Monialustainen ohjelmistokehitys – teknologiat ja käyttäjäkokemus

Turun yliopisto
Tietotekniikan laitos
LuK-tutkielma
Tietojenkäsittelytiede
Toukokuu 2025
Onni Lassy

TURUN YLIOPISTO
Tietotekniikan laitos

Onni Lassy: Monialustainen ohjelmistokehitys – teknologiat ja käyttäjäkokemus

LuK-tutkielma, 28 s.
Tietojenkäsittelytiede
Toukokuu 2025

Monialustainen ohjelmistokehitys pyrkii tuottamaan usealla laitteella toimivia sovelluksia ja tarjoamaan yhdenmukaisen käyttäjäkokemuksen eri alustojen välillä. Tässä kandidaatin tutkielmassa aihetta tarkasteltiin kirjallisuuskatsauksena monialustaisesta tutkimuskentästä. Havaittiin, että suurin osa akateemisesta kirjallisuudesta liittyy mobiilialustoihin, käyttäjäkokemukseen ja monialustakehyksiin. Aineiston pohjalta tarkasteltiin tapoja monialustaisen käyttäjäkokemuksen saavuttamiseksi teknologisilla ratkaisuilla ja suunnitteluperiaatteilla.
Tutkielmassa ei esitetä parhaita käytäntöjä monialustaiseen kehitykseen, vaan jäsennetään käyttäjäkokemukseen keskeisesti vaikuttavia tekijöitä, kuten käyttöliittymät, suunnitteluperiaatteet, kehitysratkaisut ja testaus. Lopputuloksena muodostuu kattava kokonaiskuva monialustaisesta terminologiasta ja teknologioiden merkityksestä käyttäjäkokemukseen.
Monialustaisuus on merkittävä osa ohjelmistokehityksen tutkimuskenttää. Tutkimusala hyötyisi kuitenkin nykyistä laajemmasta teknisestä tarkastelusta ja laitteistojen kasvavan moninaisuuden huomioimisesta. Tämä voisi parantaa akateemisen tutkimuksen käytännön hyötyjä monialustaisessa ohjelmistokehityksessä.

Asiasanat: monialustaisuus, ohjelmistokehitys, käyttäjäkokemus, käyttöliittymät, kehityskehykset

Sisällys

Kuvat

Taulukot

Termistö

CI/CD continuous integration and continuous deployment

CPU Central Processing Unit

CPUX Cross-Platform User Experience

IDE Integrated Development Environment

JSON JavaScript Object Notation

PWA Progressive Web Applicationl

REST Representational State Transfer

SOAP Simple Object Access Protocol

XML Extensible Markup Language

1 Johdanto

Viimeisten vuosikymmenien aikana vuorovaikutus tietokoneiden kanssa on muuttunut merkittävästi. Yhden näyttöpäätteen käytöstä on siirrytty useampien näyttöjen työpisteisiin, mobiililaitteisiin ja muihin graafisia käyttöliittymiä sisältäviin laitteisiin, kuten älykellot ja älykodinkoneet [1][2]. Nykypäivänä loppukäyttäjät ovat päivittäin tekemisissä useiden tietoteknisten laitteiden ja näiden sisältämien ohjelmistojen kanssa. Tämän vuoksi ohjelmistoyhtiöt pyrkivät tuomaan ohjelmiaan tarjolle monialustaisesti, mikä tarkoittaa ohjelman saatavuutta kahdella tai useammalla laitteella [2][3]. Tyypillisiä laajasti monialustaisia ohjelmia on musiikin ja median toistamiseen, sosiaalisen mediaan ja kommunikointiin liittyvät sovellukset, verkkoselaimet sekä pilvitallennuspalvelut.

Käyttöliittymä on järjestelmän ja käyttäjän välinen rajapinta, jossa vuorovaikutus tapahtuu käyttäjän antamien syötteiden ja järjestelmän antamien palautteiden perusteella. Tietokoneiden kohdalla puhutaan graafisista käyttöliittymistä (engl. *graphical user interface*, GUI) näyttöpäätteellä, jota ohjataan tyypillisesti hiiren ja näppäimistön avulla. Yksinkertaisemmissa tapauksissa käytetään tekstipohjaista komentoliittymää (engl. *command line interface*, CLI), johon kirjoittamalla ennaltamäärättyjä tekstikomentoja ohjataan tietokoneen toimintaa. Graafinen käyttöliittymä on komentoliittymää teknisesti monimutkaisempi, mutta käytettävyydeltään yksinkertaisempi, sillä se sisältää vuorovaikutteisia elementtejä kuten painikkeita ja

valikoita. Tämä tekee tietoteknisen laitteen käytöstä helppoa myös henkilöille ilman teknistä osaamista tai erityistä tekstikomentojen opettelua. [4]

Erilaisten tietoteknisten laitteiden yleistyminen on johtanut käyttäjiä odottamaan ohjelmiltaan samoja toimintoja ja yhtenäistä käyttökokemusta alustasta riippumatta [5][6]. Ohjelmistokehittäjät pyrkivät vastaamaan näihin odotuksiin useilla erilaisilla kehitysratkaisuilla, joilla ohjelmat saadaan tuotua useille alustoille. Käyttäjän kannalta oleellista on ohjelman hyvä käytettävyys, johon vaikuttaa samankaltainen ulkonäkö ja toiminta eri laitteiden välillä eli yhdenmukainen käyttäjäkokemus. Kehittäjien tehtävä on etsiä sopivat tekniset ja vuorovaikutukselliset ratkaisut monialustaisuuden ja yhdenmukaisen käyttäjäkokemuksen saavuttamiseksi. Tässä tutkielmassa perehdytään monialustaiseen ohjelmistokehitykseen, jonka tavoitteena on yhtenäinen käyttäjäkokemus laitteiden välillä. Tarkemmat tutkimuskysymykset, joihin tutkielmassa haetaan vastausta, ovat:

1. Mitä erityispiirteitä ja teknisiä ratkaisuja liittyy monialustaiseen ohjelmistokehitykseen?

2. Miten käyttäjäkokemus voidaan yhtenäistää monialustaisissa ohjelmistoissa?

Tutkielma toteutettiin kirjallisuuskatsauksena perehtymällä aiheeseen liittyviin tieteellisiin julkaisuihin. Lähteitä haettiin tietokannoista kuten ACM Digital Library, IEEE Xplore ja Google Scholar. Hakulausekkeina käytettiin eri muotoja monialustaisuudesta, kuten *cross-platform* ja *multi-device*, yhdistettynä termeihin kuten *User Experience, UX, challenges, user interface, approaches* ja *design patterns.* Täydentäviä lähteitä on etsitty myös muista tietokannoista, kuten Turun Yliopiston Volterista sekä Googlen hakukoneella.

Hakutulokset olivat moninaisia ja suurin osa tuloksista ei liittynyt tutkielman aiheeseen. Esimerkiksi hakualusekkeella *(cross-platform OR cross-device OR multi-platform OR multi-device) AND ((software development) OR (user interface)) AND*

((user experience OR UX) OR challenges) löytyy ACM tietokannasta 133 846 tulosta ja IEEE:stä 387 tulosta. Tuloksia rajattiin julkaisuvuoden perusteella, pyrkien käyttämään vuoden 2020 jälkeen tai vähintään vuoden 2014 jälkeen julkaistuja artikkeleita. Otsikon perusteella valittiin hakutuloksia lähempään tarkasteluun ja tiivistelmän perusteella osa tuloksista luettiin kokonaan.

Tutkielma keskittyy monialustaisuuteen ohjelmiston jakelumuotona ja teknologiana. Luvussa 2 käsitellään monialustaisuuden terminologiaa, sovellusarkkitehtuuria sekä monialustaista käyttäjäkokemusta. Luvussa 3 käsitellään monialustaisten ohjelmistojen kehitystä käyttäjäkokemuksen ja teknisten ratkaisujen näkökulmasta. Luku 4 sisältää kirjoittajan omaa pohdintaa ja ajatuksia monialustaisesta kehityksestä, tutkimuksesta ja tutkielman tekemisestä. Luvussa 5 tehdään yhteenveto keskeisistä havainnoista ja vastataan tutkimuskysymyksiin. Tutkielman tulokset voivat auttaa kehittäjiä hahmottamaan paremmin käyttäjäkokemuksen erityispiirteitä sekä erilaisia tapoja kehittää monialustaisia ohjelmistoja. Tutkielma myös selkeyttää monialustaisuuteen liittyvää terminologiaa, kehityksen haasteita ja näiden ratkaisuja.

2 Monialustaisuus ja käyttäjäkokemus

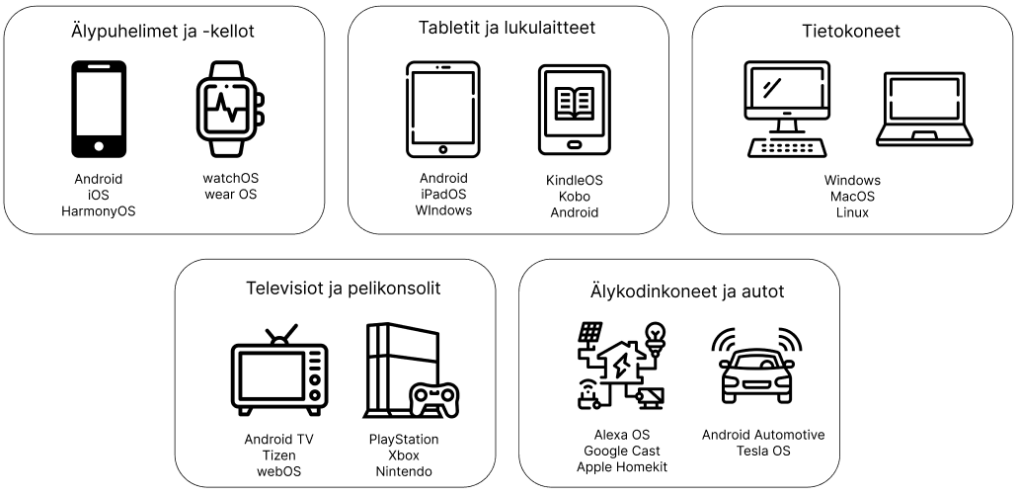
Monialustaisuus on yhä tärkeämpää ohjelmistokehityksessä, sillä kasvavan alustamäärän takia tarvetta yhdellä alustalla toimiville ohjelmille on yhä vähemmän [7]. Käyttäjät odottavat ohjelmiensa toimivan ongelmitta useilla laitteilla, joka asettaa merkittäviä haasteita ohjelmistojen tekniselle toteutukselle ja käyttöliittymän suunnittelulle. [1][8] Tässä luvussa tarkastellaan monialustaisuuden käsitettä, usealla laitteella toimivan sovelluksen arkkitehtuuria ja käyttäjäkokemusta.

2.1 Monialustaisuus ohjelmistokehityksessä

Ohjelmaa, joka toimii kahdella tai useammalla alustalla, sanotaan monialustaiseksi. [1] Monialustaisuuden tutkimuskenttä on pirstaloitunut, eikä alalla ole yhtenäistä termistöä. Termejä kuten *cross-device*, *cross-platform*, *multi-device*, *multi-channel*, *multi-screen*, *multiple user interfaces* tai *distributed* käytetään eri yhteyksissä, ja niiden merkitykset voivat vaihdella. [1][9]

Tässä tutkielmassa alustalla tarkoitetaan laitteen ja ohjelman oleellisinta suoritusympäristöä, kuten käyttöjärjestelmää (esim. Android ja iOS mobiililaitteilla, Windows ja macOS tietokoneilla) tai verkkoselainta. Alustoja kutsutaan myös termeillä kuten 'mobiilialustat' puhuttaessa kaikista mobiililaitteista ja 'työpöytä' pu-

huttaessa tietokonelaitteista. Alusta on kuitenkin eri asia kuin fyysinen laite, vaikka ne menevät usein päällekäin. Laitteita ja alustoja kuvattu tarkemmin kuvassa 2.1.



Kuva 2.1: Laitteita ja alustoja

Monialustaisella ohjelmistolla tarkoitetaan saman ohjelman saatavuutta kahdella tai useammalla alustalla toteutustavoista riippumatta. Monialustaisuutta tukevaa ohjelmointiteknologiaa kutsutaan tässä tutkielmassa monialustaiseksi kehykseksi tai monialustakehykseksi, ja termeillä viitataan työkaluihin kuten Flutter, React Native ja QT, jotka mahdollistavat osittain saman koodipohjan eri alustoilla. Tällaisella kehyksellä kehitettyä ohjelmaa kutsutaan monialustasovellukseksi.

Hyvä esimerkki laajasti monialustaisesta ohjelmistosta on Spotify, joka on saatavilla verkkosovelluksena, työpöytäsovelluksena, iOS ja Android sovelluksina, pelikonsoleilla, älykelloissa, sekä sulautetuissa järjestelmissä kuten kaiuttimissa [10]. Käyttäjän data Spotifyn taustajärjestelmissä pysyy yhtenäisenä, jolloin eri alustoille toteutetut käyttöliittymät päivittyvät aina viimeisimpien muutosten mukaan, vaikka ne olisi tehty toisella laitteella. Käyttäjä voi siis vaivattomasti vaihtaa laitetta jolla sovellusta käyttää ja data pysyy yhtenäisenä. Spotifyn kohdalla on myös pyritty pitämään käyttöliittymät eri versioissa samankaltaisina oleellisilta toiminnallisuuksiltaan ja ulkonäöltään, mukautuen silti alustakohtaisiin käyttäjien tottumuksiin ja vuorovaikutustapoihin. [8][11]

Ristiinalustainen sovellus (engl. *crossmedia service*) tarkoittaa, että ohjelman toiminnallisuus on jaettu usean laitteen välille. Tästä esimerkki on Google Chromecast, joka mahdollistaa käyttäjää ohjaamaan esim. televisiota mobiililaitteen tai verkkoselaimen avulla. Tämänkaltaisessa sovelluksessa jokaista laitetta voidaan hyödyntää sen vahvuuksien pohjalta, mutta toiminnallisuudet ovat erilaisia. Oleellisin haaste ristiinalustaisissa sovelluksissa on määrittää miten toiminnallisuudet jaetaan eri laitteiden välille. [8]

2.2 Arkkitehtuuri

Verkkopalvelut mahdollistavat eri alustoilla toimivien sovellusten yhteistoiminnan syntaktisen ja semanttisen yhteentoimivuuden avulla. Syntaktinen yhteensopivuus tarkoittaa tiedonvaihtoa määritellyn protokollan kautta. Se ei takaa kuitenkaan tiedon yhtenäistä tulkintaa eri sovelluksissa vaan tähän tarvitaan semanttista yhteensopivuutta. Se tarkoittaa, että eri palvelut ymmärtävät vastaanottamansa tiedon tarkasti ja merkityksellisesti. Ratkaisuja verkkopalvelujen toteutukseen ovat erilaiset rajapintamallit ja tiedon esitystavat. Näitä ovat esimerkiksi SOAP-protokolla, REST-arkkitehtuuri sekä tiedon esitysmuodot JSON ja XML. [9]

Verkkopalvelujen avulla voidaan rakentaa ohjelmistolle toimiva arkkitehtuuri, jonka suunnittelussa korostuu kerroksellisuus, skaalautuvuus ja uudelleenkäytettävyys. Tässä keskeinen osa on tietovarasto ja taustajärjestelmä, joka kykenee palvelemaan useita käyttöliittymiä eri alustoilla. Tavoitteena on, että ohjelman tila pysyy yhtenäisenä kaikilla käyttäjän laitteilla. Netflix on esimerkki monialustaisesta ohjelmistosta, joka on saatavilla useilla eri laitteilla ja alustoilla [12]. Toteutettu arkkitehtuuri tarjoaa mallin globaalisti skaalautuvasta useita laitteita ja käyttäjiä tukevasta rakenteesta. Netflixin tarjoamat käyttöliittymät ovat toteutettu alustakohtaisesti, mutta ne kommunikoivat yhteisen ohjelmointirajapinnan (engl. *application programming interface*, API) kautta Javalla ja Spring-alustalla toteutetun useista

mikropalveluista koostuvan taustajärjestelmän kanssa. Netflixin käyttämät palvelimet sijaitsevat laajasti ympäri maailman, jotta yhteyden latenssi olisi mahdollisimman matala. Ohjelmisto perustuu arkkitehtuuriin, jossa laitteelta lähtevä HTTP-viesti yhdistyy API-yhdyskäytävälle (engl. *API gateway*), joka yhdistää hajautetuille taustapalveluille. Nämä palvelut palauttavat datan käyttäjän laitteelle, johon käyttöliittymä piirtyy. [13] Netflixin käyttämä malli näyttää, miten tarkasti suunnitellulla arkkitehtuurilla saavutetaan suorituskykyinen ohjelmisto palvelun käyttäjille. Tässä tutkielmassa taustajärjestelmiä ei käsitellä kuitenkaan tämän enempää, vaan huomio on käyttäjäkokemuksessa ja käyttöliittymissä.

2.3 Käyttäjäkokemus

Monialustaisen ohjelmistokehityksen tavoitteena ei ole pelkästään toimivuus eri alustoilla, vaan myös käyttäjäkokemuksen (engl. *user experience*, UX) yhtenäistäminen. [11] Kansainvälisen ISO-standardin mukaan käyttäjäkokemus muodostuu käyttäjän tunteista ja reaktioista, jotka syntyvät tuotteen käytön seurauksena [14]. Norman ja Nielsen [15] puolestaan määrittelevät käyttäjäkokemuksen kattamaan kaikki loppukäyttäjän vuorovaikutuksen muodot yrityksen, sen palveluiden ja tuotteiden kanssa. Hassenzahl ja Tractinsky [16] määrittelevät käyttäjäkokemuksen koostuvan kolmesta ulottuvuudesta: käyttäjän sisäisestä tilasta (ennakkoluulot, odotukset, tarpeet, motivaatio, mieliala), järjestelmän ominaisuuksista (monimutkaisuus, tarkoitus, käytettävyys, toiminnallisuus) sekä käyttökontekstista, jossa käyttö tapahtuu (sosiaaliset normit, aktiviteetin mielekkyys, vapaaehtoisuus).

Käyttäjäkokemus on merkittävimpiä asioita digitaalisen ohjelmiston parissa toimiessa, ja se on tärkeää huomioida ohjelmistokehityksen alkuvaiheista lähtien [11]. Käyttäjäkokemukseen vaikuttaa merkittävästi käytettävyys (engl. *usability*), jolla tarkoitetaan ohjelman helppokäyttöisyyttä halutun tavoitteen saavuttamiseksi. [17]

Käytettävyyttä voidaan pitää yhtenä tärkeimmistä osa-aluesita, joista käyttäjäkokemus muodostuu [9].

Artikkelissa [9] esitellään termi monialustainen käyttäjäkokemus CPUX Hassenzahlin ja Traztinskyn [16] UX:n määritelmän pohjalta. Monialustaisten ohjelmistojen kontekstissa käyttäjäkokemus muodostuu monimutkaisemmasta kokonaisuudesta kuin yksialustaisen ohjelmiston tapauksessa. Monialustaiseen käyttäjäkokemukseen vaikuttavat järjestelmän suunnittelu, kuten jakautuminen eri laitteille ja käyttöliittymien yhtenäisyys sekä käytettyjen laitteiden tekniset ominaisuudet kuten näyttökoko, syötetavat, suorituskyky, sekä vuorovaikutuskonteksti (käyttötarkoitus, käyttöympäristö ja käyttöasento). Tämän vuoksi on tärkeää huomioida niin sanottu horisontaalinen vuorovaikutus, jossa käyttäjä voi suorittaa tehtävää usealla alustalla. Monialustaisen käyttäjäkokemuksen arvioinnissa tulisi siis huomioida käytön jatkuvuus ohjelmaa käytettäessä usealla laitteella. Tässä tutkielmassa käyttäjäkokemuksella viitataan juuri tähän määritelmään monialustaisesta käyttäjäkokemuksesta. Seuraavassa luvussa käsitellään monialustaista ohjelmistokehitystä monialustaisen käyttäjäkokemuksen saavuttamiseksi erilaisilla suunnitteluperiaatteilla ja teknisillä ratkaisuilla.

3 Monialustaisten ohjelmistojen kehitys

Käyttäjät odottavat usein sovelluksilta mahdollisuutta käyttää niitä useallammalla laitteella. Ohjelmistojen jakelu monialustaisesti on kuitenkin merkittävä haaste kehittäjien näkökulmasta. Kaikkia monialustaisuuden tuomia mahdollisuuksia ei myöskään hyödynnetä kunnolla [8]. Eri alustoilla on omat ohjelmointikielensä ja tekniikkansa, minkä lisäksi laitteet vaihtelevat näytön koon, suorituskyvyn ja syötetapojen suhteen. Tässä luvussa syvennytään aikaisempaan tutkimukseen monialustaisista ohjelmistoista käyttäjäkokemuksen ja kehitysratkaisujen näkökulmasta. Tärkeimpiä tutkimuslähteitä listattu seuraavan sivun taulukossa 3.1.

3.1 Yhtenäisen käyttäjäkokemuksen rakentaminen

Yhtenäisen monialustaisen käyttäjäkokemuksen kannalta olennaista on saumaton siirtymä laitteelta ja alustalta toiselle sekä muutosten automaattinen synkronointi eri versioiden välillä. Siirtymät laitteiden välillä helpottuvat, kun käyttöliittymät ovat mahdollisuuksien mukaan samankaltaiset ulkonäöltään ja toiminnallisuuksiltaan. [8][11] Ohjelman tilan päivittyminen varmistetaan toimivilla taustaohjelmilla ja tietokannoilla. [13] Monialustaisen käyttäjäkokemuksen kannalta erityistä huomiota tulisi kiinnittää myös suorituskykyyn, erilaisiin käyttötapoihin, saavutettavuuteen [18], käyttöliittymän johdonmukaisuuteen ja pääsyyn laitteen ominaisuuk-

Taulukko 3.1: Yhteenveto tärkeimmistä tutkimuslähteistä

Lähde	Term.	UX	Saav.	Suun.	Keh.	Ark.	Testaus
[1]	X						
[3]	X	X					
[5]					X		X
[6]		X			X		X
[7]					X		
[8]	X	X		X			
[9]	X	X				X	
[11]		X		X			
[13]						X	
[18]			X		X		
[19]				X	X		
[20]	X			X			
[21]					X		
[22]					X		
[23]					X		
[24]					X		
[25]		X			X		
[26]					X		
[27]					X		X
[28]		X	X				
[29]					X		

Sarakkeet: Terminologia, käyttäjäkokemus, saavutettavuus, suunnitteluperiaatteet, kehitysratkaisut, arkkitehtuuri, testaus

siin. [5] Lisäksi tutkimusartikkelissa [8] löydettiin kolme merkittävintä ongelmaa monialustaisisten ohjelmistojen kehityksessä:

1. Laitteiden välisen vuorovaikutuksen suunnittelu on vaikeaa.
2. Käyttöliittymien sovittaminen eri alustoille on monimutkaista.
3. Monialustaisten ohjelmistojen testaamiseen ei ole riittävästi työkaluja ja menetelmiä.

Tutkielmassa [11] pyrittiin selvittämään, kuinka tärkeää käyttäjäkokemuksen kannalta on yhtenäinen käyttöliittymä monialustaisessa ohjelmistossa ja miten käytettävyyttä voidaan parantaa yhtenäisen käyttäjäkokemuksen varmistamiseksi erityisesti musiikin kuuntelemiseen tarkoitetuissa sovelluksissa. Tutkielmassa suoritettiin kyselytutkimus, johon osallistui 58 vastaajaa. Keskeinen havainto oli, että mahdollisimman yhtenäinen visuaalinen sekä toiminnallinen suunnittelu parantaa ohjelmiston käyttäjätyytyväisyyttä ja helpottaa käyttäjän siirtymiä laitteelta toiselle. Yhtenäisyyttä luodaan visuaalisesti muun muassa samankaltaisella elementtien

asettelulla, fonteilla ja värimaailmalla. Selvisi, että peräti 91.7 % testaajista tykkäsi elementtien pyöristetyistä kulmista verrattuna teräviin kulmiin. Toiminnallista yhtenäisyyttä voidaan luoda esimerkiksi painikkeiden ja toimintojen samanlaisella käyttäytymisellä. Käyttäjätyytyväisyyttä paransi mahdollisuus vaihtaa light/dark tilojen välillä. Erityisesti havaittiin että minimalistinen lähestymistapa ja leveät välit elementtien välillä koettiin mielekkäiksi käyttäjien keskuudessa. Pienetkin erot ulkoasussa eri versioiden välillä voivat heikentää käyttäjäkokemusta ja lisätä turhautumista ohjelmaan. Kyselyn otoskoko (58 osallistujaa) on melko pieni, ja osallistujista merkittävä osa (45 %) oli opiskelijoita. Tämä rajoittaa yleistettävyyttä laajempaan käyttäjäryhmään. Suurin osa, 41.7 % vastaajista oli 24–29 -vuotiaita. 93.3 % vastaajista käytti musiikin kuunteluun mobiililaitteita ja 48.3 % tietokonelaitteita ja 35.2 % vastaajista käyttävät aktiivisesti ohjelmistoa usealla laitteella. Tutkielmassa todetaan, että tehdyt havainnot eivät välttämättä yleisty muihin ohjelmistotyyppeihin kuin musiikkisovelluksiin. Artikkeli on yksi kattavimmista monialustaiseen suunnitteluun ja käyttäjäkokemukseen liittyvistä lähteistä. Musiikkisovellukset ovat myös yksi kaikkein laajimmin monialustaisuutta tavoittelevia ohjelmistotyyppejä, joten ne sopivat hyvin monialustaisten ohjelmistojen tutkimiseen.

Käytettävyyden arviointi on olellinen osa monialustaisia ohjelmistoja ja käyttäjäkokemusta. Kuitenkin puute standardoidusta tavasta arvioida monialustaista käytettävyyttä on merkittävä haaste ohjelmistokehittäjille [6]. Tutkimuksessa [3] arvioitiin käytettävyyttä ISO 9241-11 -standardin mukaisesti, tehokkuuden, vaikuttavuuden ja käyttäjätyytyväisyyden näkökulmasta. 50 osallistujaa testasi lemmikkien hoitopalvelusovellusta mobiilisovelluksena sekä tietokoneversiota verkkosovelluksena. Käyttäjät suorittivat sarjan tehtäviä ja kommentoivat tekemiään huomioita. Molemmilla alustoilla havaittiin omat haasteensa, mutta verkkosovelluksen käyttö koettiin hitaampana ja aiheutti enemmän virheitä. Suurimmat epäkohdat liittyivät tiedonsyöttöön, käyttöliittymän epäyhtenäisyyteen ja huonoon elementtien nimeä-

miseen. Tutkimuksessa käytetty think-aloud -menetelmä havaittiin hyväksi tavaksi testata monialustaisen ohjelmiston käytettävyyttä. Menetelmän avulla saadaan suoraan palautetta ja kehitysehdotuksia testaajan kertoaessa ajatuksiaan ääneen testaamisen aikana.

Käyttöliittymä on käyttäjän kannalta yksi tärkeimmistä asioista monialustaisten sovellusten kanssa toimiessa, joten yhtenäisyyden saavuttaminen eri versioiden kanssa vaatii vankkaa suunnittelua. Yksi tärkeä ja helposti toteutettava yhdenmukaisuutta luova elementti on värimallit. Hyvä kehitystapa on käyttää samoja värikirjastoja ja yhdistää elementit samoihin väreihin tai muotoihin kaikissa eri käyttöliittymissä. Tällä luodaan jatkuvuutta ja parannetaan käyttäjän kykyä hahmottaa paremmin käyttöliittymän toiminta käytetystä alustasta riippumatta. Elementit ja fonttikoot voivat skaalautua sopivan kokoisiksi näytön koon perusteella. Hyvä tapa on myös pyrkiä käyttämään samoja elementtejä uudelleen jos mahdollista eri laitteiden välillä. [11] Käyttöliittymän ulkonäön yhdenmukaistaminen eri alustojen välillä ei kuitenkaan aina ole edes tavoiteltava asia, sillä käyttäjät odottavat tietyiltä alustoilta tietynlaista ulkonäköä. Lisäksi, vaikka useimmilla käyttäjillä olisikin vain yksi laite sovelluksen käyttöön, vahvistaa yhtenäinen käyttöliittymä kaikilla alustoilla yrityksen brändi-identiteettiä käyttäjien mielessä [11], mikä voi olla tavoiteltavaa liiketoiminnallisista syistä. Käyttöliittymän toiminnallisuuden yhdenmukaistaminen pienentää oppimiskynnystä laitteiden välillä siirtyessä ja vähentää käyttäjän tekemiä virheitä. [5]

Käyttöliittymien suunnittelussa monialustaisesti yksi tärkeimpiä huomioon otettavia asioita on kohdelaitteen näytön koko. Ne voivat vaihdella pienistä älykelloista suuriin älytelevisioihin. Oleellisimmat näyttökoot ovat älypuhelimet ja tietokoneen näytöt. Kun etsitään ratkaisuja yhtenäiseen käyttöliittymään, on tähän kaksi yleisesti käytössä olevaa ratkaisua: responsiivinen ja adaptiivinen [20]. Responsiivisuus tarkoittaa käyttöliittymän jakamista omiin komponentteihinsa, joiden sijoittelu

näytöllä vaihtuu sen koon mukaan. Responsiivisessa suunnittelussa käyttöliittymän koodi havaitsee näytön resoluution ja mukautuu siihen automaattisesti [19]. Adaptiivisessa suunnittelussa koko käyttöliittymä suunnitellaan erikseen eri näyttökoille. Jokaiselle halutulle näyttökoolle tehdään oma versionsa ja käyttäjälle avautuu omalle laitteelle parhaiten sopiva versio. Tämä mahdollista paremman optimoinnin eri laitteille, jolloin esimerkiksi puhelimella käyttöliittymän sisältö ei ole liian pientä. Adaptiivisuuden haasteena on pidempi kehitysaika sillä eri versioiden kehittäminen on työlästä. [11][20] Esimerkiksi alunperin vain tietokoneen näytöille suunniteltu käyttöliittymä on vaikea muuttaa responsiiviseksi myöhemmin. [19]. Kuvassa 3.1 visualisoituna responsiivinen suunnittelu, jossa elementit mukautuvat ja järjestyvät näytön koon perusteella.

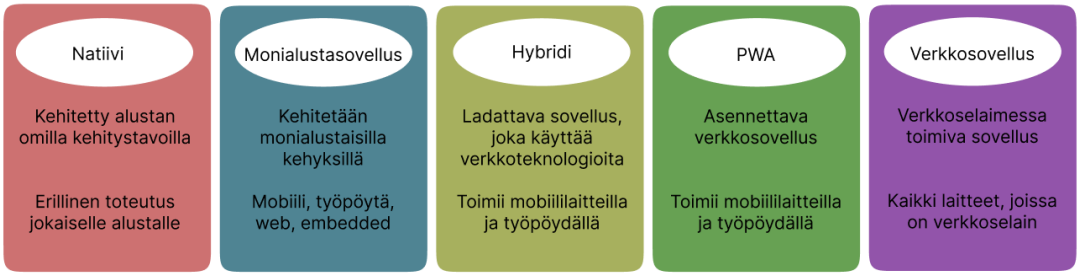
Kuva 3.1: Responsiivinen käyttöliittymä ja näyttökoot

Käyttöliittymien lisäksi pitää monialustaisessa suunnittelussa huomioida erilaiset käyttötavat ja tarkoitukset. Monialustaisessa käytössä laitteet voivat toimia samanaikaisesti tai peräkkäin. Peräkkäinen käyttö tapahtuu usein luonnollisesti sovelluksen synkronoidessa tietoa laitteiden välillä. Samanaikainen käyttö, eli ristiinalustaisuus, vaatii tarkkaa suunnittelua, sillä laitteille määritellään eri roolit osana sovelluksen vuorovaikutusta. Esimerkiksi yksi laite toimii ohjaimena ja toinen päätelaitteena. Yhä useammin käyttäjä käyttää sovellusta tai palvelua usealla laitteella eri aikoina ja eri tilanteissa. Matkaa suunnitteleva käyttäjä voi esimerkiksi aloittaa matkan suunnittelun pöytäkoneella, jatkaa selailua älypuhelimella myöhemmin ja illalla vielä tabletilla. Tällaisessa käyttötavassa korostuu käyttäjän tarve loogiselle

käyttäjäkokemukselle alustasta riippumatta. Tässä tärkeässä osassa on käyttöliittymän yhdenmukaisuus eri alustoilla. Kuitenkin, koska samanlainen käyttöliittymä ei usein ole mahdollinen tai edes tarkoituksenmukainen toteuttaa, yhtenäinen tiedon arkkitehtuuri nousee erityisen tärkeään rooliin. Tällöin samaa asiaa kuvaavat käsitteet ja tiedon kulku pysyvät samoina laitteesta riippumatta. [8]

3.2 Kehitysratkaisut

Tämä ja seuraavat kappaleet pyrkivät vastaamaan erityisesti ensimmäiseen tutkimuskysymykseen tarkastelemalla monialustaisen kehityksen erityispiirteitä ja monialustaisia sovellustyyppejä. Nämä tekniset ratkaisut muodostavat perustan yhtenäiselle monialustaisen käyttäjäkokemuksen luomiselle. Monialustaisen ohjelmiston kehittämiseen on olemassa useita lähestymistapoja, jotka voidaan jakaa kolmeen kategoriaan: monialustainen natiivikehitys, monialustaiset kehykset tai verkkoteknologiat (verkkosivut, hybridisovellukset sekä PWA). Näiden kehitystapojen eri sovellustyypit visualisoituna kuvassa 3.2.



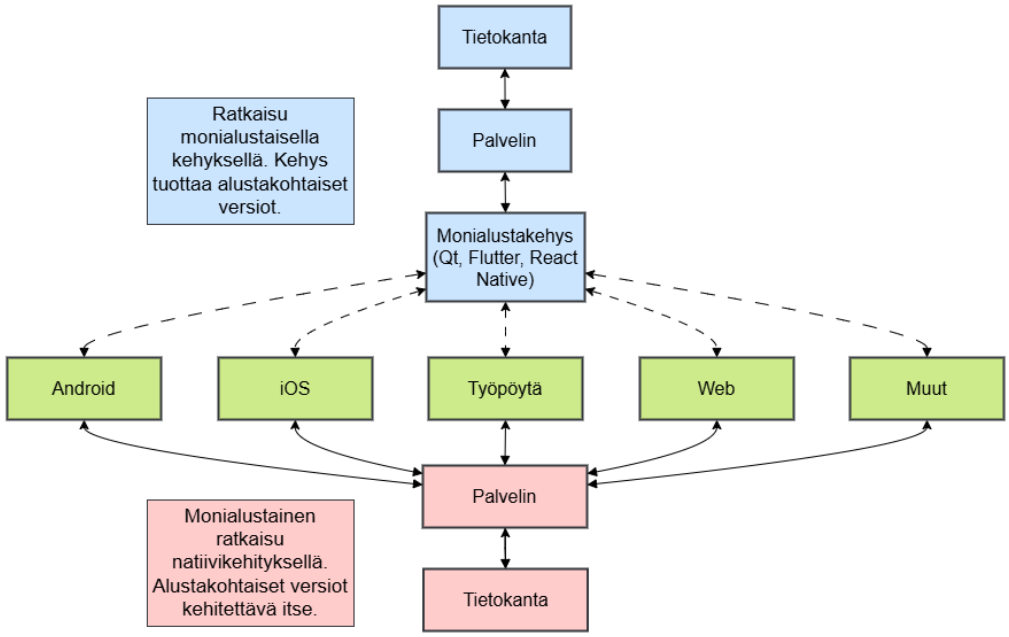
Kuva 3.2: Sovellustyypit

Natiivikehityksellä tarkoitetaan ohjelman kirjoittamista alustan tukemilla ohjelmointikielillä ja kehitystyökaluilla (engl. *Stanrdard Development Kits*, SDK) esimerkiksi Kotlin Androidilla, Swift iOS:llä [7][22] tai .net Framework Windowsilla [23]. Vahvuuksia natiivikehityksessä on täysi pääsy kaikkiin alustan ominaisuuksiin, ja paras mahdollinen suorituskyky. Alustan kehittäjä tukee usein alustan omia ohjel-

mointikieliä omilla apuohjelmillaan, esimerkiksi jetpack-compose käyttöliittymäkehys Androidille [30]. Natiivikehitys on usein hyvä ratkaisu, jos kyse on yhdellä alustalla toimivasta sovelluksesta. Kun siirrytään monialustaiseen natiivikehitykseen, jokaiselle alustalle pitää tehdä oma versio, eikä koodia voi jakaa näiden välillä. Tästä seuraa haasteita kuten pidempi kehitysaika ja kustannukset [7]. Monialustainen natiivikehitys vaatii erityistä osaamista kehittäjiltä kaikilta kohdealustoilta, jolloin useiden eri tekniikoiden käyttö voi osaamisen puutteen takia heikentää monialustaista käyttäjäkokemusta. Tällaisen ohjelmiston ylläpito on myös hankalaa, sillä useita eri koodipohjia on jatkuvasti päivitettävä ja testaukseen pitää käyttää enemmän aikaa. [24] Natiivikehityksessä useiden eri käyttöjärjestelmien tuntemuksen rinnalle lisähaasteen tuo vielä eri versiot samasta käyttöjärjestelmästä [18].

Osa natiivin kehityksen ongelmista on ratkaistavissa monialustaisilla kehyksillä, joilla voi tuottaa monialustasovelluksen. Monialustasovelluksen tarkoituksena on saman lähdekoodin käyttö eri alustoilla, jolloin ei tarvitse kehittää alustakohtaisia versioita kuten natiivikehityksessä. Monialustaiset kehitystyökalut ovat nousseet merkittävästi suosiossa natiivikehityksen rinnalle [24]. Monialustaisen kehyksen käytön vahvuus on ohjelmiston lyhyempi kehitysaika, pienempi koodin määrä ja siten pienemmät kustannukset [6] ja vähemmän teknologioiden osaamisvaatimuksia verrattuna natiivikehitykseen. Myös ohjelman ylläpitotyö julkaisun jälkeen sekä laajentaminen uusille alustoille on helpompaa. Kuvassa 3.3 tuodaan esiin, miten monialustaisen kehyksen käyttö vaikuttaa ohjelmiston perusarkkitehtuuriin monialustaiseen natiivikehitykseen verrattuna, kun kehys tuottaa alustakohtaiset versiot kehittäjän puolesta.

Merkittäviä haasteita monialustakehyksissä on kuitenkin käyttöliittymän kehityksen haasteet [22], heikompi suorituskyky verrattuna natiivisovelluksiin [7] ja usein huonompi pääsy alustakohtaisiin ominaisuuksiin [22] sekä natiivinomaisen käyttäjäkokemuksen saavuttaminen kohdealustoilla [5]. Natiivinomaisella käyttäjäkokemuk-



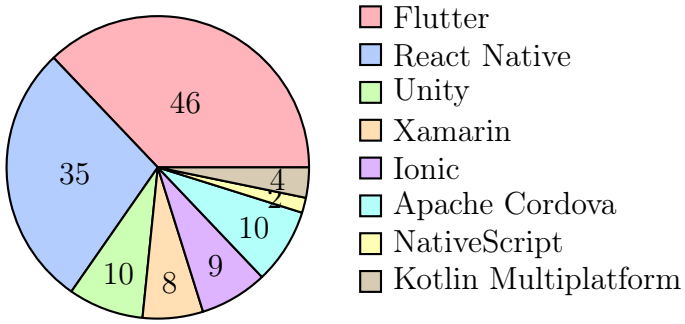
Kuva 3.3: Monialustainen perusarkkitehtuuri monialustakehyksellä ja natiivisti

sella tarkoitetaan käyttöliittymän visuaalista ilmettä, tuntumaa ja vuorovaikutusta, joka on käyttäjälle tuttu tyypillisistä alustan natiivisovelluksista.

Monialustaisia kehyksiä on tarjolla useita vaihtoehtoja eri tarkoituksiin. Tietty kehys saattaa keskittyä vain muutamiin alustoihin, ja tuki muille alustoille voi olla vähäistä, kehitysvaiheessa tai sitä ei ole lainkaan. Mobiilikehittäjien keskuudessa käytetyimpiä ovat Googlen kehittämä Flutter ja Metan kehittämä React Native [31], joiden painopiste on mobiilikehityksessä, mutta niitä voi käyttää myös työpöytä- ja verkkoalustoilla. Muita monialustaisia kehyksiä ja niiden markkinaosuuksia vuonna 2023 tehdyssä JetBrainsin kyselyssä [31] on listattu kuvassa 3.4.

Merkittävä monialustainen kehys on myös QT, jonka käyttö on yleistä sulautetuissa järjestelmissä, mutta sen esiintyminen monialustaisessa tutkimuksessa, jonka painopiste on mobiililaitteilla, on vähäistä. Osa monialustasovelluksista on niin sanottuja hybridisovelluksia, joita käsitellään lisää verkkoteknologioiden yhteydessä.

Verkkoteknologiat, eli HTML, CSS ja JavaScript ovat laajasti monialustaisia, sillä ne toimivat kaikilla verkkoselaimia tukevilla käyttöjärjestelmillä. Verkkotekno-



Kuva 3.4: Mobiilikehittäjien käytetyimmät monialustakehykset 2023 [31]

logioiden avulla tuotetaan verkkosovelluksia, jotka itsessään ovat yksialustaisia ja toimivat vain web-alustalla, mutta käytännössä monialustaisesti kaikissa laitteissa, joissa on verkkoselain. [21] Erityisesti vanhemmissa verkkosovelluksissa oli tavanomaista, että sivustolle tehdään erilliset versiot isommille tietokoneen näytöille, ja pienille mobiilinäytöille, eli adaptiivinen suunnittelu. Nämä eri versiot eivät välttämättä sisällä kaikkia samoja toimintoja, sillä mobiiliversion toiminnallisuudet voi olla rajatumpia kuin täyden työpöytäversion. Nykyään yleisempää on responsiivinen suunnittelu, jossa sivuston käyttöliittymä on ohjelmoitu mukautumaan näytön koon mukaisesti. [19][20] Verkkosovellusten haaste on heikompi suorituskyky verrattuna muihin sovellustyyppeihin, sekä hyvin rajoitettu pääsy alustan ominaisuuksiin [25].

Verkkoteknologioilla voidaan kehittää hybridisovelluksia, jotka toimivat 'natiivin kontin' sisällä. Käytännössä hybridisovellus on natiivisovellus, joka näyttää verkkosisältöä upotetussa selainnäkymässä [26]. Hybridisovelluksia voidaan kehittää monialustaisilla hybridikehyksillä kuten Apache Cordova (entinen PhoneGap) ja Ionic. [6] Hybridisovellukset asennetaan mobiililaitteilla samoin kuin natiivi- ja monialustasovellukset, esimerkiksi Androidilla Google Play sovelluskaupasta. Hybridisovellusten hyödyt ja haitat ovat hyvin samankaltaisia kuin monialustasovelluksissa [25]. Koska ohjelma toimii natiivin kehyksen sisällä, hybridisovelluksilla on parempi pääsy laitteen natiiviominaisuuksiin verrattuna verkkosovellukseen. [6] Hybridisovellukset eroavat siis monialustasovelluksista toteutustavaltaan ja suoritusympäristöltään.

Hybridisovelluksen suoritusympäristö on verkkoteknologiat natiivin kontin sisällä, kun monialustasovellukset pyrkivät renderöimään käyttöliittymän natiivinkaltaisesti tai jopa natiiviin muotoon [6]. Molemmilla kehitystavoilla pyritään kuitenkin samaan tarkoitukseen, eli koodin uudelleenkäytettävyyteen eri alustoilla. Aina näitä sovellustyyppejä ei erotella toisistaan, kuten kehysten luokittelussa kuvassa 3.4, jossa on molempia sekä monialusta- että hybridikehyksiä. Kuvassa 3.3 hybridikehys toimisi samoin kuin monialustakehys.

Toinen verkkoteknologiaa vielä suoremmin hyödyntävä ohjelmistotyyppi on nimeltään progressiivinen verkkosovellus (engl. *progressive web application*, PWA). PWA-sovellus yhdistää verkkosovelluksen ja natiivisovelluksen, kuitenkin hyvin eri tavalla kuin hybridisovelluksessa. PWA-sovellus on pohjimmiltaan normaali verkkosivu, joka toimii natiivisovelluksen kaltaisesti ilman näkyvää verkkoselainta. [26] Tavallisen verkkosivun tapaan PWA-sovelluksella on rajoitettu pääsy laitteen natiiveihin ominaisuuksiin [6]. PWA-sovellus asennetaan suoraan selaimesta älypuhelimelle tai tietokoneelle.

3.3 Kehitysratkaisujen tarkastelu

Nykyään mobiililaitteet ovat laajemmin käytössä kuin työpöytä- tai kannettavat tietokoneet [18]. Akateeminen tutkimus keskittyy myös vahvasti mobiilialustoille sekä monialustakehyksiin. Seuraavaksi tarkastellaan muutamia tutkimuksia liittyen kehitysratkaisuihihin.

Andreas Biørn-Hansenin ym. tutkimusartikkelissa [22] selvitettiin monialustaisten kehysten käyttöä. Tutkimus perustui kyselytutkimukseen, jossa tarkoituksena oli kerätä mobiilikehittäjien kokemuksia ja näkemyksiä monialustaisista kehyksistä. Taulukossa 3.2 on esitetty yleisimmin mainitut haasteet, jotka kehittäjät nostivat esiin kyselyn vastauksissa. Aineisto perustuu 101 vastaukseen.

Taulukko 3.2: Mobiilikehittäjien esiin nostamia haasteita monialustaisissa kehyksissä. Muokattu lähteestä [22]

Haaste	**Mainintojen osuus**
Suorituskyvyn heikkous verrattuna natiiviin	62 %
Käyttäjäkokemuksen puutteet	57 %
Kehyksen kehittymättömyys ja riskit	54 %
Heikot työkalut hyvän käyttöliittymän luomiseen	50 %
Yhteisöjen epäkypsyys	35 %
Vaikeus käyttää laitteen rajapintoja	32 %
Testauksen ja virheenjäljityksen vaikeus	31 %
Tietoturvaongelmat	13 %

Ylimpänä on useissa lähteissä havaittu ilmiö, että monialustaisilla kehyksillä kehitetyt ohjelmistot ovat suorituskyvyltään heikkoja verrattuna natiivisovelluksiin. Kuitenkaan ero ei ole välttämättä merkittävä käyttäjän kannalta erityisesti tehokkailla laitteilla. Tärkeää on tunnistaa ohjelmiston tarpeet, ja valita kehitysratkaisut sen perusteella. Toisena nousee esiin käyttäjäkokemuksen haasteet ja neljäntenä siihen oleellisesti liittyvä käyttöliittymäkehityksen vaikeus. Nämä ongelmat mainitaan usein myös muissa lähteissä. Käyttäjät pitävät natiivisovelluksia houkuttelevampina ja luotettavampina, erityisesti sovelluksen noudattaessa alustan omia ohjeistuksia. Monialustaisten kehysten välillä voi olla merkittäviä eroja muun muassa eleohjauksen ja natiivikomponenttien tuen osalta. Yhteisöjen epäkypsyys mainitaan 35 % tapauksista. Kehittäjien on kehystä valittaessa tärkeää miettiä ohjelman toimintaa tilanteessa, jossa kehyksen kehitys lopetetaan tai se hidastuu. Tutkimuksissa nousee esiin välillä vanhentuneita kehyksiä, kuten MoSync. Vaikeutta käyttää laitteen rajapintoja nosti esiin 32 % vastaajista, joka on suosittu aihe tutkimuskirjallisuudessa, mutta tiedot on ristiriitaisia. Näitä ominaisuuksia on esimerkiksi pääsy kameran, yhteystietojen ja GPS:n käyttöön. Osassa tutkimuksissa pääsy nähdään olevan rajoittunutta, kun taas toisissa korostetaan hyvää yhteensopivuutta. [22] Huomioitavaa on, että tutkimuksen aineisto kerättiin vuonna 2016. Siinä mainitut kehykset ovat monet joko poistuneet kokonaan käytöstä, tai käyttö nykyään on vähäistä. Osallistu-

jat kyselytutkimukseen kerättiin julkisilta internetfoorumeilta ja sosiaalisen median palveluista pyrkien tavoittamaan laajasti mobiilikehittäjiä. 101 osallistujan otos on kohtalaisen pieni, eikä siitä voida tehdä varmaa johtopäätöstä että otos edustaisi kehitysyhteisöä laajasti. Artikkelin havainnot ovat kuitenkin hyödyllisiä ja linjassa muiden lähteiden kanssa.

Tutkimusartikkelissa Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter [7] testattiin kahta suosittua monialustaista kehystä (Flutter ja React Native) sekä Kotlin-kieleen perustuvaa natiivia Android-kehitystä toteuttamalla jokaisella ristinolla-peli Android alustalle. Tulosten mukaan Kotlin-versio käytti vähiten muistia ja tallennustilaa, ja CPU käyttö oli vähäisintä. React Native pärjäsi heikoiten, mutta koodin uudelleenkätettävyyttä tarkasteltaessa React Native saavutti korkeimman tason. Vertailu ei anna vastausta siihen, mikä kehitystapa on paras, mutta se havainnoillistaa hyvin natiivin kehityksen etuja verrattuna monialustaisiin kehyksiin. Tutkimuksessa kehitetty ohjelma oli hyvin yksinkertainen, jonka lisäksi ohjelma toimi vain paikallisesti android alustalla, joten tämän perusteella ei saada tietoa minkälaisia tuloksia muilla alustoilla saataisiin. Tulokset ovat kuitenkin täysin linjassa vastaaviin lähteisiin, joissa natiivikehityksen selvä tehokkuusetu nousee esiin. Vastaavia tuloksia saatiin artikkelissa [27].

Tutkimuksessa [28] selvitettiin monialustaisten kehysten saavutettavuustukea mobiilialustoilla ja havaittiin, että se on usein rajallinen verrattuna natiivikehitykseen. Saavutettavuuden kannalta kriittiset näytönlukijat olivat huonosti tuettuja tutkimuksen kohteina olleilla kehyksillä Xamarin ja React Native. Sen sijaan natiivikehityksessä Android ja iOS tukevat hyvin saavutettavuusominaisuuksia. Monialustaisia kehyksiä käytettäessä voidaan silti saavuttaa natiivi API:en saavutettavuusominaisuudet kirjoittaen alustakohtaista koodia erikseen kaikille alustoille. Tämä kuitenkin heikentää perustelua käyttää monialustaista kehystä, joiden keskei-

nen etu on koodin uudelleenkäytettävyys. Samankaltaisia käytettävyyskehityksen haasteita monialustakehyksien ja yleisesti monialustaisuuden näkökulmasta kertoo myös tutkimusartikkeli [18], jossa esitetään myös lähestymistapa miten saavutettavuus voidaan huomioida paremmin monialustaisten käyttöliittymien suunnittelussa ja toteutuksessa.

Artikkelissa [25] vertailtiin vesilaitoksen asiakassovellusta kehitettynä sekä hybridisovelluksena että natiivina. 76.67 % (46/60) käyttäjistä ei havainnut merkittävää eroa versioiden välillä, ja vain 13.33 % käyttäjistä havaitsi suorituskykyeroja natiivin hyväksi. Tutkielmassa havaittiin, että hybridisovellus toimii hyvänä vaihtoehtona monialustaisuuden saavuttamiseen. Hybridikehityksen yhtenä etuna nähtiin useiden kehittäjien osaaminen verkkoteknologioista, joita hybridikehitys hyödyntää, jolloin uusien teknologioiden opettelemista alustakohtaisesti ei tarvita. Tutkielmassa todetaan natiivikehitys hyväksi vaihtoehdoksi ainoastaan jos ohjelmassa tarvitaan vahvaa pääsyä alustan rajapintoihin ja kehitystyökaluihin tai kun tarvitaan suorituskykyisiä käyttöliittymiä. Artikkeli on vuodelta 2015, mutta se tarjoaa edelleen hyödyllistä näkökulmaa verkkotekniikoiden hyödyntämisestä monialustaisessa kehityksessä. Täysin ei ole tiedossa kuinka suosittuja hybridisovellukset ovat suhteessa monialustasovelluksiin. Näyttäisi kuitenkin vahvasti siltä, että hybridisovellukset ovat vanhempi teknologia, joka on menettänyt suosiotaan monialustasovelluksille. Verkkotekniikoita vielä suoremmin hyödyntävät PWA-sovellukset ovat myös todennäköisesti korvanneet tarvetta hybridisovelluksille.

Tutkimusartikkelissa [29] tarkastellaan verkkoteknologioilla kehitettyjä työpöytäsovelluksia kehitettynä Electron ja NW.js-kehyksillä, jotka ovat aktiivisimmat verkkoteknologiaa käyttävät monialustakehykset työpöytäsovelluksille. Monialustaisuuden näkökulmasta tutkimuksessa tuodaan esiin samoja ongelmia kuin monialustakehyksillä yleisesti, mutta tutkielmassa korostetaan kehittäjien kohtaamia ongelmia, kuten sovelluskontin rakentamiseen ja julkaisuun liittyvää (engl. *build*

and deploy) monimutkaisuutta, koodikirjastojen uudelleenkäytettävyyttä, API-yhteensopivuusongelmia ja toiminnallisuuseroja eri käyttöjärjestelmissä. Vaikka Electron kehitysalustana lupaa yhtä koodipohjaa kaikille alustoille, kehittäjät joutuvat käytännössä tekemään alustakohtaisia muutoksia. Lisäksi sovelluskontin rakentaminen ja ohjelman julkaisu eri kohdealustoille on yleinen haaste, joka on erityisen tärkeä huomio monialustakehyksistä, jota muissa artikkeleissa ei juuri nouse esiin. Tutkimuksessa todetaan myös ettei verkkoteknologioiden käyttöä työpöytäsovelluksissa ole tutkittu laajasti.

3.4 Testaus

Monialustaisen sovelluksen kattava alustakohtainen testaus on välttämättömyys yhtenäisen käyttäjäkokemuksen ja käytettävyyden saavuttamiseksi. Useissa lähteissä mainitaan monialustaisen ohjelmiston testaamisen olevan vaikeaa, sillä tehtävää varten on vähän valmiita ratkaisuja [5][6][8]. Monialustaisen ohjelmiston testaaminen on merkittävästi hankalampaa kuin yksialustaisen ohjelmiston, sillä ohjelman toiminta pitää varmistaa erikseen kaikilla laitteilla ja alustoilla. Tämä on usein aikaa vievää ja vaatii kehittäjillä olevan käytössään kaikki kohdelaitteet ja alustat [5]. Luomalla testiskenaarioita voidaan arvioida kuinka samankaltaista tehtävän tekeminen eri laitteilla on. Monialustaista testausta voidaan helpottaa siirtämällä osa toiminnoista suoritettavaksi taustapalvelimelle, sen sijaan että ne suoritettaisiin käyttäjän laitteella. [19] Monialustaiset kehykset tarjoavat yleensä valmiita tapoja testata ohjelmiston toimintaa [27], mutta ohjelmointivirheiden korjaaminen voi olla hankalampaa kuin natiivitoteutuksella. Natiivisovellusten testaus on usein helpompaa, sillä alustan valmiit testaustyökalut ovat vapaasti käytössä. [7] Tutkimusartikkelissa [22] tehdyssä kyselyssä 51 % kertoi tekevänsä lähinnä manuaalista testausta. Monet ohjelmistoympäristöt sisältävät valmiita toimintoja automaattiseen testaamiseen, ohjelmointivirheiden korjaamiseen ja suorituskyvyn seuraamiseen. CI/CD-työkalut

ovat myös kehittyneet tukemaan paremmin monialustaisia projekteja. Nämä työkalut automatisoivat joitain toimintoja, kuten testausta ja julkaisua. [5]

4 Pohdinta

Tässä tutkielmassa tarkasteltiin monialustaista ohjelmistokehitystä ja siihen liittyviä teknisiä ratkaisuja erityisesti käyttäjäkokemuksen näkökulmasta. Kirjallisuuskatsauksen perusteella havaittiin, että tutkimuskenttä on vahvasti mobiilipainoitteinen. Alan tutkimuksessa korostuu erityisesti monialustaiset kehityskehykset, joista monet on suunniteltu ensisijaisesti Android ja iOS-alustoille. Tuki työpöytä- ja verkkosovelluksille on usein lisätty myöhemmin, joka selittää osaltaan miksi tutkimus keskittyy niin vahvasti mobiilialustoihin. Mobiilialustat ovat myös merkittävästi suositumpia kuin tietokonelaitteet [32]. Lisäksi näiden kehysten aktiivisia kehittäjiä muilla kuin mobiilialustoilla on todennäköisesti vähän ja käyttö on mahdollisesti myös teknisesti rajoittunutta. Artikkeleissa ei juuri nouse esiin merkittävää riskiä siitä, että monialustakehyksen kehitys voi yllättäen hidastua [33] tai loppua kokonaan.

Tutkielman perusteella voidaan todeta, että monialustaiseen käyttäjäkokemukseen liittyy monia tekijöitä, kuten käyttöliittymän rakenne ja vuorovaikutus, tekninen toteutus, valitut kehitysratkaisut, testaaminen ja ylläpidon laatu. Monialustaisuus tuo kehitykseen merkittäviä haasteita, sillä jokainen ohjelmiston kohdealusta tuo mukanaan omat tekniset rajoitteet, laitekohtaiset ohjaustavat, käyttäjien odotukset ja kehitystiimin osaamisvaatimukset. Monialustaista käyttäjäkokemusta suunnitellessa kannattaisi pyrkiä hyödyntämään jokaisen laitteen ominaisuuksia ja ohjaustapoja, sen sijaan että pyritään täysin identtiseen toteutukseen. Monet ohjelmistot voisivat jopa hyötyä selkeästä ominaisuuksien eriyttämisestä, esimerkiksi kie-

lenoppimissovellus voisi työpöytäversioissa käyttää enemmän kirjoitustehtäviä kun käytössä on näppäimistö ja mobiililaitteilla suosittaisiin pelillistämistä.

Yksi keskeinen huomio tutkielmassa on terminologian ja lähestymistapojen hajanaisuus. Monialustaisuudelle on useita termejä, jotka tilanteesta riippuen tarkoittavat eri asioita. Lähdemateriaalin ymmärtäminen edellyttää ensin selvittämään mitä monialustaisuudella tarkoitetaan ja missä laajuudessa sitä tarkastellaan. Niinpä osaksi tutkielman sisältöä nousi myös terminologian selventäminen sekä erilaiset kehitysratkaisujen ja niihin liittyvien sovellustyyppien esittely. Tämän pohjalta muodostui kokonaiskuva monialustaisuuden vaihtoehdoista ja niiden vaikutuksista kehitykseen ja käyttäjäkokemukseen. Omasta näkökulmastani tämä terminologinen ja tekninen aiheen selkiyttäminen tuntui myös erityisen tärkeältä. Ennen tutkielman aloittamista ero hybridi-, PWA- tai monialustasovelluksen välillä ei olisi ollut itselleni selkeä. Tutkielmasta jätettiin tarkoituksella pois videopelit ja pelimoottorit, sillä vaikka ne selkeästi ovat monialustaisia, pelit muodostavat täysin omanlaisensa ohjelmistotyypin ja ne eivät olisi sopineet tutkielman aiheeseen eikä lähdemateriaaleissa videopelejä tai edes mobiilipelaamista huomioida lainkaan.

Tutkielma osoitti, että teknisillä ratkaisuilla on merkittävä rooli hyvän monialustaisen käyttäjäkokemuksen rakentamisessa. Ei ole lainkaan poikkeuksellista, että sovellus on saatavilla vain mobiilialustoille ja mobiililaitteiden suosio voi heikentää yritysten kiinnostusta tukea muita alustoja, vaikka oikeilla kehitysratkaisuilla voitaisiin tavoittaa isompi käyttäjäkunta ja parempi käyttäjätyytyväisyys pienellä lisävaivalla. Itselläni käytössä oleva älykotilaitteen ohjaussovellus osoittaa tämän konkreettisesti. Kyseinen sovellus toimii sulautettuna itse laitteessa, sekä Android- ja iOS-sovelluksina. Ohjelmisto on esimerkki sekä monialustaisuudesta että ristiinalustaisuudesta, jossa laitteilla on erilaiset tehtävät. Laitetta voi käyttää pelkästään sen omasta käyttöliittymästä, mutta mobiiliversiot tuovat siihen lisäominaisuuksia, kuten etäohjauksen ja aikataulujen tekemisen. Kuitenkin jos tällaisia älykotilaittei-

ta olisi useita, pelkästään mobiilikäyttöliittymä kävisi nopeasti epäkäytännölliseksi niiden hallinnointiin. Jos ohjaussovellus olisi saatavilla myös työpöytä- tai verkkosovelluksena, se toisi selvää lisäarvoa esimerkiksi toimintaraporttien lukemiseen ja aikataulujen hallintaan. Sovelluksen käyttäjäkokemusta on myös heikentänyt erityisesti tekniset ongelmat, joka korostaa tutkielman rajauksen osuvuutta, eli tekniset ratkaisut ovat keskeinen osa monialustaisen käyttäjäkokemuksen rakentamista.

Jatkossa monialustaisuuden tutkimuskenttä kaipaisi tarkastelua laajempana kokonaisuutena niin, että huomioitavina alustoina olisi useammin mobiililaitteiden lisäksi tietokoneet ja verkkoselaimet sekä jatkuvasti kasvava laitteistojen kirjo sisältäen esimerkiksi älykellot, kodinkoneet ja ajoneuvojen näytöt. Tulevaisuudessa korostuu entisestään tarve yhtenäisen käyttäjäkokemuksen rakentamiselle monimuotoisille laitteistoille ja käyttötavoille. Tutkimuskirjallisuus aiheesta tarvitsisi myös teknisempää ja syvällisempää tarkastelua kehitysratkaisujen näkökulmasta. Nähdäkseni yhdessäkään artikkelissa ei esimerkiksi esitelty monialustakehysten teknistä toimintaa.

Tutkielman aihe valikoitui kiinnostuksesta käyttöliittymien kehittämiseen ja halusta ymmärtää paremmin monialustaista ohjelmistokehitystä. Monialustaisuus on nykyään täysin normaali osa ohjelmistoja, sillä käytännössä kaikki kaupalliset sovellukset toimivat useilla alustoilla. Yksialustaiset tai vain paikalliset sovellukset ovat harvinaisuus. Tästä huolimatta aihe jää vaille kunnollista huomiota yliopisto-opetuksessa.

Kirjallisuuslähteitä olisi voinut etsiä laajemmin ilman monialustaisuuteen viittaavia termejä, sillä käyttäjäkokemukseen ja ohjelmistokehitykseen yleisemmin keskittyvät aineistot olisivat myös sisältäneet hyvää tietoa monialustaisuutta ajatellen. Kuitenkin tämä olisi laajentanut tutkielman tekemistä liikaa, ja nyt käytetty tapa auttoi rajaamaan aihetta sopivan kokoiseksi.

5 Yhteenveto

Tutkielmassa selvitettiin monialustaista ohjelmistokehitystä ja monialustaista käyttäjäkokemusta. Tutkimuskysymykset olivat:

1. Mitä erityispiirteitä ja teknisiä ratkaisuja liittyy monialustaiseen ohjelmistokehitykseen?

2. Miten käyttäjäkokemus voidaan yhtenäistää monialustaisissa ohjelmistoissa?

Ensimmäiseen kysymykseen vastataan kartoittamalla kolme eri teknistä kehitysratkaisua monialustaisuuden saavuttamiseksi: natiivikehitys, monialustaiset kehykset ja verkkoteknologiat. Näiden avulla toteutettavia sovellustyyppejä esiteltiin viisi: natiivisovellukset, monialustasovellukset sekä verkkoteknologioiden tavalliset nettisivut, PWA-sovellukset ja hybridisovellukset.

Tutkielma keskittyy ohjelmistokehityksen niihin osa-alueisiin, jotka ovat erityisen oleellisia juuri monialustaisuuden kannalta, ei niinkään ohjelmistokehitykseen yleisesti. Näitä erityispiirteitä on kehitysratkaisujen merkitys, sillä ne tuottavat erilaisia sovelluksia, joilla on omia vahvuuksia ja heikkouksia. Kehitysratkaisun valintaan vaikuttaa kehitystiimin koko, osaaminen, budjetti ja minkälaista ohjelmistoa kehitetään. Monialustaisella testauksella varmistetaan käyttäjäkokemuksen toimivuus ja valittujen teknisten ratkaisujen sopivuus ohjelmiston käyttötarkoitukseen.

Toiseen tutkimuskysymykseen vastataan määrittelemällä ensin termi monialustainen käyttäjäkokemus (CPUX) kuvaamaan monialustaista kontekstia paremmin. Käyttöliittymän mukautumiseen eri näyttökokojen perusteella esitellään ratkaisut

kuten responsiivinen ja adaptiivinen suunnittelumalli. Yhtenäisyyttä versioiden välillä voidaan parantaa samoilla värimalleilla ja käyttöliittymän samankaltaisilla elementeillä, joihin tietyt tekniset ratkaisut voivat mahdollistaa saman koodipohjan käytön. Täysi yhtenäisyys versioiden välillä ei ole kuitenkaan aina tavoiteltavaa. Käyttäjäkokemusta voi parantaa alustakohtaiset erot silloin, kun ne mukailee käyttöalustan yleisiä ja käyttäjälle tuttuja toimintamalleja ja ohjaustapoja. Hyvän käyttäjäkokemuksen kannalta kehityksessä tulisi myös pyrkiä huomioimaan erilaiset käyttötavat ja laitteiden väliset siirtymät.

Suurin osa käytetyistä kirjallisuuslähteistä keskittyi mobiilikehitykseen ja monialustakehyksiin, mutta tutkielmassa pyrittiin lähestymään monialustaisuutta ohjelmiston jakelumuotona kaikilla alustoilla. Laajempi tutkimus monialustaisen kehityksen erityispiirteistä ohjelmistokehityksen osa-alueena ja/tai teknisenä ratkaisuna olisi helpottanut tutkielman tekemistä ja on potentiaalinen empiirinen tutkimuskohde.

Lähdeluettelo

[1] F. Brudy, C. Holz, R. Rädle, C.-J. Wu, S. Houben, C. N. Klokmose ja N. Marquardt, "Cross-Device Taxonomy: Survey, Opportunities and Challenges of Interactions Spanning Across Multiple Devices", teoksessa *CHI Conference on Human Factors in Computing Systems Proceedings (CHI 2019)*, Glasgow, Scotland UK: ACM, toukokuu 2019, s. 28. DOI: 10.1145/3290605.3300792.

[2] W. Li, Y. Zhou, S. Luo ja Y. Dong, "Design Factors to Improve the Consistency and Sustainable User Experience of Responsive Interface Design", *Sustainability*, vol. 14, s. 9131, heinäkuu 2022. DOI: 10.3390/su14159131.

[3] N. A. Nik Ahmad, N. I. M. Hamid ja A. Mohd Lokman, "Performing Usability Evaluation on Multi-Platform Based Application for Efficiency, Effectiveness and Satisfaction Enhancement", *International Journal of Interactive Mobile Technologies*, vol. 15, nro 10, s. 4–19, 2021. DOI: 10.3991/ijim.v15i10.20429.

[4] T. Willberg-Laine, "Testing Graphical User Interfaces with Property-Based Testing", Master's thesis, University of Turku, Department of Computing, kesäkuu 2022, s. 52. url: https://urn.fi/URN:NBN:fi-fe2022062749897.

[5] O. D. Segun-Falade, O. S. Osundare, W. E. Kedi, P. A. Okeleke, T. I. Ijomah ja O. Y. Abdul-Azeez, "Developing crossplatform software applications to enhance compatibility across devices and systems", *Computer Science & IT*

Research Journal, vol. 5, nro 8, s. 2040–2061, elokuu 2024, ISSN: 2709-0043. DOI: 10.51594/csitrj.v5i8.1491.

[6] H. Abbas, S. Munir, M. Tahir, M. Noor, Q. Honey, M. A. Hamza ja M. W. Iqbal, "Enhancing User Experience: Cross-Platform Usability in Digital Banking Apps", *Journal of Computing & Biomedical Informatics*, vol. 07, nro 02, 2024. DOI: 10.56979/702/2024.

[7] B. Suri, S. Taneja, I. Bhanot, H. Sharma ja A. Raj, "Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter", sarja ICIMMI '22, Jaipur, India: Association for Computing Machinery, 2023, ISBN: 9781450399937. DOI: 10.1145/3590837.3590897.

[8] T. Dong, E. F. Churchill ja J. Nichols, "Understanding the Challenges of Designing and Developing Multi-Device Experiences", sarja DIS '16, Brisbane, QLD, Australia: Association for Computing Machinery, 2016, s. 62–72, ISBN: 9781450340311. DOI: 10.1145/2901790.2901851.

[9] K. Majrashi, M. Hamilton ja A. L. Uitdenbogerd, "Multiple User Interfaces and Crossplatform User Experience: Theoretical Foundations", teoksessa *Computer Science & Information Technology (CS & IT)*, sarja CCSEA-2015, Academy & Industry Research Collaboration Center (AIRCC), tammikuu 2015, s. 43–57. DOI: 10.5121/csit.2015.50205.

[10] Spotify. "Supported devices for Spotify". (2025), url: https://support.spotify.com/us/article/supported-devices-for-spotify/ (viitattu 02. 05. 2025).

[11] J. Venkova ja N. Essien, "Optimizing User Experience Across Platforms: A Study on Enhancing Music-Streaming Applications' Cross-Device Unification", Scope: 15 hp, Master's thesis, Jönköping University, School of Engineering,

toukokuu 2024. url: https://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-65114.

[12] Netflix. "Netflix Supported Devices". (2025), url: https://help.netflix.com/en/node/14361 (viitattu 29. 04. 2025).

[13] P. Bakker. "How Netflix Uses Java - 2025 Edition". JavaOne 2025, California. (2025), url: https://www.youtube.com/watch?v=XpunFFS-n8I (viitattu 19. 04. 2025).

[14] International Organization for Standardization. "Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems". (2019), url: https://www.iso.org/standard/77520.html (viitattu 28. 05. 2025).

[15] D. Norman ja J. Nielsen. "The Definition of User Experience (UX)". (1998), url: https://www.nngroup.com/articles/definition-user-experience/ (viitattu 11. 03. 2025).

[16] M. Hassenzahl ja N. Tractinsky, "User experience - A research agenda", *Behaviour and Information Technology*, vol. 25, s. 91–97, maaliskuu 2006. DOI: 10.1080/01449290500330331.

[17] International Organization for Standardization. "Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts". (2018), url: https://www.iso.org/standard/63500.html (viitattu 25. 03. 2025).

[18] A. Bouraoui ja I. Gharbi, "Model driven engineering of accessible and multi-platform graphical user interfaces by parameterized model-transformations", *Science of Computer Programming*, vol. 172, marraskuu 2018. DOI: 10.1016/j.scico.2018.11.002.

[19] J.-P. Voutilainen, J. Salonen ja T. Mikkonen, "On the Design of a Responsive User Interface for a Multi-device Web Service", teoksessa *2015 2nd ACM*

International Conference on Mobile Software Engineering and Systems, 2015, s. 60–63. DOI: 10.1109/MobileSoft.2015.16.

[20] E. Pirinen, "Monialustaiset käyttöliittymät: adaptiivisuus vs. responsiivisuus", Bachelor's thesis, Tampereen yliopisto, marraskuu 2023. url: https://urn.fi/URN:NBN:fi:tuni-202310259095.

[21] T. Zohud ja S. Zein, "Cross-Platform Mobile App Development in Industry: A Multiple Case-Study", *International Journal of Computing*, s. 46–54, maaliskuu 2021. DOI: 10.47839/ijc.20.1.2091.

[22] A. Biørn-Hansen, T.-M. Grønli, G. Ghinea ja S. Alouneh, "An Empirical Study of Cross-Platform Mobile Development in Industry", *Wireless Communications and Mobile Computing*, vol. 2019, s. 1–12, tammikuu 2019. DOI: 10.1155/2019/5743892.

[23] D. Alymkulov, "Desktop Application Development Using Electron Framework: Native vs. Cross-Platform", Supervisor: Timo Mynttinen, Bachelor's Thesis, South-Eastern Finland University of Applied Sciences (Xamk), huhtikuu 2019.

[24] D. Andersson ja A. Axelsson, *Evaluation of The Software Development Process for A Multi-Platform Solution in Flutter*. url: https://urn.kb.se/resolve?urn=urn:nbn:se:hj:diva-54302.

[25] P. R. M. d. Andrade, O. Frota, F. Silva, A. Albuquerque ja R. Silveira, "Cross Platform App: A Comparative Study", *Journal of Computer Science and Technology*, helmikuu 2015. DOI: 10.5121/ijcsit.2015.7104.

[26] G. de Andrade Cardieri ja L. M. Zaina, "Analyzing User Experience in Mobile Web, Native and Progressive Web Applications: A User and HCI Specialist Perspectives", sarja IHC '18, Belém, Brazil: Association for Computing Machinery, 2018, ISBN: 9781450366014. DOI: 10.1145/3274192.3274201.

[27] M. Mahendra ja B. Anggorojati, ”Evaluating the performance of Android based Cross-Platform App Development Frameworks”, sarja ICCIP ’20, Tokyo, Japan: Association for Computing Machinery, 2021, s. 32–37, ISBN: 9781450388092. DOI: 10.1145/3442555.3442561.

[28] S. Mascetti, M. Ducci, N. Cantù, P. Pecis ja D. Ahmetovic, ”Developing Accessible Mobile Applications with Cross-Platform Development Frameworks”, sarja ASSETS ’21, Virtual Event, USA: Association for Computing Machinery, 2021, ISBN: 9781450383066. DOI: 10.1145/3441852.3476469.

[29] G. L. Scoccia, P. Migliarini ja M. Autili, ”Challenges in Developing Desktop Web Apps: a Study of Stack Overflow and GitHub”, teoksessa *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, 2021, s. 271–282. DOI: 10.1109/MSR52588.2021.00039.

[30] Android Developers. ”Android’s Kotlin-first approach”. (2024), url: https://developer.android.com/kotlin/first (viitattu 05. 04. 2025).

[31] JetBrains. ”The State of Developer Ecosystem 2023”. (2023), url: https://www.jetbrains.com/lp/devecosystem-2023/development/ (viitattu 05. 04. 2025).

[32] Statcounter. ”Desktop vs Mobile vs Tablet Market Share Worldwide”. (2025), url: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet (viitattu 20. 05. 2025).

[33] W. Yin-Poole. ”Flutter forked as Flock, developer cites ‘company-wide issues at Google’”. (2014), url: https://devclass.com/2024/10/30/flutter-forked-as-flock-developer-cites-company-wide-issues-at-google/ (viitattu 20. 05. 2025).