Tampereen yliopisto

Aleksander Ilves

HYBRIDISOVELLUSTEN OHJELMISTOKEHITYS

Mobiilisovellusten tuottaminen alustariippumattomasti

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Elokuu 2022

TIIVISTELMÄ

Aleksander Ilves: Hybridisovellusten ohjelmistokehitys - Mobiilisovellusten tuottaminen alustariippumattomasti
Kandidaatintyö
Tampereen yliopisto
Tietotekniikka
Elokuu 2022

Mobiilisovellusten tuottamiseen on olemassa useita eri teknologioita ja ratkaisuja. Sovellusten päärakenteena toimii kuitenkin joko verkko-, hybridi- tai natiivikehysrakenne. Tässä kandidaatintutkielmassa perehdytään hybridisovellusten rakenteeseen, tuottamiseen ja ominaisuuksiin.

Hybridisovellukset perustuvat yhteen koodipohjaan, jolloin voidaan samanaikaisesti tuottaa sovellus usealle alustalle. Yhdellä koodipohjalla voidaan tuottaa sovellus, joka toimii samanaikaisesti molemmilla suurilla mobiilikäyttöjärjestelmillä Androidilla ja iOS:llä. Näin säästetään merkittävästi tuotantokustannuksissa, sillä kehitysaika vähenee merkittävästi koska kehitys keskittyy vain yhteen jaettuun koodipohjaan. Sovelluksen päivittäminen ja jatkokehittäminen on myös yksinkertaisempaa, sillä muutokset sovelluksen toimintaan tarvitsee tehdä vain yhteen koodipohjaan ja ne voidaan päivittää samanaikaisesti jokaiselle eri alustalle.

Hybridikehysrakenne tuo mukanaan myös kompromisseja ja haittoja, sillä se ei tarjoa yhtä optimoituja ja laitteen ominaisuuksiin integroituja ratkaisuja natiiveihin alustakohtaisiin sovelluksiin verrattuna. Hybridisovellusten käyttökokemus ja nopeus ovat usein myös heikompia verrattaessa natiiveihin alustakohtaisiin sovelluksiin. Hybridikehysrakenteen tuomat edut ovat kuitenkin merkittäviä, joten ohjelmistoprojektin kohdalla on tärkeä miettiä mikä malli sopii kyseiseen projektiin parhaiten.

Työn tuloksena havaittiin, että hybridikehysrakenne vähentää merkittävästi ohjelmistokehityksen kustannuksia, nopeuttaa sovellusten tuotantoa ja yksinkertaistaa ohjelmistoprojektin rakennetta. Hybridikehysrakenne ei kuitenkaan sovellu käyttötarkoituksiin, jossa sovelluksen tehokkuus ja käytettävyys ovat ohjelmistoprojektin pääpainona. Hybridikehysrakennetta on siis syytä harkita ohjelmistoprojektin tavoitteiden ja haluttujen lopputuotteen ominaisuuksien mukaan.

Avainsanat: Hybridisovellus, mobiilisovellus, sovelluskehitys, alustariippumaton ohjelmistokehitys, ohjelmistotuotanto

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

API	Application Programming Interface
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
SDK	Software Development Kit
SDLC	Software Development Life Cycle
TCO	Total Cost of Ownership

1. JOHDANTO

Sovelluskauppoihin lisätään tuhansittain sovelluksia joka päivä. Mobiililaitteiden fyysiset rajoitteet kuten näytön koko ja rajattu suorituskyky kannustavat ohjelmistokehittäjiä tuottamaan erillisiä sovelluksia, selaimessa toimivien verkkosovellusten sijaan. Ladattavat sovellukset tarjoavat käyttäjäkokemuksen, joka on suunniteltu toimimaan mobiililaitteen rajoitteiden mukaan. Erillinen sovellus pystyy myös paremmin hyödyntämään laitteen eri ominaisuuksia, kuten kameraa ja mikrofonia. [1]

Sovellusta toteuttaessa on olemassa kolme eri pääkehysrakennetta: verkko, natiivi ja hybridi. Verkko- ja hybridisovellukset ovat suunniteltu toimimaan usealla alustalla samalla pohjalla ja natiivit sovellukset tuotetaan alustakohtaisesti. Verkkosovellukset toimivat laitteen selaimen kautta, kun taas natiivit ja hybridisovellukset ovat ladattavissa sovelluskaupasta. Osa hybridisovelluksista omaa monia samoja piirteitä verkkosovellusten kanssa, koska tällaiset hybridisovellukset ovat pitkälti verkkosovelluksia, jotka on viety natiiviin ohjelmistokuoreen. Uudempia hybridiviitekehyksiä hyödyntävät hybridisovellukset taas käyttävät osaltaan natiiveja komponentteja. Natiivit sovellukset taas toteutetaan alustakohtaisesti eri käyttöjärjestelmälle ja eivät toimi muilla alustoilla kuin sillä mille ne on tuotettu. [1] Tässä kandidaatintyössä tullaan käsittelemään alustariippumatonta mobiiliapplikaatioiden ohjelmistokehitystä ja erityisesti perehdytään juuri hybridikehysrakenteisten sovellusten tuottamiseen. Työssä käydään läpi hybridisovellusten toiminta, kehysrakenteen vaikutus ohjelmisto tuotantoon sekä rakenteen tuomiin etuihin ja haittoihin.

Hybridisovellukset perustuvat yhteen koodipohjaan, joka viedään eri teknologioita käyttäen eri alustoille. Jaettu koodipohja luodaan usein käyttäen yleisiä verkkoteknologioita, jonka takia hybridisovellusten kehitys ei vaadi alustakohtaisten ohjelmointikielien osaamista. Tämän takia hybridisovellusten kehitys on usein paljon nopeampaa ja yksinkertaisempaa suhteessa natiivien sovellusten kehittämiseen. Myös hybridisovelluksen ylläpitäminen ja päivittäminen yksinkertaistuu, sillä muutoksia tarvitsee tehdä vain yhteen koodipohjaan. Hybridikehysrakenne tuo mukaan myös kompromisseja ja haasteita, sillä hybridisovellukset eivät pysty olemaan yhtä optimoituja ja integroituja mobiililaitteeseen kuin natiivit sovellukset.

Mobiililaitteiden ominaisuudet, käyttöjärjestelmät ja sovellusten ohjelmistokehykset kehittyvät nopeaa tahtia, joten työn lähteiksi on pyritty valitsemaan mahdollisimman tuoreita ja nykyteknologioille ajankohtaisia julkaisuja. Erityisesti hybridiviitekehyksiin

liittyen vanhoja julkaisuja, joissa ei mainita tai huomioida uudempia viitekehyksiä, kuten Flutteria ja React Nativea löytyy paljon. Tämän takia tällaisten vanhojen lähteiden käyttöä on pyritty välttämään. Osa käytetyistä lähteistä ovat näitä uudempia viitekehyksiä vanhempia, koska ne sisältävät yleispätevää ja nykypäivänä yhä relevanttia tietoa sovelluskehityksestä.

Tutkimusmenetelmänä tässä kandidaatintutkielmassa on käytetty kirjallisuuskatsausta. Vertaisarvioituja tieteellisiä tutkimuksia ja julkaisuja on etsitty Andor, Google Scholar, IEEE Xplore tietokannoista. Hakusanoina on käytetty: ”hybrid application”, ”mobile (application OR app) development”, ”(hybrid OR cross-platform) application development”, ”mobile (application OR app) frameworks” ja ”mobile software development”. Lisäksi lähteitä on harkiten valittu tunnettujen ja ohjelmistokehityksen alalla suurten toimijoiden kuten Ionic, Microsoft ja StackOverFlow verkkosivuilta.

Tutkielman alussa luvussa 2 käsitellään hybridisovellusten toimintaa yleisesti. Luvussa 3 selvitetään hybridisovellusten kehityksen eri osa alueita. Luvussa 4 käydään läpi hybridisovellusrakenteen eri haittoja ja hyötyjä. Lopuksi luvussa 5 esitellään päätelmät ja yhteenveto työn sisällöstä.

2. HYBRIDISOVELLUSTEN TOIMINTA

Hybridisovellukset perustuvat yhteen koodipohjaan, jonka kautta pystytään luomaan alustariippumaton sovellus. Koodipohja luodaan käyttäen eri teknologioita käyttäen, ja sovelluskehyksen avulla ohjelma viedään omaksi sovelluksekseen, joka voidaan laittaa sovelluskauppoihin ladattavaksi. Tässä työssä hybridisovelluksella tarkoitetaan sovelluksia, jotka eri hybriditeknologioita hyödyntämällä voidaan tuottaa alustariippumattomasti. Vanhemmat hybridisovellukset perustuvat verkkoteknologioihin, kun taas uudemmilla hybridiviitekehyksillä tuotetut sovellukset mukailevat enemmän natiivia koodia. Tämän takia uudemmat sovellukset, jotka käyttävät alustariippumattomia hybriditeknologioita saatetaan osassa kirjallisuutta määritellä erillisiksi hybridisovelluksista.

2.1 Teknologiat

Mobiilikäyttöjärjestelmien maailmanmarkkinoita hallitsee vuonna 2022 kaksi merkittävää alustaa, Googlen Android noin 71 %:n markkinaosuudella ja Applen iOS noin 28 %:n markkinaosuudella [2]. Näin hallitsevien osuuksien takia mobiilisovellusten kehitys on merkittävissä määrin keskittynyt vain näille kahdelle alustalle. Jotta sovelluksen voi julkaista molemmille alustoille, on joko luotava alustakohtaiset natiivit sovellukset molemmille käyttöjärjestelmille tai hybridisovellus, joka toimii molemmilla alustoilla samalla koodipohjalla [3].

Pääosin hybridisovellusten koodipohja luodaan käyttäen perinteisiä verkkoteknologioita kuten HTML:ää (engl. Hypertext Markup Language), CSS:ää (engl. Cascading Style Sheets) ja JavaScriptiä. Nämä kielet mahdollistavat eri viitekehysten sekä kirjastojen käytön ja ovat tärkeä osa nykyaikaisten verkkosovellusten luontia sekä toimintaa. Hybridisovellus viedään verkkoteknologioiden lisäksi sovelluskehyksen avulla avulla omaksi sovelluksekseen. Näin sovellus voidaan viedä sovelluskauppoihin käyttäjien ladattaviksi, ja on ajettavissa omana erillisenä sovelluksenaan käyttäjien laitteilla. [1]

Hybridisovellukset toimivat eri tavoin riippuen valitusta sovelluskehyksestä. Vanhemmat hybridisovellukset toimivat upotetun verkkoselainelementin (engl. embedded WebView) avulla, joka esittää HTML-sisältöä käyttäjälle. iOS käyttöjärjestelmillä tämä verkkoselainelementti on WKWebView ja Androidilla WebView. Tällöin käyttöliittymä (engl. User Interface, UI) on käytännössä responsiivinen verkkosivu, joka näytetään sovelluksen sisällä. Hybridikehysrakenteita, jotka hyödyntävät tätä lähestymismallia,

ovat esimerkiksi Ionic ja Xamarin [4]. Uudemmat hybriditeknologiat kuten React Native ja Flutter mahdollistavat natiivimman toteutuksen, sillä ne eivät käytä WebView-elementtejä käyttöliittymän näyttämiseksi. Nämä viitekehykset kokoavat näkymän itse laitteessa. Näin sovellus mukautuu paremmin eri laitetyypeille ja pystyy tarjoamaan paremman käyttäjäkokemuksen. [1]

Laitteen selaimessa toimivat verkkosovellukset eivät pysty käyttämään laitteen eri ominaisuuksia kuten kameraa ja gyroskooppia yhtä integroidusti kuin laitteeseen asennetut sovellukset. Natiivit hybridisovellukset pystyvät käyttämään näitä toimintoja laitteen API:en (engl. Application Programming Interface) kautta. Hybridisovellukset eivät kuitenkaan pysty suoraan kanssakäymään laitteen API:en kanssa natiivien sovellusten tapaan vaan vaativat lisäosia (engl. plugins) [4]. Lisäosat mahdollistavat laitteen eri ominaisuuksien hyödyntämisen universaalien ominaisuuskohtaisten API:en kautta, jolloin järjestelmä ja laitekohtaisia ominaisuus API:ja ei tarvita. Hybridisovelluksen ohjelmistokoodissa voidaan siis tehdä universaali API-kutsu, jolloin lisäosa muuntaa kutsun natiiveiksi laitekohtaisiksi API-kutsuiksi. Näin ohjelmistokehittäjän ei tarvitse itse toteuttaa API-kutsuja jokaiselle laitetyypille, vaan kutsut sisältyvät lisäosaan ja niitä pystyttään kutsumaan universaalisti yhdeltä koodipohjalta ja ne toimivat eri alustoilla. [5]

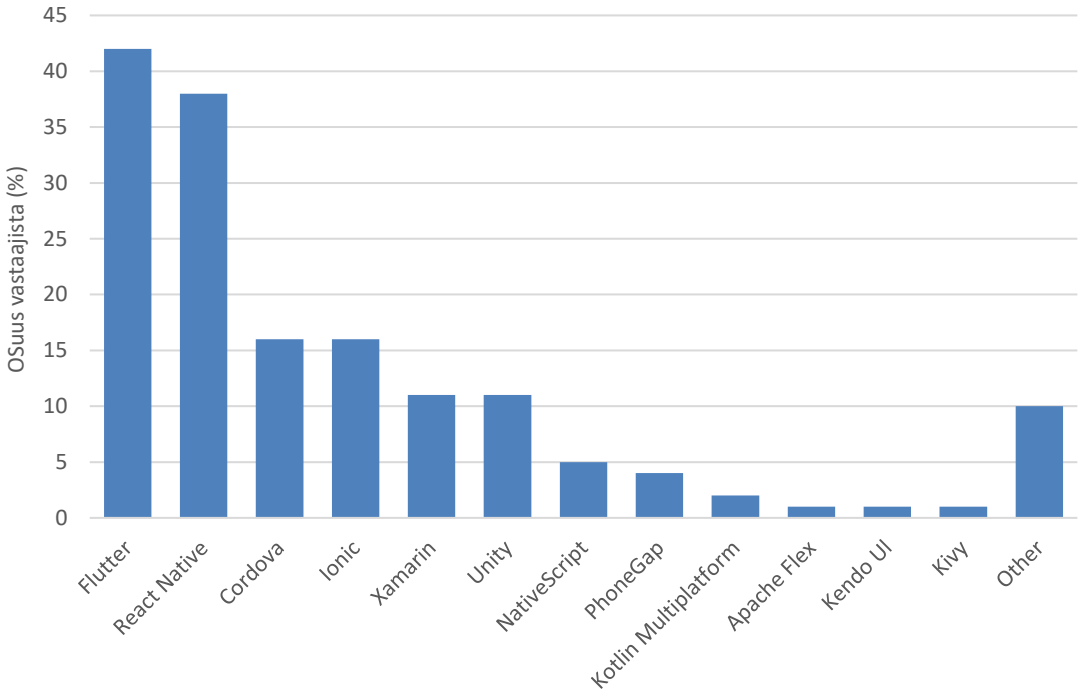
Vaikka useat hybridisovellukset toimivat verkkosovellusten tapaan, niiden käyttö on mahdollista ilman verkkoyhteyttä toisin kuin verkkosovellusten, jotka vaativat internet-yhteyden palvelimeen, jossa toiminnot tapahtuvat. Koska hybridisovellukset on asennettuna laitteelle ja pystyvät lisäosien avulla tekemään laitteelle API-kutsuja, voidaan laitteen muistiin tallentaa paikallinen tietovarasto sovelluksen käytettäväksi. Näin tietovarastoon päästään käsiksi ja sovellusta on mahdollista käyttää, vaikka laitteella ei olisi internetyhteyttä. [4]

2.2 Viitekehykset

Viitekehykset (engl. framework) ovat ohjelmistokirjastoja, jotka luovat perustan ohjelmiston rakentamiselle. Ne tarjoavat vakiotyökaluja ja -rakenteita sovelluksen tuottamiseksi ja pyrkivät näin yksinkertaistamaan sekä helpottamaan ohjelmistoprojektia. Viitekehysten sisältö ja käyttötarkoitukset vaihtelevat paljon, minkä takia on tärkeä valita oikea viitekehys käyttötarkoituksen ja haluttujen ominaisuuksien mukaan. [6]

Alustariippumattomaan ohjelmistokehitykseen tarkoitettuja viitekehyksiä on useita. Yleisin alustariippumattomaan sovelluskehitykseen käytetty viitekehys oli vuonna 2021

Flutter ja toiseksi yleisin React Native [7]. Jetbrainsin 31 000 ohjelmistokehittäjän kyselytutkimuksen mukaan näillä kahdella viitekehyksellä on yhteensä merkittävä lähes 80 prosentin markkinaosuus. Tutkimuksen tulokset graafisessa muodossa kuvassa 1. Flutter on Googlen vuonna 2017 julkaisema viitekehys ja React Native Metan (ent. Facebook) vuonna 2015 julkaisema viitekehys. Molempien suosituimpien viitekehysten suosio on siis kasvanut hyvin nopeasti suhteessa niiden julkaisuvuoteen, ja ne ovat ohittaneet vanhemmat aiemmin hyvin suositut viitekehykset kuten Cordovan, julkaistu vuonna 2009 ja Xamarinin, julkaistu vuonna 2011.



Kuva 1: Ohjelmistokehittäjien käyttämät alustariippumattomat viitekehykset 2021. [7]

Alustariippumattomat viitekehykset mahdollistavat hybridisovellusten tuotannon. Ne tarjoavat työkaluja hybridisovellusten tuotantoon ja mahdollistavat koodipohjan viennin muotoon, jossa se voidaan laittaa mobiililaitteiden sovelluskauppoihin ladattavaksi ja toimii omana irrallisena sovelluksenaan. Viitekehysten toimintaperiaatteet vaihtelevat paljon. Vanhemmat viitekehykset kuten PhoneGap ja Xamarin toimivat wrappereinä, jotka vievät ohjelmistokoodin natiiviin ohjelmistokuoreen. Näin sovellus on ikään kuin verkkosivu, joka vain näytetään sovelluksen sisällä käyttäjän laitteella. Uudemmat viitekehykset toimivat eri tavoin, sillä ne pyrkivät parantamaan käyttäjäkokemusta olemalla lähempänä natiivia ohjelmistokoodia. Suosituin viitekehys Flutter toimii pienoisohjelmien (engl. widget) komponenttien kautta. Widgetit ovat elementtejä käyttäjän ruudulla, joita tarvittaessa päivitetään niiden tilan mukaan. Widgetit käännetään reaaliajassa ohjelmistokoodista natiiviksi koodiksi. Toiseksi suosituin

viitekehys React Native käyttää natiiveja komponentteja. Komponenttien tiloja muutetaan API-kutsuilla, jotka sillataan (engl. bridge) koodipohjasta natiiviksi koodiksi. [5]

Hybridisovelluksen kehitykseen käytettävää viitekehystä valittaessa on tärkeää miettiä projektin tavoitteet ja tarvittavat ominaisuudet viitekehykseltä. Kehysten toimintaperiaatteet vaihtelevat ja ne tarjoavat ne erilaisia ratkaisuja samojen toimintojen ja ominaisuuksien toteuttamiseen. Denkon ja Spelan tutkimuksessa [4] toteutettiin sama yksinkertainen testisovellus eri viitekehyksiä käyttäen. Tutkimus osoitti, että tehokkuusparametrit vaihtelevat paljolti eri viitekehysten välillä. Esimerkiksi sovelluksen käyttämä prosessoriaika (engl. CPU usage) oli Ionic viitekehyksellä tuotetulla sovelluksella yli puolet vähemmän verrattuna React Nativella tuotettuun sovellukseen. Toisaalta React Nativella tuotetun sovelluksen käynnistymisaika oli noin puolet vähemmän kuin Ionic viitekehyksellä toteutetun sovelluksen. Tämän vuoksi ohjelmistokehittäjän on tärkeää valita ohjelmistoprojektin mukainen viitekehys, joka soveltuu parhaiten ohjelmistoprojektin tavoitteisiin.

3. HYBRIDISOVELLUSTEN OHJELMISTOKEHITYKSEN RAKENTEET

Hybridisovelluksia voidaan tuottaa hyvin monella tavalla ja eri teknologioilla. Tämän takia ohjelmistoprojektia suunniteltaessa on syytä määritellä projektin tavoitteet, laajuus ja rajoitteet, joiden perusteella valitaan projektiin sopivat teknologiat ja mallit. Hybridisovellusmalli sopii projekteihin, joiden päämääränä on kustannustehokkuus, nopea tuotantovauhti ja yksinkertainen rakenne.

3.1 Ohjelmistoprojektin kehityskaari, SDLC

Mobiilisovelluksen tuottamiseen voi olla useita syitä. Sovellus voi esimerkiksi mahdollistaa kanssakäymisen asiakkaiden kanssa, lisätä bränditietoisuutta tai olla yksi liiketoiminnan tulonlähteistä [6]. Koska motiiveja sovelluksen tekoon voi olla useita on erityisen tärkeää miettiä mikä on ohjelmistoprojektin päämäärä ja miten siihen päästään. Hybridisovellusten tuottaminen tarjoaa monia vapauksia ohjelmistoprojektiin, mutta myös lisää joitakin rajoitteita lopputuotteeseen, siksi on tärkeää määrittää, sopiiko hybridiviitekehys toteutettavalle projektille.

Ohjelmistoprojektin kehityskaari eli SDLC (engl. Software Development Life Cycle) koostuu neljästä eri pääkomponentista: feasibility, analysis, design ja coding. Feasibility vaiheessa määritellään, onko projektin toteuttaminen kannattavaa. Siinä määritellään projektin tuoma arvo, arvioitu budjetti ja aikataulu. Analysis vaiheessa tarkastellaan tarkemmin projektin rakennetta ja sen eri vaatimuksia. Tavoitteena on tuottaa vaatimusdokumentaatio (engl. Requirements document), josta ohjelmistokehittäjät voivat tarvittaessa tarkistaa kaikki projektin perustiedot. Design vaiheessa tehdään konkreettiset päätökset projektin eri osista kuten mitä ohjelmointikieltä ja millaista tietokantaa käytetään. Tässä vaiheessa suunnitellaan myös sovelluksen ulkoasu. Design vaihe on usein iteratiivisin osa ohjelmistoprojektia, sillä siinä tehdään paljon valintoja, jotka vaikuttavat itse sovelluksen käyttäjäkokemukseen. Viimeinen komponentti ohjelmistoprojektia on itse koodaus. Siinä luodaan sovellus käyttäen valittua ohjelmointikieltä ja viitekehyksiä. [8]

Kun ohjelmistoprojektin tavoitteena on tuottaa sovellus molemmille suurille mobiilialustoille, hybridisovellusten tuotanto tehostaa SDLC:n vaiheita monella tapaa verrattuna natiivin sovelluksen tuottamiseen. Koska hybridisovellukset perustuvat yhteen koodipohjaan ja sen kautta luodaan sovellus kaikille alustoille, moni vaihe

yksinkertaistuu. Feasibility vaiheessa arvioitavat kokonaiskustannukset sovellukselle pienenevät ja hybridisovellusten tuottaminen on usein nopeampaa kuin alustakohtaisen sovelluksen. Analysis vaiheessa ei tarvitse huomioida usean alustan koodipohjan vaatimuksia, niiden eroavaisuuksia ja yhteensopivuutta. Design vaiheessa erilaisia valintoja ohjelmiston rakenteen suhteen on vähemmän, koska valinnat on tehtävä vain yhden koodipohjan suhteen. Koodausvaiheessa hybridisovelluksen yksi koodipohja mahdollistaa yksinkertaisemman kokonaisrakenteen verrattuna useaan alustakohtaiseen koodipohjaan. Toisaalta hybridisovelluksen rakenne lisää myös osaltaan haasteita ohjelmiston kehityskaareen. Analysis vaiheessa selvitettävät riippuvuudet saattavat olla haastavimmin toteutettavissa hybridisovellukseen, koska mahdollisesti tarvittavat ominaisuudet sovellukselta voivat vaatia kolmannen osapuolten tuottamia lisäosia. Kolmannen osapuolen tuottamat lisäosat lisäävät riippuvuuksia, koska niiden toiminta ja ylläpito ei ole enää täysin ohjelmistokehittäjän käsissä. Design vaihe myös hankaloituu, sillä hybridisovellukset eivät mukaudu eri laitteisiin yhtä hyvin kuin natiivit sovellukset. Erityisesti käyttöliittymän suunnitellussa on huomioitava hybridiviitekehyksen rajoitteet ja tehtävä valintoja, jotka parhaiten sopivat usealle alustalle toimivaksi. Koodausvaiheessa haasteita tuo riippuvuus lisäosiin tiettyjen toimintojen tuottamiseksi. Lähtökohtaisesti mitä itsenäisemmin ohjelma toimii sen paremmin. [8]

SDLC eri osat vaihtelevat paljon projektin mukaan. On siis tärkeää valita projektille oikeat työkalut ja teknologiat, jotta lopputulos olisi mahdollisimman optimoitu ja käyttäjäystävällinen. Hybridisovelluksen tuottaminen tarjoaa monia etuja natiiviin verrattuna, mutta tuo mukanaan myös epäedullisia ominaisuuksia, joten on tärkeä valita projektin mukaan, onko natiivi vai hybridisovellus kokonaisuudessaan parempi vaihtoehto.

3.2 Tekniikat ja teknologiat

Natiivit sovellukset ovat kehitetty käyttäen alustakohtaisia ohjelmistokehityspaketteja (engl. Software Development Kit, SDK) ja ohjelmointikieliä. iOS:llä alustakohtainen ohjelmointikieliä ovat Swift sekä Objective-C ja kehitysympäristönä (engl. Integrated Development Environment, IDE) on tyypillisesti Xcode. Androidilla taas kielinä ovat Kotlin sekä Java ja ympäristönä Android Studio. Natiivin koodin jakaminen toiselle alustalle ei kuitenkaan suoraan onnistu vaan väliin vaaditaan tulkitsija (engl. interperator) tai kääntäjä, jotka muuntavat natiivin ohjelmistokoodin toiselle natiiville kielelle. Tällaiset tulkitsijat eivät kuitenkaan ole kovin yleisiä, sillä niiden toiminta on saattaa olla

epävakaata ja epätarkoituksen omaista. Yleisesti käytetympi tapa hyödyntää jaettua koodipohjaa on tuottaa hybridisovellus [9].

Hybridisovellusten tuotannossa ohjelmistokehittäjät tuottavat yhden koodipohjan, joka viitekehyksen avulla viedään eri alustoille. Tällä tavoin samalla koodipohjalla saadaan samanaikaisesti sovellus molemmille suurille mobiilikäyttöjärjestelmille, Androidille ja iOS:lle. Tämä on yksi pääsyistä, miksi hybridisovelluksia tehdään. Koska useat alustat jakavat saman koodipohjan kehitysaika pienenee ja ohjelmiston ylläpito sekä päivittäminen yksinkertaistuvat merkittävästi. Jaettu koodipohja tuo myös vapauksia eri tekniikoiden ja teknologioiden käyttöön. Usein hybridisovellusten koodipohjat tuotetaan verkkoteknologioita kuten JavaScriptiä ja HTML:ää käyttäen, mutta myös muita ohjelmointikieliä kuten C#:ia ja Dartia on mahdollista käyttää. Myös IDE:n valinta on helpompaa sillä hybridisovelluksia tehdessä ei tarvitse välttämättä käyttää alustakohtaisia kehitysympäristöjä. Alustakohtaisten kehitysympäristöjen käyttö on lähes pakollista natiivien sovellusten kohdalla, koska laitevalmistajien tuki hyvin vahvasti sidottu näille IDE:lle. Hybridisovellusten kehitys antaa siis ohjelmistokehittäjille monia vapauksia eri teknologioiden ja tekniikoiden hyödyntämiseen, toisin kuin natiivien sovellusten kehitys, joka on taas suora viivaisempaa, koska mahdollisuuksia ohjelmointikielien ja IDE:en suhteen on vähemmän. [5]

4. HYBRIDIKEHYSRAKENTEEN HAASTEET JA HYÖDYT

Hybridikehysrakenne tarjoaa monimuotoisemman tavat tuottaa sovelluksia natiiviin kehysrakenteeseen verrattuna. Sen suurimmat hyödyt ovat sen pohjautuminen yhteen koodipohjaan näin nopeuttaen ja tehostaen kehitysprosessia. Natiivimalli taas tarjoaa paremmat mahdollisuudet käyttäjäkokemuksen luomiseen ja optimoidumpaan lopputulokseen.

4.1 Nopeus ja tehokkuus

Mobiililaitteiden tehokkuus on vuosi vuodelta kasvanut merkittävästi, mutta laitteiden pieni koko rajoittaa niihin mahtuvaa teknologiaa. Tämän takia on tärkeää optimoida mobiilisovellusta mahdollisimman paljon, jotta se olisi tehokas ja näin miellyttävämpi käyttää. Sovelluksen käyttönopeus on yksi suurimmista tekijöistä hyvän käyttäjäkokemuksen luomiseksi. Tämän vuoksi sovelluksen nopeuden ja tehokkuuden optimointi on yksi tärkeimmistä osa-alueista sovelluskehityksessä.

Hybridisovellusmalli kärsii tehokkuuden suhteen, sillä ne eivät ole täysin optimoitu käytetylle laitteelle. Ajayi et al. tutkimuksessa [10] tuotettiin sama yksinkertainen testisovellus natiivi- ja hybridiversiona, joiden tehokkuusparametrejä vertailtiin. Kaikissa testeissä havaittiin, että natiivisovellus oli nopeampi ja tehokkaampi, mutta erojen suuruus vaihteli paljon testin mukaan. Erityisesti sovelluksen käynnistymisajassa havaitaan merkittäviä eroja. Hybridiversion käynnistyminen kesti noin kaksikymmentä kertaa kauemmin kuin natiivin sovelluksen käynnistyminen. Integraalilaskenta ja Merge sort algoritmi testeissä taas ero sovellusversioiden välillä oli merkittävästi pienempi.

Hybridisovellukset eivät pärjää tehokkuus ja nopeus testeissä natiiveille sovelluksille. Natiivit sovellukset pystyvät hyödyntämään laitteen ominaisuuksia paremmin, koska ne ovat suoraan yhteydessä laitteen toimintaan. Tämän vuoksi mahdollisimman suurta nopeutta vaativat sovellukset tulisi kehittää natiiveina, mutta mikäli nopeus ja tehokkuus eivät ole pääparametri sovellukselle, hybridimalli tarjoaa kevyille sovelluksille varteen otettavan vaihtoehdon.

4.2 Käyttäjäkokemus

Sujuva käyttäjäkokemus (engl. User Experience, UX) on yksi tärkeimmistä parametreistä onnistuneelle sovellukselle. Mikäli käyttäjät kokevat sovelluksen huonoksi käyttää he

nopeasti tympääntyvät ja eivät jatka sen käyttöä. Käyttäjäkokemuksen merkitys on siis suuri ja siihen tulisi panostaa merkittävästi ohjelmistotuotannossa.

Malavolta et al. tutkimuksessa [11] käytiin läpi lähes 12 tuhatta Androidin sovelluskaupan Google Play Storen sovellusta ja kolme miljoonaa niiden arvostelua. Tutkimuksessa selvitettiin loppukäyttäjien kokemuksia hybridisovelluksista. Havainnoitiin, että loppukäyttäjät arvostavat natiiveja ja hybridisovelluksia yhtä paljon, mutta sovelluksen käyttötarkoituksella on merkitystä. Hybridisovellusmalli soveltui hyvin yksinkertaisille sovelluksille, kun taas natiivimalli soveltui erityisesti paremmin laitteen ominaisuuksia hyödyntäviin sovelluksiin. Käyttäjät myös havainnoivat enemmän ohjelmistovirheitä (engl. bugs) hybridisovelluksissa ja kokivat natiivit sovellukset tehokkaammiksi. Havainnot tukevat yleistä käsitystä hybridisovellusten toiminnasta suhteessa natiiveihin sovelluksiin. Hybridisovellukset eivät pysty olemaan yhtä käyttäjäystävällisiä ja nopeita natiiveihin sovelluksiin verrattuna niiden rakenteen vuoksi.

Hybridisovellusten käyttäjäkokemushaasteet, johtuvat pääosin hybridisovellusten rakenteesta, joka pohjautuu jokaisella alustalla samaan koodipohjaan. Se ei ole natiivin sovelluksen tapaan kytköksissä juuri käytettävälle laitteelle taikka alustalle, vaan sovellus toimii kaikilla laitteilla samalla logiikalla. Tällöin sovelluksen ulkoasu ja nopeus kärsivät. Uudemmat hybridiviitekehykset kuten React Native ja Flutter pyrkivät korjaamaan tätä ongelmaa, käyttämällä natiiveja komponentteja ja widgettejä. Tällöin sovelluksen ulkoasu päivitetään ja luodaan itse laitteella eikä sitä haeta verkkosovelluksen tapaan palvelimelta. Tämä tehostaa sovellusten toimintaa ja responsiivisuutta merkittävästi. [5]

4.3 Kustannukset

Ohjelmistokehityksessä kustannukset ovat hyvin tärkeä ja suuri osa kokonaisuutta. Kehityksen kulut tulevat pääosin seitsemästä osasta: julkaisu, testaus, applikaation hallinnointi, infrastruktuuri, toiminnot, design ja suunnittelu [12]. Natiivin sovelluksen kehityksessä näistä osista design ja suunnittelu voidaan pääosin tehdä alustariippumattomasti, mutta muut osiot joudutaan usein toistamaan kaikille alustoille erikseen. Hybridisovellusten kehityksessä pystytään monissa osioissa tuottamaan alustariippumattomia ratkaisuja.

Ohjelmistoprojektien kokonaiskustannukset vaihtelivat merkittävästi riippuen projektista. Suurin kustannuserä on itse sovelluksen kehitys ja sen suunnittelu. Kehityskustannukset kostuvat pääosin ohjelmistokehittäjien palkoista ja mahdollisista lisensseistä.

Suunnittelukustannukset koostuvat taas ulkoasun, käytettävyyden ja ohjelmiston logiikan suunnittelusta. [13]

Hybridisovellusta kehittäessä ei tarvitse palkata kalliita alustakohtaisia natiiveja ohjelmointikieliä osaavia ohjelmistokehittäjä, koska hybridisovellukset pohjautuvat usein yleisiin verkkoteknologioihin. Verkkoteknologia osaajia riittää enemmän kuin alustakohtaisten kielien osaajia, joten heidän palkkaamisensa on lähtökohtaisesti edullisempaa. Hybridisovellusten käyttämä pääohjelmointikieli JavaScript on myös merkittävästi suositumpi kuin iOS:n Swift tai Androidin Kotlin alustakohtaiset ohjelmointikielet [14]. Tästä johtuen työvoimaa, jotka osaavat tuottaa verkkosovellusten teknologioilla sovelluksia löytyy merkittävästi enemmän sekä edullisemmin. Näin suuri kustannuserä eli ohjelmistokehittäjien palkat pienenevät merkittävästi. Myös käytettyjen työtuntien määrä vähenee, koska samanaikaisesti voidaan yhdellä koodipohjalla tuottaa sovellus molemmille alustoille. Työtuntien ja ohjelmistokehittäjien palkkojen vähentyessä sovelluksen kokonaiskustannukset laskevat. [15]

Total Cost of Ownership (TCO) kuvaa tuotteen tai työkalun kokonaiskustannuksia sen koko elinkaaren aikana. Se sisältää suorat ja epäsuorat, sekä mahdolliset aineettomat kustannukset, joilla on rahallinen arvo. Mobiilisovelluskehityksessä TCO:n määrittämiseksi pitää selvittää projektin skaala, laajuus, monimutkaisuus tekijät ja kustannus komponentit [16]. Hybridisovellusmalli auttaa erityisesti vähentämään kustannuskomponentteja, jotka koostuvat kolmesta osasta: tuki-, ylläpito- ja uponneet kustannukset. Koska hybridimallissa luodaan vain yksi koodipohja, laskee tuen ja ylläpidon kustannukset merkittävästi. Samoin myös uponneet kustannukset projektin aikana vähenevät, koska ohjelmistotiimi voi keskittyä vain yhden koodipohjan luomiseen usean eri alustalle tulevan sijaan. Myös mahdolliset muutokset sovelluksen toimintaan tehdään vain yhteen koodipohjaan verrattaessa natiiveihin sovelluksiin, joissa muutos tarvitsee erikseen tehdä jokaiselle koodipohjalle erikseen. Näin kehitys- ja ylläpitoaika sekä kehityksen aloituksesta sovelluksen julkaisuun mennyt aika (engl. Time to Market) vähenevät. Time to Market ja TCO ovat erittäin tärkeitä osia ohjelmistoprojektissa. Näiden hallintaan hybridisovellusmalli soveltuu hyvin, sillä hybridisovellusten toteuttaminen on nopeampaa ja sovelluksen koko elinajan kustannukset pienempiä suhteessa eri alustoille tehtäviin natiiveihin sovelluksiin. [17] [8]

4.4 Tietoturva

Tietoturva on hyvin tärkeä osa sovelluksen toimintaa ja merkittävä vaatimus sekä käyttäjiltä, että sovelluskaupoilta. Hybridisovelluskehitys vaatii usein kolmannen osapuolen lisäosia (engl. plugin) laitteen eri toimintojen toteuttamiseen. Nämä

ulkopuolisten toimijoiden tuottamat lisäosat lisäävät tietoturvariskiä, sillä ohjelmistokehittäjä ei välttämättä tiedosta lisäosan koko toimintaa ja lisäosan mahdollisten tietoturva aukkojen päivittäminen saatetaan usein sivuuttaa. Mahdolliset pahantahtoiset lisäosat saattavat näin päästä käsiksi käyttäjän laitteeseen hybridisovelluksen kautta. Hybridisovelluksen ohjelmistokehittäjä saattaa siis jopa tiedostamattaan altistaa sovelluksen käyttäjät tietovuodoille. [3]

Usein hybridisovelluksen ohjelmointikielenä toimii JavaScript, joka lisää riskiä erityisesti *code injection cross-site scripting* hyökkäyksille. Tätä voidaan kuitenkin välttää käyttämällä sirpalointiohjelmaa (engl. obfuscator software), joka sekoittaa ohjelmistokoodia. Näin mahdollisen pahantahtoisen tietoturva-aukkoja etsivän ohjelmistokehittäjän on vaikeampi tulkita ja väärinkäyttää ohjelmistokoodia. Lisäksi sirpalointiohjelma tehostaa ohjelmistokoodin toimintaa, koska se minimoi ohjelmakoodin ja poistaa käyttämättömät ohjelmistopaketit. [18]

4.5 Ohjelmistokehittäjän näkökulma

Hybridisovellukset pohjautuvat usein verkkoteknologioihin, jotka ovat monelle ohjelmistokehittäjille tunnettuja ja dokumentaatiota sekä foorumikeskusteluita niiden osalta löytyy paljon. StackOverFlown 2022 kehittäjäkyselyn mukaan noin 12,6 % vastanneista käyttää tai aikoo käyttää hybridisovellusten tuotannossa hyödynnyttäviä viitekehyksiä React Nativea ja Flutteria vuoden aikana. Myös noin 5,2 % vastanneista on käyttänyt tai aikoo käyttää hybridiviitekehyksiä Ionicia ja Xamarinia. Kun verrataan hybridisovellusten pääohjelmointikieltä JavaScriptiä ja iOS:lle natiivia kieltä Swiftiä on JavaScriptin suosio yli 11-kertainen. Näin ohjelmistokehittäjän on helpompi etsiä dokumentaatiota ja tukea ongelmaan JavaScriptin osalta verratuin Swiftiin. Alustakohtaiset kielet kuten Kotlin ja Swift ovat suhteessa melko uusia kieliä, joten niiden ympärille ei ole vielä ehtinyt kehittyä yhtä suurta kehittäjäverkostoa kuin JavaScriptin, joka on peräisin jo 1990-luvulta ja on kehittynyt siitä saakka. Nämä syyt johtavat siihen, että natiiveja ohjelmistokehittäjiä on vähemmän. [14]

Biørn-Hansen et al. [9] ja Ahmad et al. [19] tutkimuksissa selvitettiin ohjelmistokehittäjien kokemia ongelmia mobiilisovelluskehityksessä. Merkittäviksi ongelmiksi hybridisovelluskehityksessä koettiin hybridisovellusten suorituskyvyn heikkous verrattuna natiiveihin sovelluksiin, epäoptimaalinen käyttäjäkokemuksen luominen ja hybridisovellusten testauksen hankaluus. Tutkimuksissa havaittiin myös, että hybridisovellusten usealle alustalle jaetun koodipohjan ansiosta, koodin uudelleen käyttö ja muutokset sovelluksen toimintaan koettiin merkittävästi harvemmin ongelmiksi hybridisovellusten kohdalla natiiveihin verrattuna.

5. YHTEENVETO

Hybridisovellukset ovat hyvä ratkaisu, kun tavoitteena on kevyt, nopeasti tuotettu, kustannustehokkaasti tuotettu ja usealle alustalle tarkoitettu sovellus. Ne soveltuvat käyttötarkoituksiin, joissa laitteelta vaadittu teho on pientä ja sovelluksen rakenne ei vaadi vahvaa integraatiota laitteen ominaisuuksien kanssa. Alustakohtaiset natiivit sovellukset ovat kuitenkin johtavassa asemassa tehokkuuden, käyttäjäkokemuksen ja laitteen ominaisuuksien hyödyntämisen kannalta. Hybridisovellusteknologiat kehittyvät kuitenkin nopeaa tahtia ja saavuttavat natiivien sovellusten tunnuslukuja ja ominaisuuksia. On kuitenkin lähes mahdotonta päästä tasolle, jossa käännetty yleiskielinen hybridisovelluksen ohjelmistokoodi toimisi samalla tehokkuudella kuin alustalle optimoitu natiiviohjelmistokoodi. Uudemmat hybridisovelluskehykset kuitenkin pyrkivät eri tekniikoita käyttäen toimimaan natiivin sovelluksen tavoin ja näin ovat parantaneet hybridisovellusten tehokkuus lukemia.

Aina tehokkuuden ja käyttöliittymän optimointi ei ole välttämättä tärkein asia ohjelmistoprojektissa, vaan hybridisovellusmallin edut kuten nopea kehitys ja jaetun koodipohjan tarjoamat edut saattavat osoittautua paremmaksi vaihtoehdoksi. Ohjelmistoprojektin suunnittelu vaiheessa on siis erityisen tärkeää määrittää projektin vaatimukset, joiden perusteella voidaan tehdä päätös, tuotetaanko sovellus natiivi vai hybridimuotoisena.

6. LÄHTEET

[1] C. Griffith. What is Hybrid Mobile App Development: Hybrid vs Native vs Web. Luettu: 30.6.2022. Saatavilla: https://ionic.io/resources/articles/what-is-hybrid-app-development

[2] GlobalStats. Mobile Operating System Market Share Worldwide. 2022. Luettu: 1.7.2022. Saatavilla: https://gs.statcounter.com/os-market-share/mobile/worldwide

[3] K. Shah, H. Sinha, P. Mishra. Hybrid Application Development and Implementation. 2019 6th International Conference on Computing for Sustainable Global Development (I2CT), Mumbai, Intia, 2019.

[4] B. Denko ja P. Spela. A Comprehensive Comparison of Hybrid Mobile Application Development Frameworks. International Journal of Security and Privacy in Pervasive Computing. 2021.

[5] K. Shah, H. Sinha, P. Mishra. Analysis of Cross-Platform Mobile App Development Tools. 2019 IEEE 5th International Conference for Convergence in Technology (I2CT), Mumbai, Intia, 2019.

[6] J. C. McWherter ja S. Gowell. Professional mobile application development. Indianapolis: Wiley. 2012.

[7] Statista. Cross-platform mobile frameworks used by software developers worldwide from 2019 to 2021. Kesäkuu 2021. Luettu: 1.7.2022. Saatavilla: https://www.statista.com/statistics/869224/worldwide-software-developer-workinghours/

[8] A. M. Langer. Guide to Software Development. New York, Springer London. 2012.

[9] A. Biørn-Hansen, T.-M. Grønli, G. Gheorghita, A. Sahel, ”An Empirical Study of Cross-Platform Mobile Development in Industry. Wireless Communications and Mobile Computing 2019.

[10] O. O. Ajayi, Adebola Okunola Orogun, Ayokunle Abiodun Omotayo, Taiwo Gabriel Omomule, Segun Michael Orimoloye. Performance Evaluation of Native and Hybrid Android Applications. Foundation of Computer Science FCS, New York. 2018.

[11] I. Malavolta, S. Ruberto, T. Soru, V. Terragni. End Users' Perception of Hybrid Mobile Apps in the Google Play Store. IEEE International Conference on Mobile Services. 2015.

[12] N. Craigmile. Cost to Build a Mobile App: A Survey. 30.1.2015. Luettu: 20.7.2022 Saatavilla: https://clutch.co/app-developers/resources/cost-build-mobile-app-survey-2015

[13] C. Radu. Average mobile app development cost breakdown in 2022. Toukokuu 2022. Luettu: 13.7.2022.Saatavilla: https://blog.mobiversal.com/app-development-costbreakdown.html

[14] StackOverFlow. 2022 Developer Survey. Toukokuu 2022. Luettu 20.7.2022. Saatavilla: https://survey.stackoverflow.co/2022/

[15] M. Kremer. The ROI of Native vs. Hybrid App Development. Ionic, 21.3.2020. Luettu: 20.7.2022 Saatavilla: https://ionic.io/resources/articles/roi-hybrid-vs-native

[16] L. Ayers. How to Estimate TCO of Mobile Apps. Luettu 1.7.2022. Saatavilla: https://www.sapbwconsulting.com/blog/how-to-estimate-tco-of-mobile-apps

[17] C. d. l. Torre ja S. Calvert. Microsoft Platform and Tools for Mobile Application Development. Microsoft. 2016.

[18] M. L. Hale ja S. Hanson. A Testbed and Process for Analyzing Attack Vectors and Vulnerabilities in Hybrid Mobile Apps. IEEE World Congress on Services. 2015.

[19] A. Ahmad, K. Li, C. Feng, S. M. Asim, A. Yousif, S. Ge. An Empirical Study of Investigating Mobile Applications Development Challenges. IEEE Access. 2018.