olla suhteellisen suuri, vaikka itse sovellus olisi pieni. Yksinkertainen ”Hello world” -sovellus voi saavuttaa Xamarin-kehystä käyttäessä jopa 15.8 megatavun koon (“Application package size” n.d.).

Xamarin ei ole yrityskäytössä ilmainen, toisin kuin useat muut alustariippumattoman mobiilikehityksen sovelluskehykset. Maksullisuuden vuoksi Xamarin-kehyksen kehittäjäyhteisö on pienempi, mikä näkyy avoimenlähdekoodin kehityksessä ja Xamarin-kehyksestä löytyvän materiaalin määrässä.

Xamarin-kehyksellä ei voi kehittää muuta kuin kahdessa kehitysympäristössä, jotka ovat Xamarin studio ja Visual studio.

Xamarin-kehyksen suosio on ollut Google trends -datan mukaan laskussa vuodesta 2017 lähtien. Myös kehittäjien suosimalla Stackoverflow-sivustolla toteutetun kyselyn perusteella Xamarin ei kuulu kehittäjien suosituimpiin sovelluskehyksiin (“Developer Survey Results 2019” 2019).

Xamarin-kehyksen päivittämisessä seuraa viive alustojen päivittymisen jälkeen. Xamarin ei myöskään tarjoa kehittäjille mahdollisuutta luoda itse tukea uusille alustojen päivityksille. Viiveestä voi seurata yhteensopivuusongelmia uusimpaan versioon päivitettyjen laitteiden osalta.

4 React native- ja Xamarin-sovelluskehysten eroavaisuudet

Xamarin on julkaistu vuonna 2011 ja React native vuonna 2015. Kokeneimmat Xamarin-kehittäjät ovat siis selvästi kokeneempia kuin kokeneimmat React native -kehittäjät. Toisaalta aikaisempi julkaisuvuosi ei vaikuta näkyvän Xamarin-kehyksen suosiossa, vaan React native on ohittanut Xamarin-kehyksen ainakin Google-hakujen osalta jo vuonna 2017 (“Google Trends” n.d.). Google-hakujen määrien seuraamisessa on tosin erityisesti se heikkous, että Google-sivun käyttö on estetty eräällä valtavalla markkina-alueella, eli Kiinassa. Myös
Stackoverflow-sivulla näkyy React native -kehyksen suurempi suosio, sillä sivustolle on listattu noin 39 000 kysymystä koskien Xamarin-sovelluskehystä ja noin 62 000 kysymystä koskien React native -sovelluskehystä (“Tags” n.d.). React native -kehyksen ja Xamarin-kehyksen perustuessa avoimeen lähdekoodiin, suosio on niille erittäin tärkeää, sillä se näkyy suoraan kehyksen kehittäjien määrässä. React native kehittäjien määrän ollessa Xamarin-kehittäjien määrää suurempi johtaa osaltaan myös siihen, että yrityksen on vaikeampi löytää Xamarin-kehittäjiä kuin React native -kehittäjiä.

React native ja Xamarin molemmat käyttävät Android-alustalla JIT-kokoamista (just-in-time), sillä se on Android-alustan käyttämällä Java-kielellä mahdollista. IOS-alustalla JIT-kokoamisen käyttö ei ole mahdollista, joten IOS-alustalla Xamarin käyttää AOT-kokoamista (ahead of time). React native puolestaan käyttää IOS-alustalla tulkitsijaa (interpreter) Javascript-kielelle tyypillisesti. Xamarin-kehyksen käyttämä AOT-kokoaminen säästää aikaa ohjelmaa käynnistäessä verrattuna React native -kehyksen käyttämään tulkitsijaan, mutta toisaalta AOT-kokoaminen lisää käytetyn massamuistin määrää.

Xamarin-kehyksellä voi kehittää ainoastaan käyttäen Xamarin Studio -kehittämisympäristöä tai Visual Studio -kehittämisympäristöä. Xamarin-kehyksellä luotu koodi mahdollistaa virheenjäljityksen käyttäen Android-emulaattoria. Samaten React Native -kehitystä ei ole solmittu tiettyihin kehittämisympäristöihin, vaan käyttäjä voi valita kehittämisympäristönsä vapaasti. Molemmilla sovelluskehyksillä IOS-käyttöjärjestelmälle kehittäminen vaatii Mac-

koneen käyttöä Apple-yrityksen vaatimusten vuoksi. Xamarin-kehyksen Android-kehitys ei virallisesti tue Linux-käyttöjärjestelmää. React native -kehyksellä on puolestaan mahdollista kehittää
Android-sovelluksia Linux-järjestelmällä käyttäen esimerkiksi Android studio -kehittämisympäristöä.

React native -kehyksellä kehittäessä kehittäjä voi vapaasti valita mitä kehittämisympäristöä haluaa käyttää. Xamarin-kehyksellä kehittäessä kehittäjällä ei ole näin paljoa valinnan vapautta, vaan kehittäjän täytyy valita Visual studio- ja Xamarin studio -kehittämisympäristöjen välillä. Toisaalta Visual studio ja Xamarin studio ovat molemmat vahvasti Microsoft-yrityksen tukemia.

React native -kehyksen suuri etu verrattuna Xamarin-kehykseen on reaaliaikainen päivitys kehitysnäkymässä, joka on mahdollinen React native -kehyksen perustuessa Javascript-kieleen. Expo-sovelluksen avulla React native -kehittäjä näkee sovellukseen luodut muutokset suoraan puhelimen näytöllä. Xamarin puolestaan vaatii sovelluksen uudelleenkokoamisen aina muutoksien jälkeen. Suuremmissa sovelluksissa uudelleenkokoamisaika voi olla jopa useita minuutteja.

Xamarin ja React native molemmat vaativat Mac-tietokoneen IOS-sovelluksien luomiseen Apple-yrityksen luomien rajoituksien vuoksi. Microsoft on kuitenkin luonut Xamarin-kehykselle lisäosan, joka mahdollistaa IOS-sovelluksen testaamisen Windows-konetta käyttäen. Tämä lisäosa ajaa IOS-sovelluksen etänä kehittäjän oman Mac-koneen kautta (“Remoted iOS Simulator for Windows” n.d.). React native -kehyksellä ei löydy Facebook-yrityksen tukemana vastaavaa lisäosaa. Vaikka vastaava lisäosa olisi toteutettuna yhteisön puolesta, se ei välttämättä tarjoa samantasoista vakautta kuin sovelluskehystä tukevan yrityksen oma toteutus.

Contaselin toteuttamassa tutkimuksessa Xamarin-kehyksellä toteutettu sovellus toimii keskimäärin noin 10% hitaammin kuin natiivikoodi (Contasel ym. 2018). React native -kehykselle toteutetussa tutkimuksessa puolestaan kirjoittaja Rasmus Eskola puhuu jossain määrin huonommasta suorituskyvystä verrattuna natiiviin kehitykseen (Eskola 2018). Nämä kehyksille toteutetut tutkimukset poikkeavat paljon toisistaan, joten niiden tuloksien suora vertaaminen ei ole järkevää. Xamarin-kehyksen esimerkkitapauksen tutkielmassa tätä toteutustapaa ver-

rattiin selvästi hitaampaan webview-periaatteeseen, jonka nopeus natiiviin toteutukseen verrattuna oli 35% hitaampi. Myös React native -kehyksen on todettu olevan Xamarin-kehyksen tavoin selvästi webview-tekniikkaa nopeampi. (Eisenman 2017)

Ainoa kirjoittajan löytämä suoraan Xamarin-kehyksen ja React native -kehyksen suorituskykyä vertaava tutkielma vertaa näiden kehyksien animaatioiden suorituskykyä (Biørn-Hansen ym. 2019). Andreas Biørn-Hansenin toteuttamassa animaatiotestissä React native -kehyksen ja Xamarin-kehyksen prosessorin käyttö oli hyvin samantasoista ja samaten sekunnissa näytettävien kuvien määrä. React native kuitenkin pärjäsi Xamarin-kehystä huomattavasti paremmin keskusmuistin käytön osalta (Biørn-Hansen ym. 2019). Xamarin-kehyksen eduksi suorituskyvyn osalta on kuitenkin Xamarin-kehyksen mahdollisuus jakaa muisti natiivin koodin kanssa. React native -kehyksellä tämä ei ole mahdollista React native -kehyksen käyttämän asynkronisen siltamallin vuoksi.

Xamarin ja React native molemmat suoriutuvat hyvin langattomassa kommunikoinnissa. Xamarin-kehyksellä toteutettu Bluetooth-sovellus toimi Contaselin toteuttamassa tutkimuksessa keskimäärin noin 10% hitaammin kuin natiivilla koodilla toteutettu versio (Contasel ym. 2018). Lifhin ja Lidholmin (Lifh ja Lidholm 2018) toteuttamassa tutkielmassa React native -kehyksellä toteutettu sovellus käytti jopa hiukan vähemmän muistia Android-alustalla ajaessa kuin natiivi sovellus, mutta IOS-alustalla käytti natiivia toteutusta selvästi enemmän muistia. Prosessorin käytön osalta React native -kehyksellä toteutettu Android-sovellus suoriutui hiukan natiivia toteutusta huonommin, mutta IOS-alustalla sovellus käytti selvästi enemmän prosessorin tehoja. Tutkijat tosin huomauttavat, että IOS-alustan prosessorin kuormituksessa saattoi piillä jokin React native -kehyksestä riippumaton ongelma (Lifh ja Lidholm 2018).

Toisin kuin React native, Xamarin ei ole suuremmissa yrityksissä ilmainen käyttää. Xamarin-kehyksen maksullisuus kuitenkin mahdollistaa Microsoft-yrityksen käyttää enemmän resursseja Xamarin-kehyksen kehittäjien tarpeisiin, kuten parempaan dokumentointiin. Xamarin-kehykselle on myös luotu Microsoft-yrityksen puolesta useita lisäosia, kuten IOS-kehittämisen Windows-koneella mahdollistava lisäosa.

5 Yhteenveto

Xamarin ja React native ovat molemmat hyviä vaihtoehtoja alustojen väliseen mobiilikehitykseen. Verrattuna toisiinsa, suuria etuja sovelluskehyksissä keskenään ovat Xamarin-kehyksen suuri tuki Microsoft-yritykseltä ja React native -kehyksen suuri yhteisö sekä jatkuvasti kasvava suosio. Xamarin-kehyksen osalta toinen vahvuus on sen kehittämisympäristö, eli Visual studio. Xamarin-kehyksen virheenjäljitys sekä muut kehittäjien kehittämisympäristölle antamat vaatimukset on huomioitu vahvasti Visual studio -ympäristössä, mutta toisaalta kehittäjällä ei ole vapautta valita kehittämisympäristöään, toisin kuin React native -kehyksellä kehittäessä. Xamarin-kehyksen kolmas vahvuus verrattuna React native -kehykseen on sen aikaisempi julkaisuvuosi, minkä vuoksi kokeneimmat Xamarin-kehittäjät ovat React native -kehittäjiä kokeneempia. React native on nuoremmasta iästään huolimatta kuitenkin ohittanut jo Xamarin-kehyksen suosiossa ainakin Google-hakujen ja Stackoverflow-kysymysten määrässä. React native -kehyksen muihin hyviin puoliin verrattuna Xamarin-kehykseen kuuluu se, että React native on täysin ilmainen, kun taas Xamarin on maksullinen suuremmille tiimeille. Xamarin-kehyksen maksullisuus tosin näkyy Microsoft-yrityksen tarjoamien palveluiden määrässä ja laadussa. Suorituskyvyn osalta React native -kehyksen suorituskyky on Xamarin-kehystä parempi kirjoittajan löytämässä tutkimuksessa.

Xamarin ja React native suoriutuvat yhtä hyvin useassa piirteessä. Molemmat sovelluskehykset tarjoavat laadukkaita käyttöliittymäkirjastoja sekä molemmat kehykset pääsevät käsiksi Android- ja IOS-alustojen natiiveihin käyttöliittymiin. Molemmat sovelluskehykset myös suoriutuvat langattomasta kommunikoinnista ilman suurta suorituskyvyn menetystä verrattuna natiiviin kehitykseen. React native ja Xamarin pystyvät jakamaan suurimman osan koodista eri alustojen välillä. Molemmat sovelluskehykset tukevat virheenjäljitystä sekä Android- ja IOS-alustalla. React native ja Xamarin molemmat perustuvat avoimeen lähdekoodiin.

React native ja Xamarin perustuvat eri teknologioihin, Xamarin-kehyksen perustuessa C#-kieleen ja Dotnet-kirjastoon, sekä React native -kehyksen perustuessa Javascript-kieleen ja React-kirjastoon. Molemmat sovelluskehykset pärjäävät hyvin alustariippumattomia sovelluskehyksiä koskevissa vaatimuksissa, minkä lisäksi niillä on toisistaan eroavia hyviä ja huo-

noja puolia. Molemmat sovelluskehykset, Xamarin ja React native, ovat hyviä vaihtoehtoja alustariippumattomaan mobiilikehitykseen, ja kehittäjien täytyy tehdä valinta näiden sovelluskehyksien välillä omiin tarpeisiinsa perustuen.

Lähteet

Alpert, S. 2018. “State of React Native 2018”. (Haettu: 3.12.2019), https://facebook.github.io/react-native/blog/2018/06/14/state-of-react-native-2018.

Andersson, J. 2017. “Using React Native and AWS Lambda for cross-platform development in a startup”.

“App center crashes”. n.d. (Haettu: 4.5.2020), https://docs.microsoft.com/en-us/appcenter/sdk/crashes/xamarin.

“Application package size”. n.d. (Haettu: 2.5.2020), https://docs.microsoft.com/en-us/xamarin/android/deploy-test/app-package-size.

“Architecture”. n.d. (Haettu: 29.12.2019), https://docs.microsoft.com/en-us/xamarin/android/internals/architecture.

Axelsson, O., ja F. Carlström. 2016. “Evaluation Targeting React Native in Comparison to Native Mobile Development”.

Biørn-Hansen, A., T. Grønli, G. Ghinea ja S. Alouneh. 2019. “An Empirical Study of Cross-Platform Mobile Development in Industry”.

Boushehrinejadmoradi, N., V. Ganapathy, S. Nagarakatte ja L. Iftode. 2015. “Testing Cross-Platform Mobile App Development Frameworks”.

Contasel, C., R. Rughinis, D. Rosner ja D. C. Tranca. 2018. “Impact of Cross-Platform Development Frameworks on the Performance of Mobile Communications for Short Distances”. *Teoksessa The 14th International Scientific Conference eLearning and Software for Education Bucharest, April 19-20, 2018.*

Delia, L., N. Galdamez, P. Thomas, L. Corbalan ja P. Pesado. 2015. “Multi-Platform Mobile Application Development Analysis”.

“Developer Survey Results 2019”. 2019. (Haettu: 18.2.2020), https://insights.stackoverflow.com/survey/2019.

Eisenman, B. 2017. “Learning React Native – Building Native Mobile Apps with JavaScript. Second edition.”

Eskola, R. 2018. “React Native Performance Evaluation”.

“Google Trends”. n.d. (Haettu: 10.12.2019), https://trends.google.com/trends.

Hunt, P. 2013. “Why did we build React?” (Haettu: 3.12.2019), https://Reactjs.org/blog/2013/06/05/why-react.html.

“Introducing JSX”. n.d. (Haettu: 1.9.2019), https://Reactjs.org/docs/introducing-JSX.html.

“iOS App Architecture”. 2017. (Haettu: 29.12.2019), https://docs.microsoft.com/en-us/xamarin/iOS/internals/architecture.

Johansson, E., ja J. Söderberg. 2018. “Evaluating performance of a React Native feature set”.

Lifh, O., ja P. Lidholm. 2018. “Recreating a Native Application in React Native — Feasibility of Using React Native With Bluetooth Background Processing”.

Mounaim, L., Y. Lakhrissi, E. H. Nfaoui ja N. Es-Sbai. 2016. “Cross platform approach for mobile aplication development: a survey”.

Palmieri, M., I. Singh ja A. Cicchetti. 2012. “Comparison of Cross-Platform Mobile Development Tools”.

“Part 1 – Understanding the Xamarin Mobile Platform”. n.d. (Haettu: 29.12.2019), https://docs.microsoft.com/en-us/xamarin/cross-platform/app-fundamentals/building-cross-platform-applications/understanding-the-xamarin-mobile-platform.

“Performance”. n.d. (Haettu: 13.12.2019), https://facebook.github.io/react-native/docs/performance.

“Remoted iOS Simulator for Windows”. n.d. (Haettu: 11.1.2020), https://docs.microsoft.com/en-us/xamarin/tools/iOS-simulator/.

Reynolds, M. 2014. “Xamarin mobile application development for Android”.

"Tags". n.d. (Haettu: 10.12.2019), https://stackoverflow.com/tags.

"The state of the Octoverse". n.d. (Haettu: 2.5.2020), https://octoverse.github.com/.

Vartanova, N. 2019. "Xamarin Review: Cross-Platform Mobile Development from a Business Perspective". (Haettu: 12.2.2020), https://www.iflexion.com/blog/xamarin-review.

"What is Xamarin". n.d. (Haettu: 2.5.2020), https://dotnet.microsoft.com/learn/xamarin/what-is-xamarin.

Xiaoping, J. 2018. "A Performance Evaluation of Cross-Platform Mobile Application Development Approaches".