Jami Seilonen

Mobiilisovelluksen suunnittelun vaiheet

Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2021

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Seilonen Jami

Työn nimi: Mobiilisovelluksen suunnittelun vaiheet

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: ohjelmistokehitys, esiselvitys, vaatimusmäärittelyt, riskianalyysi, käyttöjärjestelmät, sovellustyypit, ohjelmistokehykset

Tämän työn tilaajana toimi Prometec Oy. Prometec on kajaanilainen asiantuntija- ja teknologiayritys, joka keskittyy biopolttoaineiden laadun valvontaan. Työn tavoite oli tuottaa Prometecille tutkimus, jossa selvitetään, kuinka Prometecin ajoneuvovaakajärjestelmälle voidaan rakentaa mobiilisovellus.

Opinnäytetyön tavoite oli tutustua ohjelmistokehitystyön suunnitteluvaiheisiin ja selvittää, millaisilla teknologioilla mobiilisovelluksia voidaan toteuttaa. Sovelluksen kehitys aloitetaan huolellisella esitutkimuksella, jossa selvitetään projektin kannattavuus yritykselle. Kun esitutkimuksen perusteella on tehty päätös projektin aloittamisesta, voidaan siirtyä vaatimusmäärittely vaiheeseen, jossa selvitetään, miten sovelluksen pitää toimia ja millaisia ominaisuuksia se tulee sisältämään. Järjestelmäanalyysivaiheessa vaatimukset muuntuvat ohjelmistovaatimuksiksi, ja niiden perusteella voidaan suorittaa järjestelmän toiminnallinen määrittely. Tässä vaiheessa on hyvä aloittaa alustava riskianalyysi, ettei toiminnallista määrittelyä jouduta tekemään uusiksi liiallisten riskien takia. Toiminnallisen määrittelyn perusteella voidaan tehdä sovellukselle teknologiasuunnitelma, josta käy ilmi sovelluksen tekninen toteutus.

Teknologiatutkimuksen tarkoitus oli perehtyä erilaisiin sovellustyyppeihin ja ohjelmistokehyksiin. Mobiilisovellustyypit voidaan jaotella karkeasti natiivi-, cross-platform- ja hybridisovelluksiin. Sovellustyypeistä valittiin parhaiten vaatimuksia vastaava teknologia, jonka jälkeen voitiin aloittaa tutustuminen sovelluskehyksiin.

Tutkimuksen tuloksena sovelluskehyksien välillä voitiin suorittaa vertailu, jonka avulla Prometecin mobiilisovellukselle valittiin kehitysteknologia. Opinnäytetyötä voidaan tulevaisuudessa käyttää ohjenuorana, kun yrityksessä aloitetaan ohjelmistokehitys.

KAMK • University of Applied Sciences

Abstract

Author: Seilonen Jami

Title of the Publication: Steps to Plan a Mobile Application

Degree Title: Bachelor of Engineering, Information and communication technologies

Keywords: software development, preliminary study, requirements specifications, risk analysis, operating systems, application types, software frameworks

This work was commissioned by Prometec Ltd. Prometec is a technology company in Kajaani, focusing on the quality control of biofuels. The goal of this Bachelor's thesis was to conduct a study for Prometec that explores how a mobile application can be built for their vehicle scaling system.

The research was divided into two parts, the phases of software development and technology research. The material used for the research was obtained from the literature, the Internet, and Kajaani University of Applied Sciences. The software development phases deal with how an application should be planned and what issues should be considered in the planning phase. It includes feasibility study, requirements definition, system and risk analysis, and the design phase. The technology research introduces different types of applications and a few software frameworks that can be used to build a mobile application.

The thesis itself serves as a research result. It provides the practices for the planning work and can be used to select the development technology and software framework for the mobile application.

This thesis can be used as a guideline when starting software development in a company. The thesis covers the planning phase of software development, including things that needs to be addressed before the coding process. In addition, the work provides a good foundation for mobile development technologies.

Sisällys

Symboliluettelo

.NET Framework	Microsoftin kehittämä ohjelmistokomponenttikirjasto, jota Microsoft Visual Studio -ympäristöllä kehitetyt ohjelmistot käyttävät.
Asynkroninen kommunikaatio	Asynkroninen sarjaliikenne, tietokoneiden ja elektroniikkalaitteiden keskinäisessä kommunikaatiossa käytetty sarjaliikennemuoto, joka toimii yhdellä signaalilla kuhunkin suuntaan.
CSS	Cascading Style Sheets, WWW-sovelluksen kehityksessä käytettävä tiedosto, jolla voidaan muokata verkkosivun tyylimäärittelyä ja ominaisuuksia.
HTML	Hypertext Markup Language, verkkosivujen kehityksessä käytettävä kuvauskieli, jolla määritellään verkkosivun rakenne ja sen sisältö.
IDE	Integrated development environment eli ohjelmointiympäristö on ohjelma, jolla ohjelmoija suunnittelee ja toteuttaa ohjelmiston.
JavaScript	Web-ympäristössä käytettävä dynaaminen komentosarjakieli. JavaScriptin avulla verkkosivuille voidaan lisätä interaktiivisuutta ja dynaamista toiminnallisuutta.
Säie	Säikeet ovat prosesseja kevyempiä ratkaisuja, suoritusajan jakamiselle.

1 Johdanto

Tämän opinnäytetyön tilasi Prometec Oy. Prometec on kajaanilainen asiantuntija- ja teknologiayritys, joka keskittyy biopolttoaineiden laadunvalvontaan. Laadunvalvonnassa polttoainetta kuljettavan rekan tai junan kuormasta otetaan näyte Prometecin omalla Q-robotilla, joka on kehitetty vastaamaan laadunvalvonnan standardeja. Polttoaineesta eritellään laboratorio-olosuhteissa tiettyjä ominaisuuksia, kuten polttoaineen kosteusarvo ja energiamäärä. Laadutuksen jälkeen tiedot lähetetään voimalaitokselle. Näin voimalaitos varmistuu siitä, että polttoaine on mahdollista hyödyntää energiantuotannossa, eikä se maksa polttoaineen toimittajalle huonolaatuisesta tuotteesta.

Uudet tuulet kuitenkin puhaltavat Prometecillä, kun tuotekehitystiimi on ryhtynyt kehittelemään kuormatietoihin perustuvaa hallintajärjestelmää. Uusin innovaatio on Prometecin tarjoama Q-data, jossa polttoaineista saadut tiedot siirretään reaaliajassa pilveen ja sieltä asiakkaille näkyviin erilaisten taulukoiden ja graafien avulla. Järjestelmän hienous piilee sen kyvyssä laskea arvio polttoaineen energiamäärästä. Järjestelmä kykenee myös tallentamaan rekka-autojen lasti- ja painotiedot, ja lisää ominaisuuksia kehitellään jatkuvasti.

Opinnäytetyöni sai alkunsa, kun Prometec oli saanut asiakkailtaan toiveen vaakajärjestelmään liitettävästä mobiilisovelluksesta. Työni alkoi ideatason jälkeisestä ohjelmistosuunnittelusta ja päättyi siinä vaiheessa, kun sovellusta alettiin rakentamaan. Tehtävänäni oli tuottaa Prometecille sellaiset dokumentit, joista selviää, onko kyseisen mobiilisovelluksen tuottaminen kannattavaa ja mikäli on, niin kuinka se olisi kustannustehokkainta toteuttaa. Koska Prometecillä ei ole varsinaista mobiilikehitystoimintaa, piti ensimmäiseksi tehdä tutkimusta, miten mobiilisovellusta pitää oikeaoppisesti lähteä kehittämään, millaisia työkaluja tarvitaan, mitä koodikieliä pitää opetella ja mitä ohjelmistokehyksiä tarvitaan.

Salassapitosopimuksen vuoksi en voi mennä sovelluksen suunnitelmien yksityiskohtiin, joten opinnäytetyöni käsittelee ohjelmistokehityksen suunnittelun eri vaiheita yleispätevällä tasolla sekä muutamia teknologioita ja ohjelmistokehyksiä, joita voidaan käyttää mobiilisovelluksen rakentamiseen.

2 Ohjelmistokehityksen vaiheet

2.1 Esitutkimus

Projektin onnistumisen varmentamiseksi tulee tehdä huolellinen esitutkimus kehitettävästä tietojärjestelmästä. Yleisin syy projektin epäonnistumiseen on tilaajan tietämättömyys siitä, mitä se haluaa ja projektin tavoite jää epäselväksi. Myös projektin puutteelliset rajaukset sekä epäselvä tehtävien jako johtaa usein projektin keskeytymiseen. Esitutkimuksen tarkoituksena on selvittää, onko projekti ylipäätään mahdollinen toteuttaa ja onko se linjassa organisaation strategian kanssa. Esitutkimuksessa ei siis vielä rakenneta, ohjelmoida, eikä tehdä minkäänlaisia teknisiä ratkaisuja. (1.) (2.)

Esitutkimuksessa tulee selvittää ainakin:

- Projektin tausta ja tarpeellisuus
- Alustavat tavoitteet ja vaatimukset
- Alustava palvelukonsepti
- Projektin tuottamisen kustannukset
- Projektiin liittyvät riskit
- Onko projekti kannattava toteuttaa
- Projektin aikataulutus
- Projektiryhmän organisointi
- Projektin resursointi ja hallinta (3)

Esitutkimuksen ensisijainen tehtävä on tuottaa tietoa organisaation johdolle, joka päättää tietojärjestelmän hankkimisesta tai kehittämisestä. Se myös määrittelee lähtökohdat tietojärjestelmän rakentamiselle. (1)

2.2 Vaatimusmäärittely

Järjestelmän vaatimukset ovat eri sidosryhmien asettamia vaatimuksia ja toiveita tulevan järjestelmän toiminnasta. Sidosryhmillä tarkoitetaan kaikkia niitä tahoja, jotka ovat joko suorasti tai epäsuorasti tekemissä kehitettävän järjestelmän kanssa. Vaatimusmäärittelyssä ei kuitenkaan vielä oteta kantaa siihen, miten erilaiset vaatimukset toteutetaan käytännön tasolla. (1)

Vaatimusmäärittelyn vaiheet voidaan luokitella seuraavasti:

- Vaatimusten kartoitus
- Vaatimusten dokumentointi
- Vaatimusten analysointi
- Vaatimusten hallinnointi
- Vaatimusten validointi

Kun nämä vaiheet on käyty läpi, niitä toistetaan niin pitkään, kunnes vaatimusten määrä on riittävä ja järjestelmä tyydyttää sidosryhmien tarpeet. (4)

2.2.1 Vaatimusmäärittelyn vaiheet

Ensimmäinen vaihe vaatimusmäärittelyssä on tunnistaa kaikki mahdolliset sidosryhmät, joita järjestelmällä tulee olemaan. Yleensä näitä ovat ainakin järjestelmän aiotut loppukäyttäjät sekä tilaajaorganisaation edustajat. Kun kaikki sidosryhmät on tunnistettu, voidaan niiden vaatimuksia järjestelmää kohtaan alkaa kartoittamaan. Sidosryhmien edustajia voidaan joko haastatella erikseen, tai heidän ja kehitystiimin kesken järjestetään palavereita ja aivoriihisessioita, joissa vaatimukset täsmentyvät. (4)

Kerätyt vaatimukset on pakko dokumentoida tavalla tai toisella. Dokumenttia johon vaatimukset listataan, kutsutaan vaatimusdokumentiksi. Se on kehittäjäorganisaatiolle arvokas asiakirja, sillä siihen kerättyjen vaatimusten perusteella voidaan määritellä asiakkaalle myytävän tuotteen hinta. Vaatimusdokumentti toimii myös speksilistana järjestelmän koodaajalle. Siitä tulee käydä ilmi, miten sovellus toimii ja millaisia ominaisuuksia sovelluksessa on. (1.) (4.)

Vaatimusten keräämisen ohella niitä täytyy myös analysoida. Vaatimuksien analysointivaiheessa selvitetään, ovatko vaatimukset mahdollisia ja taloudellisesti järkeviä toteuttaa. Lisäksi varmennetaan, etteivät vaatimukset ole keskenään ristiriidassa toistensa kanssa ja että vaatimukset huomioivat jokaisen käyttöskenaarion. (4)

Vaatimusten hallinnointi tarkoittaa vaatimuksien muuttumista ja täsmentymistä projektin edetessä. Yleisimpiä syitä muutoksiin ovat:

- Liian tiukan aikataulun takia jokin ominaisuus jätetään toteuttamatta.
- Kilpailija voi julkaista projektin aikana uuden tuotteen ja markkinatilanteen muuttumisen takia reagoidaan lisäämällä vaatimuksia omaan tuotteeseen.
- Ohjelmiston toimintaympäristössä tapahtuu muutoksia.
- Asiakkaan määrittämä ominaisuus osoittautuu turhaksi tai käyttökelvottomaksi. (5)

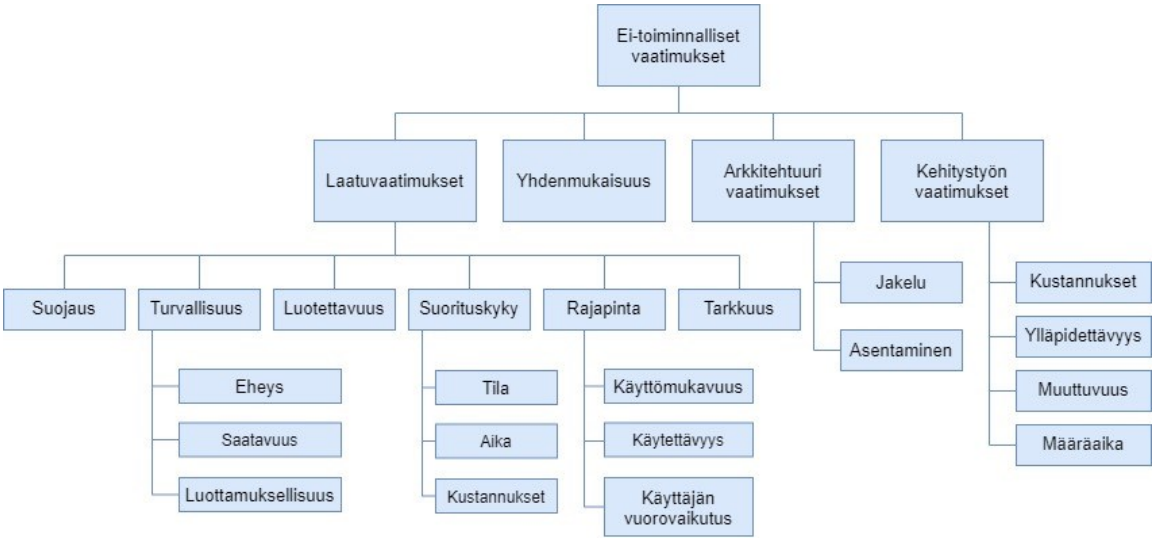
Vaatimusten validointi eli kelpoistaminen on joukko toimenpiteitä, joilla pyritään osoittamaan, että toteutettava järjestelmä vastaa sidosryhmien tarpeita. Validointia voi suorittaa projektin aikana erilaisten prototyyppien avulla. Loppuvaiheessa validointi tapahtuu tuotetestauksen yhteydessä, kun tuote pääsee oikeaan käyttöympäristöönsä. (5)

2.2.2 Toiminnalliset ja ei-toiminnalliset vaatimukset

Vaatimukset voidaan luokitella toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Toiminnalliset vaatimukset kertovat tulevan ohjelmiston palvelut, eli mitä se tekee. Toiminnalliset vaatimukset kuvaavat, mitä toimintoja järjestelmässä on, miten se kommunikoi ympäristönsä kanssa ja millä tavoin eri sidosryhmät ovat yhteydessä järjestelmään. (1.) (4.)

Ei-toiminnalliset vaatimukset ovat puolestaan rajoituksia ja reunaehtoja toiminnallisille vaatimuksille. Ne helpottavat suunnittelu- ja toteutustyön laajuuden mittausta, sillä niiden perusteella määritellään, miten järjestelmän arkkitehtuuri tulee suunnitella. Ne käsittävät koko järjestelmän toimintaa eivätkä ota kantaa yksittäiseen ominaisuuteen. (4.) (6.)

Ei-toiminnalliset vaatimukset voivat olla vaikeasti havaittavia, mutta niistä löytyy erilaisia luokitteluja, jotka helpottavat vaatimuksien tunnistamista. Kuvassa 1 on esitetty Axel van Lamsweerden malli, kuinka ei-toiminnalliset vaatimukset voidaan luokitella. (7)



Kuva 1. Ei-toiminnalliset vaatimukset

2.2.3 Vaatimusten priorisointi MVP-ajattelulla

MVP on lyhenne sanoista Minimum Viable Product eli minimitoiminnallisuus. Sillä tarkoitetaan pienintä määrää toiminnallisuutta, jolla palvelua voidaan käyttää. Vaatimusmäärittelyissä jokaiselle vaatimukselle asetetaan prioriteettinumero:

1 - Pakollinen: Vaatimuksen on oltava mukana uudessa ratkaisussa.

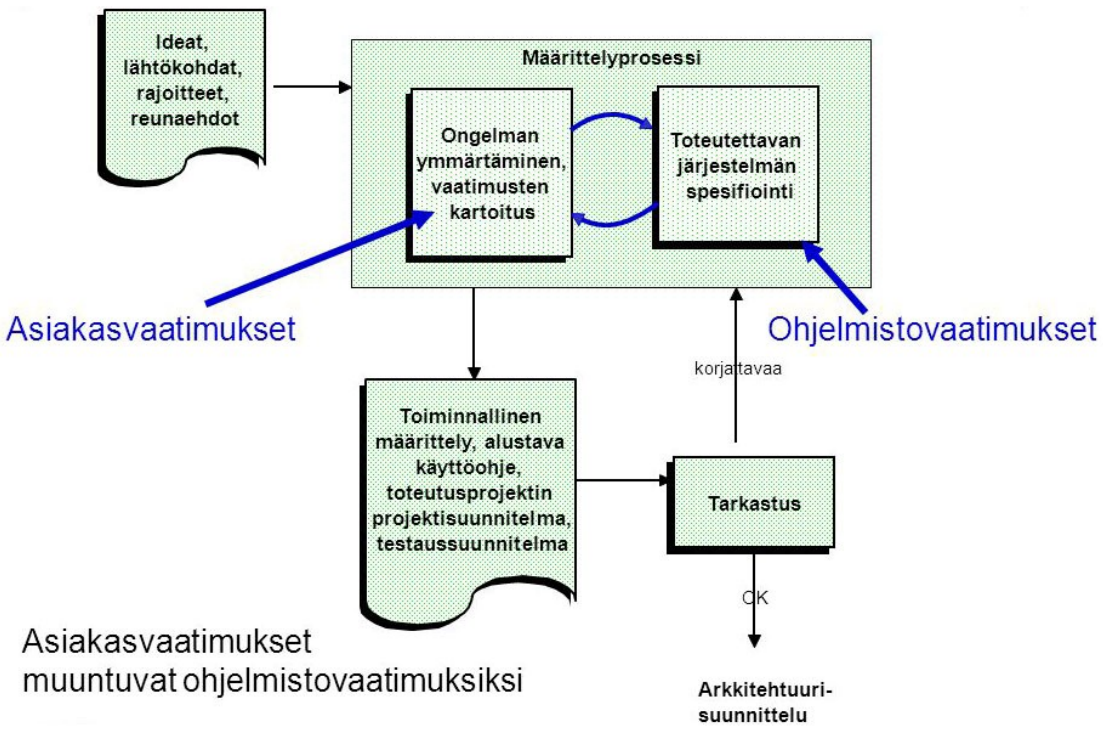
2 - Hyödyllinen: Vaatimuksesta on paljon hyötyä järjestelmälle.

3 - Toivottu: Vaatimus olisi hyvä olla järjestelmässä, mutta se ei ole välttämätön.

Pakolliset vaatimukset eli 1-merkityt vaatimukset muodostavat MVP:n. Järjestelmää ei voida ottaa käyttöön ennen kuin kaikki pakolliset vaatimukset on lisätty järjestelmään. Pitää kuitenkin muistaa, että priorisoinnilla määritellään vaatimusten toteuttamisjärjestys. Alempia prioriteetteja ei saa unohtaa, vaikka järjestelmä toimisikin pienimmällä mahdollisella työpanoksella. Mikäli projektin aikataulu uhkaa ylittyä, voidaan tärkeystasojen 2 ja 3 vaatimusten toteutus siirtää ohjelmiston ylläpitovaiheeseen. (6)

2.3 Järjestelmäanalyysi

Järjestelmäanalyysivaiheessa suoritetaan rakennettavan järjestelmän määrittely eli selvitetään, mitä järjestelmän tulee tehdä. Kun analysoidaan vaatimusmäärittelyvaiheessa tuotetut asiakasvaatimukset, niistä alkaa muodostua järjestelmälle ohjelmistovaatimuksia. Näistä voidaan johtaa järjestelmän toiminnallinen määrittely sekä alustava käyttöohje järjestelmälle. (1) Kuvassa 2 on esitetty järjestelmän määrittelyvaihe.



Kuva 2. Määrittelyvaihe (5)

Järjestelmän toiminnallinen määrittelydokumentti on tärkeä, sillä se toimii pohjana suunnitteluvaiheen teknisille määrittelyille. Määrittelydokumentista tulisi ilmetä ainakin seuraavat asiat:

- Järjestelmän toiminta yleisellä tasolla kuvailtuna
- Jokainen järjestelmän toiminto yksityiskohtaisesti kuvailtuna
- Mikä on järjestelmän tarkoitus
- Millaisessa ympäristössä järjestelmä toimii
- Ketkä käyttävät järjestelmää

- Mitkä ovat järjestelmän rajoitteet
- Kuinka järjestelmä otetaan käyttöön ja millaisia riippuvuuksia sillä on
- Mitä tietokantaa järjestelmä käyttää
- Millaista tietoa järjestelmä käsittelee
- Järjestelmän rajapintojen kuvaukset
- Kuvaukset järjestelmän rajoitteista (1)

2.4 Riskianalyysi

Riskianalyysin tehtävä on hallita riskejä mahdollisimman tehokkaasti. Kehittäjäorganisaatio hyötyy riskianalyysistä, sillä se antaa avaimet riskien hallintaratkaisuihin, joita voi olla muuten vaikea selvittää. Se myös tarjoaa objektiivista tietoa johtoportaalle, joka päättää sovelluskehityksestä. Lisäksi sen avulla voidaan varmistaa, että lainsäädännölliset vaatimukset täyttyvät. Riskianalyysi on kuitenkin vain yksi vaihe riskin arvioinnista ja riskienhallintaprosessista (kuva 3). Riskianalyysi rakentuu riskianalyysin rajojen määrittelystä, vaarojen tunnistamisesta ja riskien arvioinnista. (8)

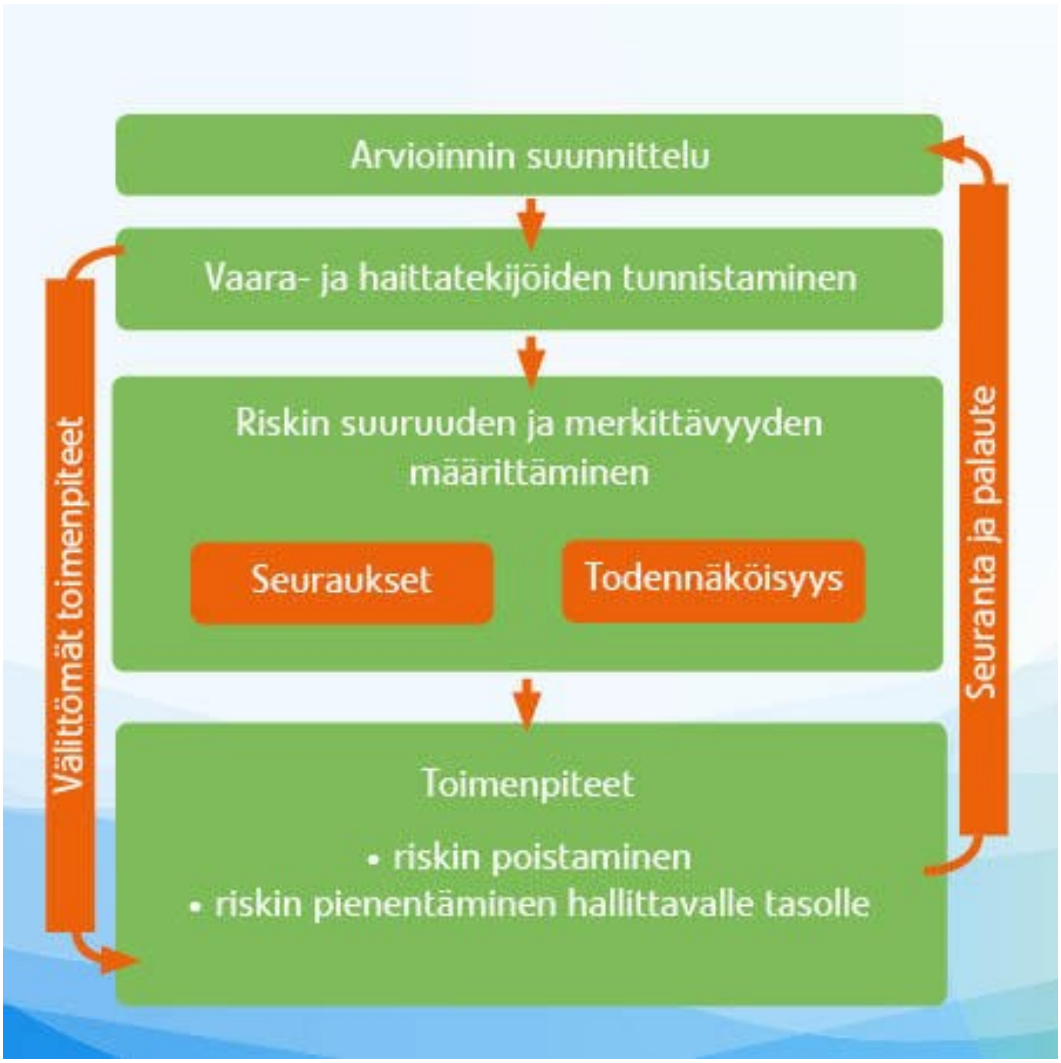
Riskianalyysissä ensimmäinen vaihe on analyysin rajojen määrittely. Tämä sisältää kuvaukset seuraaviin asioihin:

- Miksi riskianalyysi käynnistettiin?
    - Käsittäen riskianalyysin tavoitteet, eli mitkä tunnistetut huolenaiheet projektissa johtivat riskianalyysin aloittamiseen
    - Järjestelmän onnistumis- ja vikaantumiskriteerienmäärittäminen
- Analysoitavan järjestelmän määritys (tähän kannattaa käyttää edellisessä luvussa esiteltyä järjestelmän toiminnallista määrittelyä).
- Kaikkien mahdollisten lähteiden tunnistus, joista saadaan tietoa järjestelmän teknisistä, ympäristöön liittyvistä ja lainsäädäntöön liittyvistä asioista. Myös inhimilliset ja organisatoriset asiat, joita voidaan käyttää riskianalyysivaiheessa.

- Analyysin rajoitukset ja oletukset
- Päätöksen tekijät ja tutkimukselta vaadittavat tulokset (8)

Seuraavaksi tunnistetaan ja dokumentoidaan kaikki mahdolliset järjestelmän vaarat eli pyritään tunnistamaan kaikki haitalliset seuraukset, joita järjestelmä voi aiheuttaa tai mitä sille voi tapahtua. Tähän kannattaa käyttää apuna monien alojen asiantuntijoita, sillä heillä on todennäköisesti erilaisia näkemyksiä järjestelmän vaaroista. (9)

Tämän jälkeen arvioidaan riskien suuruudet ja määritetään niiden taajuus. Vaarat muutetaan riskeiksi asettamalla ne riskimatriisiin. Matriiseja on monia erilaisia, ja tiettyyn analyysiin sopivin riippuu tilanteesta ja projektista. Riskille annetaan ensin suuruusarvio, eli miten suurta vahinkoa riski voi tapahtuessaan tuottaa. Tämän jälkeen riskille tuotetaan taajuusanalyysi. Taajuusanalyysissä arvioidaan kunkin vaaran tapahtumisen todennäköisyyttä. Tapahtumataajuuksien arvioinnissa voidaan käyttää apuna historiatietoja, asiantuntijoiden arvioita tai erilaisia analytiikka- ja simulaatiotekniikoita. Kaikki kolme tapaa ovat valideja arviointitapoja yksikseenkin, mutta kun käyttää kaikkia kolmea tapaa yhdessä, jokaisen keinon heikot puolet korvaantuvat toisten vahvuuksilla. Kun riskit on dokumentoitu riittävällä tarkkuudella, päättäjät tekevät päätökset riskien poistamiseen, hallintaan tai sivuuttamiseen. (8)



Kuva 3. Riskien arvioinnin vaiheet (10)

2.5 Suunnittelu

Suunnitteluvaiheessa tuotetaan järjestelmän toteutussuunnitelma, asiakkaan tarpeet huomioon ottaen. Tässä vaiheessa toiminnallinen määrittely muunnetaan tekniseksi määrittelyksi, josta käy ilmi, miten järjestelmä rakennetaan. (1)

Suunnitteluprosessi voidaan jakaa kahteen osa-alueeseen: arkkitehtuurisuunnitteluun ja moduulisuunnitteluun. Arkkitehtuurisuunnittelu on järjestelmän yleisen rakenteen määrittämistä ja sen osittamista pienempiin osiin eli moduuleihin. Moduuli on kokonaisuus, jossa on määritetty järjestelmän tietyn osakokonaisuuden toiminnot sekä rajapinnat. Moduulien tulisi olla niin pieniä, että

yksittäiset kehittäjät voivat ne erikseen suunnitella ja toteuttaa. Moduulien väliset kytkennät tulee harkita tarkkaan, sillä kytkentöjen määrä on suoraan verrannollinen järjestelmän testattavuuden ja ylläpidettävyyden vaikeuteen. Kytkentöjä voidaan karsia abstraktoimalla ja informaatiota piilottamalla. Näin moduuleista saadaan toisistaan riippumattomia, mikä tarkoittaa, että jokainen moduuli tekee yksin jonkin pienen osakokonaisuuden, eikä se tarvitse siihen muiden moduulien apua. (1.) (11.)

Moduulisuunnittelu on puolestaan moduulien sisäisten rakenteen suunnittelua. Pienet moduulikoot ovat avainasemassa, sillä suuria moduuleja tehtäessä ylläpito vaikeutuu huomattavasti. Tämän lisäksi moduulin suunnittelussa tulee ottaa huomioon sen alimoduulien määrää, joita saa olla maksimissaan kuusi kappaletta. (1)

Teknisen määrittelyn dokumenttiin tehdään tiivistelmä järjestelmän tarkoituksesta. Lisäksi siinä tulee olla kuvaukset ainakin seuraaviin aiheisiin:

- Järjestelmän sovellusalue ja järjestelmän osuus siinä
- Järjestelmän arkkitehtuuri, sisältäen yleiset ratkaisuperiaatteet
- Ohjelmistojen ja tietokantojen arkkitehtuurit
- Laitteisto- ja ohjelmistoympäristö
- Järjestelmän moduulit ja alijärjestelmät
- Kaikki mahdolliset ratkaisukeinot (sekä hyväksytyt, että hylätyt)
- Toteutukseen vaikuttavat asiat (1)

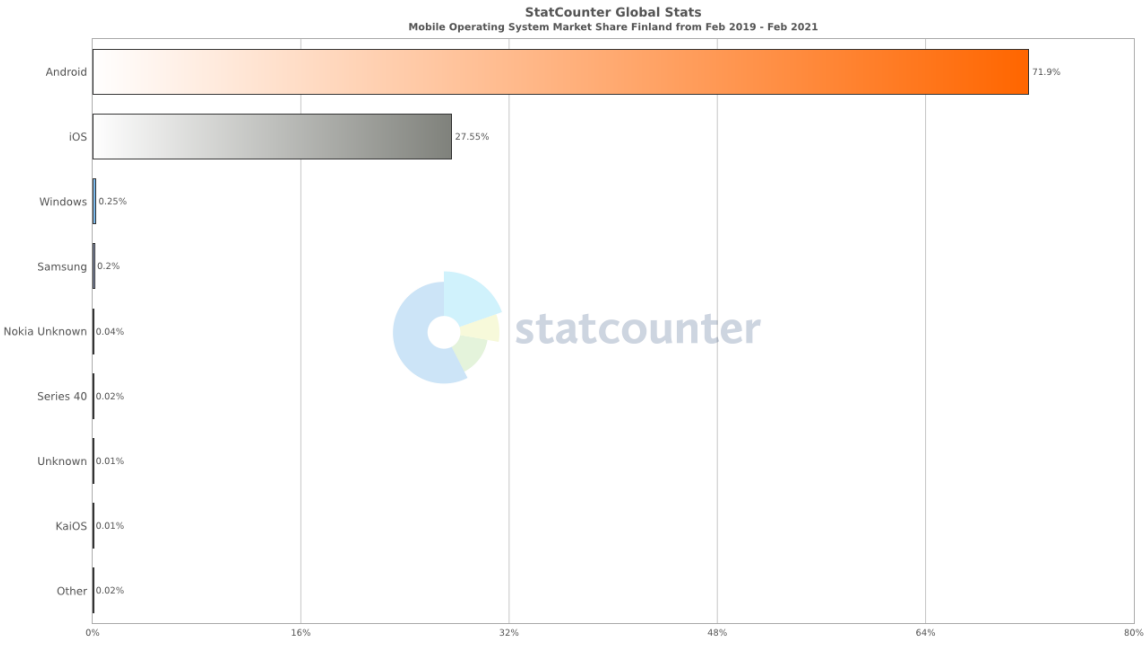
3 Teknologiatutkimus

3.1 Käyttöjärjestelmät

Puhelimista suurin osa käyttää käyttöjärjestelmänään joko Googlen kehittämää Androidia tai Applen iOS:a.

Android on avoimen lähdekoodin käyttöjärjestelmä, mikä tarkoittaa, että monet laitevalmistajat voivat käyttää Androidia laitteissaan. Google myy vain muutamia omia laitteitaan, mutta monet Android-käyttäjät käyttävät muiden yritysten, kuten Samsungin, LG:n tai HTC:n valmistamia laitteita. Tämän vuoksi Androidia, pidetään usein avoimempana alustana verrattuna Appleen. (12)

Applen kehittämää iOS:a käytetään vain heidän omissa laitteissaan, iPhoneissa ja iPadeissa. Apple-universumissa valmistaja hallitsee sekä laitteistoa, että ohjelmistoa. Tämän vuoksi Apple voi valvoa tarkemmin laitteidensa ja iOS App Storesta ladattujen mobiilisovellusten toimintaa. Tämä antaa heille mahdollisuuden ylläpitää uskollista käyttäjäkantaa ja vankkaa markkinaosuutta. (12)



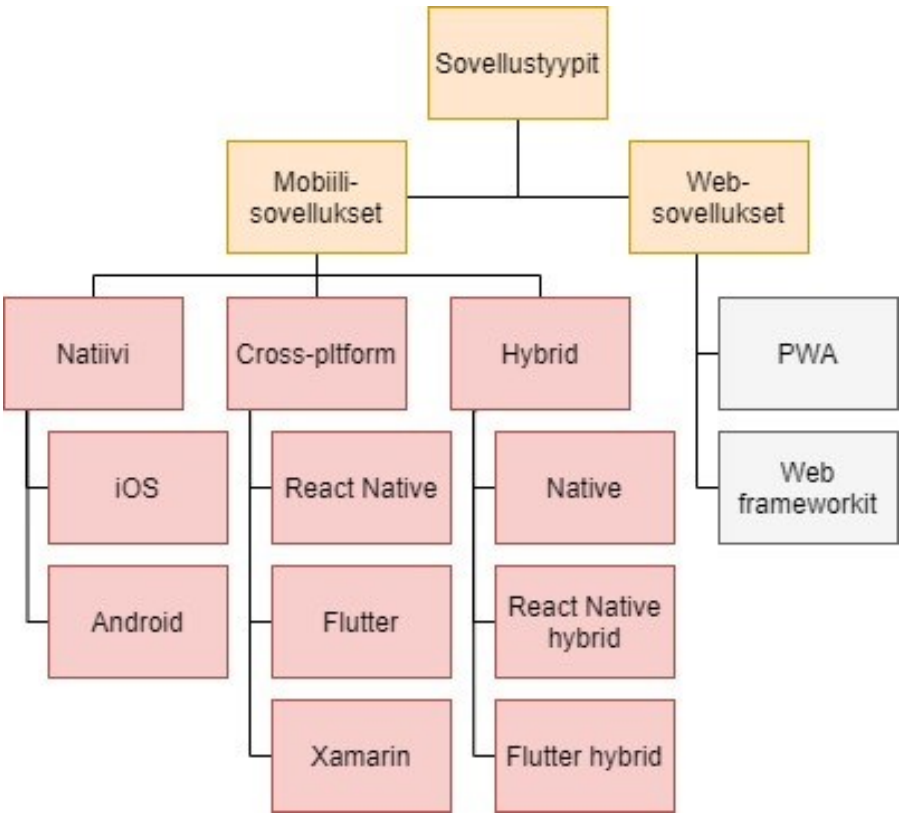
Kuva 4. Käyttöjärjestelmien osuus Suomessa vuosina 2019-2021 (13)

Tarkastelemalla iOS:n ja Androidin markkinaosuuksia (kuva 4) on helppo ymmärtää, että kumpikin näistä alustoista on hyvä ottaa huomioon sovelluskehityksessä, kun rakennetaan sovellusta julkiseen käyttöön. (12)

3.2 Sovellustyypit

Sovelluskehityksessä sovelluksen tyypin valinta on koko kehityksen kannalta oleellisin seikka, sillä se määrittelee työkalut, joilla sovellusta lähdetään rakentamaan. Valinta tehdään teknisen määrittelyn pohjalta.

Sovellustyypit voidaan jakaa aluksi kahteen kategoriaan: mobiilisovelluksiin ja web-sovelluksiin. Mobiilisovellukset kirjoitetaan jollakin ohjelmistokehyksellä ja ne ovat ladattavia sovelluksia, kun taas web-sovellukset ovat vain reagoivia versioita verkkosivustoista, jotka toimivat millä tahansa mobiililaitteella tai käyttöjärjestelmällä, jossa on verkkoselain. Mobiiliapplikaatiot jaetaan vielä kolmeen eri sovelluskategoriaan: natiivi-, cross-platform- ja hybridisovelluksiin (kuva 5). Natiivisovellukset on suunniteltu erityisesti mobiililaitteen omaa käyttöjärjestelmää varten ja niillä saa käyttöön kaikki laitteen ominaisuudet. Cross-platform- ja hybridisovellusten erot ovat hyvin pieniä ja ne eroavat toisistaan lähinnä vain toteutustekniikoiltaan. Kummallekin tekniikalle ominaista on kuitenkin alustariippumattomuus, eli kehitettyä sovellusta voidaan käyttää kaikilla mahdollisilla käyttöjärjestelmällä yhdellä pohjakoodilla. (14.) (15.)



Kuva 5. Sovellustyypit

3.2.1 Natiivisovellus

Kun puhutaan natiivisovelluksista, viittaan yleensä sovelluksiin, jotka on kirjoitettu täysin alustan omilla ohjelmointikielillä ja ohjelmistokehityspaketeilla. Natiivi iOS-sovellukset kirjoitetaan Swift- tai Objective-C-tiedostoilla ja natiivi Android-sovellukset Java ja Kotlin-kielillä. (11)

Natiivisovelluksilla on monia etuja tavallisiin verkkoratkaisuihin nähden, sillä ne ovat nopeampia ja luotettavampia suorituskyvyltään. Natiivisovellukset käyttävät laitteiden natiivikäyttöliittymää, mikä antaa käyttäjälle optimoidun asiakaskokemuksen. Käyttöliittymän ansiosta natiivisovelluksissa voidaan hyödyntää erilaisia sormikomentoja kuten kaksoiskosketusta, raahausta, swaippausta ja monia muita komentoja. Koska natiivisovellukset muodostavat yhteyden suoraan laitteen laitteistoon, ne voivat käyttää kaikkia laitteen ominaisuuksia esimerkiksi bluetoothia ja puhelimen kameraa. Natiivisovelluksia voidaan käyttää ilman internet-yhteyttä, mutta ne täytyy kuitenkin ladata ensin sovelluskaupasta. (12.) (14.)

Natiivisovellusten ongelmana on, että koodi täytyy aina kirjoittaa uudelleen kullekin eri alustalle. Koodia, joka on luotu yhdelle alustalle, ei voida käyttää uudelleen toisessa. Tämä nostaa kustannuksia ja lisää työtä, jota tarvitaan kunkin version koodipohjan ylläpitämiseen ja päivittämiseen. Lisäksi kun sovellus päivitetään, käyttäjän on ladattava uusi ohjelmakoodi ja asennettava se uudelleen. (15)

3.2.2 Cross-platform -sovellus

Cross-platform -kehityksessä sovellus rakennetaan jollakin ohjelmistokehyksellä, joka pyrkii tekemään sovelluksesta mahdollisimman natiivin näköisen. Tämä tapahtuu siten, että kehys luo rajapinnan mobiililaitteen käyttöjärjestelmän natiivielementtien kanssa. Kehys käyttää yleensä jotain omaa ohjelmointikieltä, joka eroaa laitteiden natiiviohjelmointikielestä. Toisin kuin hybridisovellukset, jotka käyttävät verkkotekniikoita toimiakseen, cross-platform -sovelluksissa ohjelmistokehykset tarjoavat käytettävät komponentit sovellukselle. (16)

Cross-platform -kehityksen avulla voidaan rakentaa sovellus ja ottaa se käyttöön useilla eri alustoilla. Tämä tarkoittaa, että sovellus voidaan kohdistaa molempiin sekä iOS- että Android-alustoihin ja näin maksimoi tuotteen saatavuuden. Cross-platform -kehityksen ansioista sovellusta ei tarvitse tehdä uudestaan toisille alustoille, mikä vähentää koodin tuottamiseen kuluvaa aikaa ja kustannukset laskevat. Koska sovellus pyörii vain yhdellä ohjelmakoodilla, sen päivittäminen on

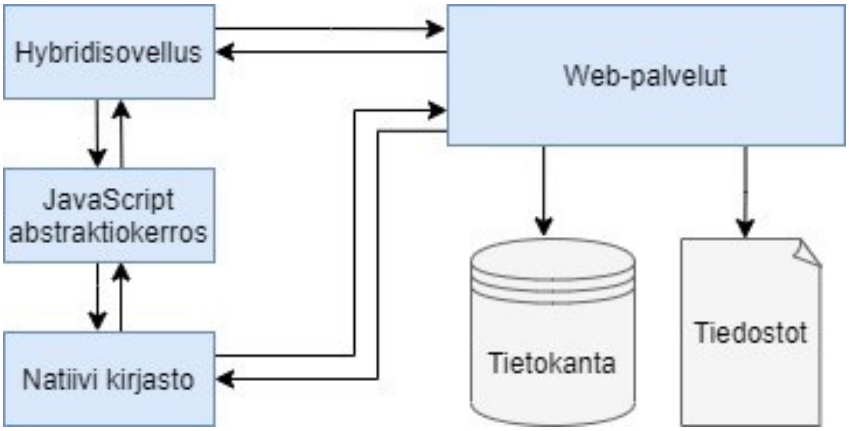
helpompaa ja nopeampaa verrattuna natiivikehitykseen. Käyttämällä cross-paltform -kehystä, sovellus pääsee käsiksi suurimpaan osaan laitteiden laitteistosta, riippuen valitusta kehyksestä. (17)

Cross-platformissa on kuitenkin omat ongelmakohtansa. Valittuun kehykseen luottaminen tarkoittaa sitä, että kehitys tulee riippuvaiseksi kehyksen omasta kehityksestä, käyttöjärjestelmän lisäksi. Tämä lisää tuotteen yleistä monimutkaisuutta ja mahdollisesti hidastaa kehitystä muilla tavoilla, kuten tekemällä virheenkorjauksesta haastavampaa ja työläämpää. Vaikka suorituskyky cross-platform -sovelluksissa on suhteellisen hyvä, niin joissakin laitteissa saattaa esiintyä yleistä hitautta tai muita yhteensopivuus ongelmia voi ilmaantua. Näiden lisäksi cross-platform -kehykset eivät välttämättä tarjoa pääsyä laitteiden jokaiseen ominaisuuteen, mikä voi tehdä joidenkin sovellusten toteutuksesta mahdotonta. (14.) (17.)

3.2.3 Hybridisovellus

Hybridisovellusten idea on yhdistellä osia mobiiliverkkosivustosta ja esittää ne ladatun sovelluksen sisällä. Yleinen termi tällaiselle lähestymistavalle on verkkonäkymä. Hybridisovelluksen luomiseen käytetyn HTML:n ja JavaScriptin renderöi ja käsittelee WebKit-renderöintimoottori Androidissa ja iOS-järjestelmässä UIWebView. Verkkonäkymän ansiosta sovellusta voidaan käyttää kaikilla laitteilla alustaan katsomatta. (14)

Myös hybridisovellukset voivat käyttää joitakin laitteen laitteisto-ominaisuuksia. Hybridisovellukset pääsevät käsiksi laitteen natiiviominaisuuksiin JavaScript-abstraktiokerroksen kautta. Tällaista rajapintaa ei voi toteuttaa web-sovelluksessa, joten tämä tekee hybridisovelluksesta erittäin houkuttavan vaihtoehdon web-sovelluksen tilalle. Hybridi lähestymistapa voi täten hyödyntää sekä selainmoottorin, että osaa laitteen ominaisuuksista (kuva 6). Myös suurin osa cross-platform-sovelluksistakin käyttää JavaScriptin tarjoamaa mahdollisuutta päästä käsiksi natiiveihin elementteihin. (18)



Kuva 6. Hybridisovelluksetkin voivat käyttää natiivikirjastoja

3.2.4 Web-sovellus

Web-sovellukset eli verkkosovellukset käyttäytyvät samalla tavalla kuin natiivisovellukset, mutta niihin pääsee vain mobiililaitteen verkkoselaimen kautta. Ne eivät ole erillisiä sovelluksia, koska niitä ei tarvitse erikseen ladata mobiililaitteelle, vaan ne sijaitsevat jollakin verkkopalvelimella. Ne ovat reagoivia verkkosivustoja, jotka mukauttavat avautuessaan laitteen käyttöliittymään. Verkkosovellukset toimivat siis yhdellä koodipohjalla kaikilla mahdollisilla laitteilla, joilla on pääsy internettiin. Verkkosovellukset on tehty HTML5, CSS, JavaScript, Ruby ja muilla vastaavanlaisilla ohjelmointikielillä, joita käytetään verkkosivujen rakentamiseen. (15)

Verkkosovellusten eri toiminnallisuuksiin liittyy kuitenkin useita rajoituksia. Ensinnäkin web-sovellukset toimivat vain, mikäli laitteella on pääsy internettiin. Tämän lisäksi laitteiston ominaisuuksiin on erittäin rajalliset käyttömahdollisuudet. Myös turvallisen offline-tallennustilan ja istuntojen hallinta on erittäin haastavaa. Täytyy myös muistaa, että eri verkkoselainten välillä voi olla eroja, jotka johtavat toisistaan eroaviin käyttäjäkokemuksiin. (15.) (16.)

3.3 Ohjelmistokehykset

Ohjelmistokehys toimii runkona rakennettavalle sovellukselle. Kehyksen päälle lisätään osia, joista ohjelmisto rakentuu. Ohjelmistokehys on ohjelmoinnin apuväline, jonka on tarkoitus nopeuttaa uusien sovellusten valmistusta siten, että kerran kirjoitettuja osia ei enää tarvitse uudelleenkirjoittaa tuotannon aikana. Kehys tarjoaa erilaisia työkaluja, kuten valmiita koodikirjastoja, kääntäjiä ja rajapintoja sovelluksen kehittämiseen. (19)

Kaikki tässä opinnäytetyössä esitetyt ohjelmistokehykset ovat cross-platform -kehitykseen soveltuvia ja avoimeen lähdekoodiin pohjautuvia kehyksiä.

3.3.1 React Native

React Native on JavaScript-pohjainen mobiilisovelluskehys, jolla voi rakentaa natiivisiti renderöityviä mobiilisovelluksia sekä Androidille että iOs:lle. React Native on Facebookin julkaisema ohjelmistokehys. Ensimmäisen kerran se esitettiin yleisölle vuonna 2015 ja vain muutama vuosi myöhemmin siitä tuli yksi suosituimmista ohjelmistokehyksistä. Menestys johtui siitä, että React-JavaScript-kirjasto oli jo valmiiksi kovassa suosiossa, kun mobiilikehys julkaistiin. Tämän lisäksi käyttöliittymäkehittäjät, jotka olivat käyttäneet ennen vain verkkotekniikoita, pystyivät nyt kehittämään kokonaisia mobiilisovelluksia. (20)

React Native saatetaan helposti sekoittaa alkuperäiseen Reactiin. React Native ei ole kuitenkaan Reactin uudempi versio, vaikka React Native käyttääkin React kirjastoja toimiakseen. React ja React Native käyttävät kummatkin JavaScriptiä ja JSX-merkintäkieltä. JSX eli ”JavaScript XML” on JavaScript syntaksi, jonka avulla voidaan kirjoittaa HTML-elementtejä Reactissa. Näiden kahden syntaksit eroavat toisistaan juurikin JSX-komponenttien renderöinnissä. Siinä missä React käyttää jonkin verran HTML- ja CSS-tiedostoja, React Native sallii vain natiivien mobiilikäyttöliittymäelementtien käytön. (20.) (21.)

React Native kehyksen hienoin ominaisuus on kaksisuuntaisen ja asynkroninen kommunikaatio JavaScript-pohjaisten säikeiden ja natiivisäikeiden välillä. Nämä säikeet kirjoitetaan täysin eri ohjelmointikielillä, mutta React Native käyttää eräänlaista ”siltaa” näiden kahden keskeiseen kommunikaatioon (kuva 7). Koska silta on rakennettu C/C++ kielillä, sitä voidaan käyttää useilla alustoilla. (20)

Kuva 7. React Nativen kommunikaatio JavaScriptin ja natiivisäikeiden välillä

React Nativen hyvät puolet:

- Suuri kehittäjäyhteisö
    - Apu kehitystyöhön on helposti saatavilla.
- Nopea päivitys
    - Kehittäjät voivat käyttää sovellusta kehitystyön aikana ja muokata sitä rakentamatta sovellusta uudelleen jokaisen päivityksen jälkeen.
- Yksinkertainen käyttöliittymä
    - React Native käyttää React-JavaScriptiä rakentamaan sovelluksen käyttöliittymän. Se tekee sovelluksesta responsiivisen ja nopeasti reagoivan, erittäin pienellä latausajalla.
- Ajan kestävä ratkaisu
    - Suuri yhteisö ja yksinkertainen lähestymistapa ongelmien ratkaisuun takaa React Nativen kirkkaat tulevaisuuden näkymät. (20)

React Nativen heikkoudet:

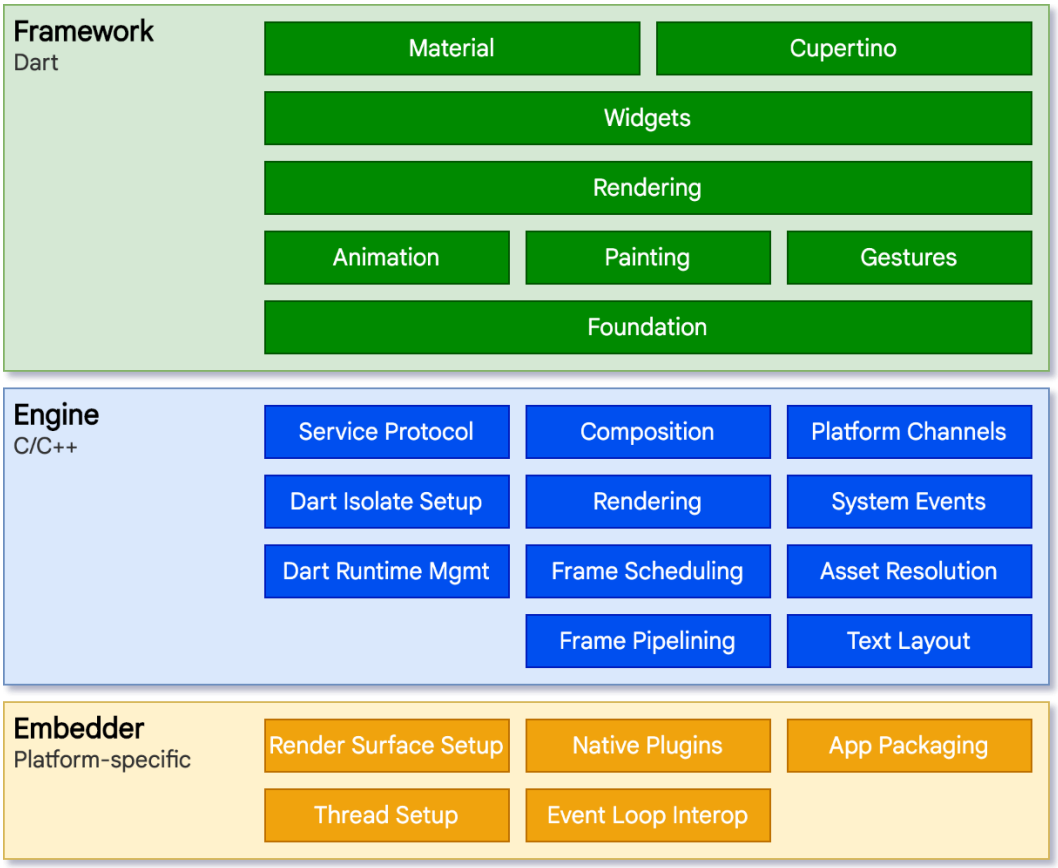
- Tarvitaan todennäköisesti natiivikehittäjän apua
- Muutamien mukautettujen moduulien puute
- Jotkin yhteensopivuus ongelmat
- Skaalattavuus (20)

3.3.2 Flutter

Flutter on Googlen luoma mobiilisovelluskehys. Flutter esiteltiin ensimmäisen kerran toukokuussa 2017, mutta ensimmäinen stabiiliversio julkaistiin vasta joulukuussa 2018. Flutterin idea on Dart-koodin kääntäminen natiivikoodiksi, joka toimii kohdelaitteessa. Flutter siis käyttää Dart-ohjelmointikieltä, jonka Google suunnitteli alun perin korvaamaan JavaScriptin omissa palveluissaan. Dart-ohjelmointikieli on peräisin vuodelta 2011, mutta vuosien varrella sitä on paranneltu paljon. Flutter on ainoa cross-platform -kehys, joka toimii ilman Javascript siltaa. (22.) (23.)

Flutterin avulla kehittäjä voi rakentaa koko käyttöliittymän yksinkertaisesti yhdistämällä erilaisia widgettejä toisiinsa. Widgettejä muuntelemalla voidaan helposti luoda asiakastarpeisiin mukautuvia käyttöliittymiä. Widgettejä ovat kaikki sovelluksen komponentit, aina napeista tekstiin asti. Flutterin mukana ei tule käyttöjärjestelmien alkuperäisiä natiiviwidgettejä, mutta Flutter tarjoaa omat widgetit jotka näyttävät hyvin samanlaisilta kuin natiivielementit. (23)

Flutter-kehys rakentuu eri tasoista ja ne on luokiteltu erilaisiin kategorioihin, perustuen niiden esiintymisjärjestykseen (kuva 8). Ylin kategoria on sovelluksen käyttöliittymä, joka on Android- ja iOS-alustoille erilainen. Tämän tason rakentamiseen käytetään Dart-ohjelmointikieltä. Se sisältää sovelluksen kaikki materiaalit ja widgetit, sekä niiden renderöintiin tarvittavat kirjastot. Toisessa kategoriassa on Flutter-moottori, joka on kirjoitettu C/C++ kielillä. Moottorin toteutustavan ansiosta se on nopeampi kuin muut mobiiliohjelmointiin tarkoitetut ohjelmistokehykset. Alin kerros on niin kutsuttu upottaja kerros. Upottaja on kirjoitettu niin, että jokaiselle alustalle on juuri sopivalla kieli sitä varten. Alustakohtainen upotin tarjoaa rajapinnan sovelluksen ja laitteen välille. Näin sovellus voi käyttää laitteen ominaisuuksia hyödykseen. (24)



Kuva 8. Flutterin arkkitehtuuri (21)

Flutterin hyvät puolet:

- Kehitys vaatii vain vähän resursseja
    - Jo yhdellä kehittäjällä voi saada aikaan sovelluksen, eikä kokonaista kehittäjätiimiä välttämättä tarvita.
- Nopea kehitettävä
    - Flutter-sovellukset kehitetään virtuaalikoneessa, mikä mahdollistaa muutosten tekemisen sovellukseen ilman uudelleenkäännöstä.
- Kaunis käyttöliittymä
    - Skia grafiikkamoottorilla saadaan aikaan näyttäviä käyttöliittymiä. Skia on avoimenlähdekoodin grafiikkamoottori ja sitä käyttävät Adobe, Chrome ja Amazon Kindle.
- Laaja ja laadukas dokumentaatio
    - Hyvin luokitellut dokumentit ja helposti seurattavat esimerkit. (24)

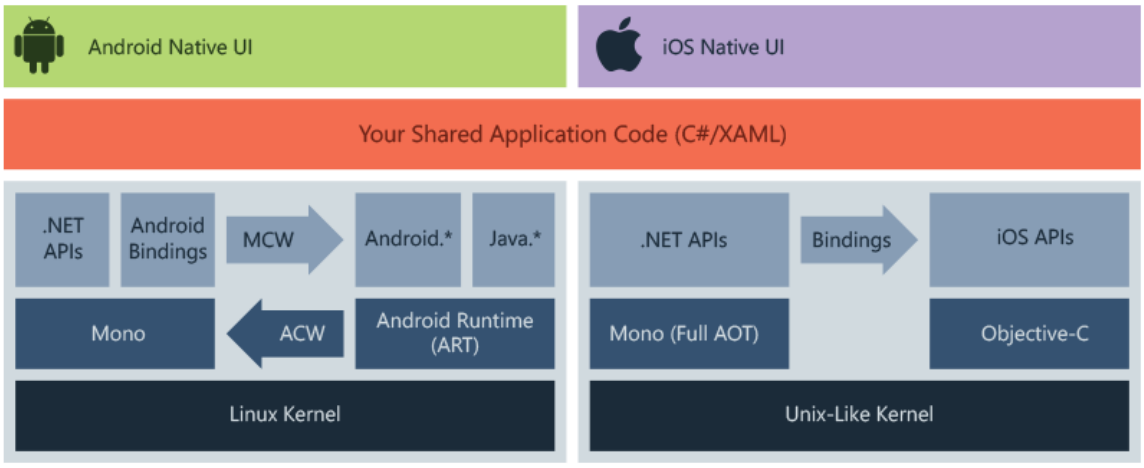
Flutterin heikkoudet:

- Kolmannen osapuolten kirjastojen puute
- Suuret tiedosto koot
- Dart ohjelmointikielen opettelu (24)

3.3.3 Xamarin

Xamarin on Microsoftin tarjoama ohjelmistokehys. Xamarinin kehitti Mono-ohjelmistokehyksen luoneet insinöörit vuonna 2011. Microsoft kuitenkin osti Xamarinin itselleen vuonna 2016 ja teki siitä avoimen lähdekoodin ratkaisun.

Kuva 9 esittää alustojen välisen Xamarin-sovelluksen kokonaisarkkitehtuurin. Useimmissa tapauksissa 80% sovelluskoodista toimii suoraan molemmilla alustoilla Xamarinin avulla. Xamarin käyttää C#- ja natiivikirjastoja, jotka on kääritty .Net-kerrokseen alustojen välisessä sovelluskehityksessä. Koska C# on yksi .NET-kehyskielistä, sitä voidaan käyttää useiden hyödyllisten .NET-ominaisuuksien kanssa. .NET pystyy hoitamaan automaattisesti muistin jakamisen, roskien keräämisen ja yhteen toimivuuden taustalla olevien alustojen kanssa. C#:n ansiosta koodissa ei myöskään esiinny odottamatonta käytöstä. (25)



Kuva 9 Xamarin-sovelluksen arkkitehtuuri (24)

Xamarinin hyvät puolet:

- Täysi laitteistotuki
    - Xamarinin avulla saadaan käyttöön natiivitason sovellustoiminnot. Se käyttää erilaisia laajennuksia ja sovellusliittymiä poistamaan kaikki laitteiden yhteensopivuusongelmat.
- Paljon kehitystyökaluja
    - Xamarinin mukana tulee täydellinen kehitystyökalusarja, mukaan lukien oma IDE. Xamarin-sovellusten rakentamiseen, testaamiseen ja käyttöönottoon ei tarvita lisätyökaluja tai integroida kolmansien osapuolten sovelluksia.
- ”Hot reload”
    - Aivan kuten flutterissa, myös Xamarinissa on koodin muutoksia reaaliaikaisesti esittävä ominaisuus, jonka avulla koodin uudelleenrakennus kertoja voidaan vähentää.
- Paljon opetusmateriaalia
    - Microsoft ylläpitää Microsoft Learn on online-koulutusportaalia, josta löytyy koulutusmateriaalia myös Xamarin ohjelmistokehykseen. (26)

Xamarinin heikkoudet:

- Rajoitettu pääsy avoimen lähdekoodin kirjastoihin
- Korkeat kustannukset yrityskäyttöön
- Sovellusten suuri koko (26)

3.4 Vertailu

Vertailun tarkoitus oli päättää, mikä ohjelmistokehys mobiilisovelluksen rakentamiseen valitaan. Kaikki vertailuun valitut kehykset käyttävät cross-platform -teknologiaa. Aluksi selvitettiin, että kaikilla kehyksillä on varmasti mahdollisuus käyttää mobiililaitteen laitteisto-ominaisuuksia riittävällä tasolla. Reunaehdot laitteisto-ominaisuuksille löytyivät vaatimusmäärittelyistä. Tämän jälkeen ohjelmistokehyksien välillä suoritettiin vertailu, jonka perusteella sovellukseen käytettävä ohjelmistokehys valikoitui. Vertailussa otettiin huomioon esimerkiksi mitä koodikieltä kehys käyttää, kehyksen dokumentaation määrä ja laajuus, sekä kehyksen suosio.

4 Pohdinta

Työn tilaajan vähäinen tieto mobiilikehittämisestä antoi suuren määrän vastuuta tutkimuksen oikeellisuudesta, sillä tutkimuksen perusteella Prometec päättää aloitetaanko mobiilikehitys vai ei. Idea mobiilisovelluksen tuottamisesta tuli siis asiakkaalta yksittäisenä projektina, mutta tutkimuksen perusteella Prometec voisi halutessaan aloittaa kokonaan uuden tuotekehityshaaran, joka keskittyisi täysin mobiilikehittämiseen.

Hain tutkimukseeni tietoa sekä kirjallisuudesta, että internetistä. Standardit opinnäytetyötäni varten hankki Kajaanin ammattikorkeakoulu. Usein tutkimusta tehdessäni huomasin, että eri lähteissä viitataan samaan asiaan hieman eri termeillä. Tämä teki työstä haastavaa, sillä termistö oli erittäin keskeinen asia, niin ohjelmistokehityksen kuin teknologiatutkimuksen osalta.

Ohjelmistokehityksen vaiheisiin löytyi paljon tietoa suomen kielellä, joten tiedon hakeminen oli tähän osioon helppoa. Kun vertailin netistä löytämiäni artikkeleita yli 15 vuotta vanhaan kirjallisuuteen, ei eroja tekniikoissa juurikaan löytynyt. Tämä todisti sen, että vaikka lähdetieto onkin vanhaa, se voi vielä tänäkin päivänä olla validia. Erilaisten kehitysteknologioiden tutkiminen on taas aivan toinen ääripää, jossa pitää jatkuvasti pysyä ajan hermoilla. Jos tieto on yli 2 vuotta vanhaa, se voi olla jo hyvinkin vanhentunutta. Nopea kehitys eri teknologioissa voi tuottaa virheellistä tutkimustulosta, ellei julkaisuajankohtaa ole otettu huomioon.

Opinnäytetyö toimii itsessään osana tutkimustulosta. Opinnäytetyön avulla selvisi, millaisia dokumentteja tarvitaan projektin onnistuneeseen läpivientiin. Teknologiatutkimuksen avulla saatiin selville ohjelmistokehyksien hyviä ja huonoja ominaisuuksia. Kun kehyksiä verrattiin keskenään erilaisin mittarein, saatiin Prometecille valittua sovelluksen kehitykseen sopiva ohjelmistokehys.

Tutkimuksen ansiosta sain paljon tietoa erilaisista ohjelmistokehityksestä, sekä mobiilikehityksen teknologioista. Uskon että näistä kumpikin aihealue on hyödyllinen tulevaisuutta ajatellen, sillä laajempi tietämys ohjelmistosuunnittelusta antaa valmiuksia hoitaa tulevaisuuden projekteja täsmällisemmin ja teknologiatietämys mobiilipuolesta avaa uusia uramahdollisuuksia teknologia-alalla.

5 Yhteenveto

Opinnäytetyöni tavoite oli tuottaa Prometecille tutkimus, jossa selvitettiin, kuinka vaakajärjestelmälle voidaan rakentaa mobiilisovellus mahdollisimman kustannustehokkaasti ja millaisilla teknologioilla se voidaan toteuttaa. Jaoin opinnäytetyön kahteen osaan, ohjelmistokehityksen vaiheisiin ja teknologiatutkimukseen.

Ohjelmistokehityksen vaiheissa käsittelin kuinka sovellus pitää suunnitella ja mitä asioita suunnitteluvaiheissa tulee ottaa huomioon. Teknologiatutkimuksessa esittelin sovellustyypit ja muutaman ohjelmistokehyksen, joilla mobiiliapplikaatio voidaan rakentaa. Lopuksi vertailin ohjelmistokehyksiä keskenään ja valikoin sieltä sopivan kehyksen Prometecin tarpeisiin.

Tutkimuksen tuloksena Prometecille saatiin tuotettua sellaiset dokumentit, joista selviää, miten ohjelmistokehityksen suunnittelu etenee ja millä tekniikalla mobiilisovellus on viisaita toteuttaa. Tulevaisuudessa Prometec voi käyttää opinnäytetyötäni pohjana ohjelmistokehityksen suunnitteluun.

Lähteet

1 Pohjonen R. Tietojärjestelmien kehittäminen. Jyväskylä: Docendo; 2002.

2 Puro.net. Projektien valinta. viitattu 14.04.2021. saatavilla: https://proha.purot.net/projektien_valinta.

3 Kalanen S. Esiselvitys luo pohjan onnistuneelle projektille. viitattu 12.04.2021. saatavilla: https://meteoriitti.com/2012/04/23/esiselvitys-auttaa-onnistumaan/.

4 Luukkainen M. Ohjelmistojen vaatimusmäärittely, tuotteen ja sprintin hallinta. viitattu 14.04.2021. saatavilla: https://ohjelmistotuotanto-hy.github.io/osa2.

5 Haikala I, Märijärvi J. Ohjelmistotuotanto. Helsinki: Talentum; 2006.

6 Helsingin kaupunki. Vaatimukset. viitattu 12.04.2021. saatavilla: https://kehmet.hel.fi/menetelmalaari/vaatimukset/.

7 Paakki J. Ohjelmistojen vaatimusmäärittely. viitattu 15.04.2021. saatavilla: https://www.cs.helsinki.fi/u/paakki/Vaatimus-11-Luentokalvot-4.pdf.

8 SFS-IEC 60300-3-9 (30. 6 2000). Luotettavuusjohtaminen. Osa 3: Käyttöopas. Luku 9: Teknisten järjestelmien riskianalyysi. Suomen standardisoimisliitto SFS. Viitattu 06.05.2021

9 Helsingin kaupunki. Riskianalyysi. viitattu 20.04.2021. saatavilla: https://kehmet.hel.fi/menetelmalaari/riskianalyysi/.

10 Työturvallisuuskeskus. Työturvallisuus- ja työterveysriskien tunnistaminen ja arviointi. viitattu 06.05.2021. saatavilla: https://ttk.fi/tyoturvallisuus_ja_tyosuojelu/tyosuojelu_tyopaikalla/vastuut_ja_velvoitteet/tyon_vaarojen_selvittaminen_ja_arviointi.

11 Helsingin yliopisto. Suunnitteluperiaatteet. viitattu 07.05.2021. saatavilla: https://www.cs.helsinki.fi/u/taina/ohjelmistotuotanto/luennot/k99/suunn1/kaikki.html.

12 Saccomani P. Native Apps, Web Apps or Hybrid Apps? What's the Difference? viitattu 26.04.2021. saatavilla: https://www.mobiloud.com/blog/native-web-or-hybrid-apps.

13 Statcounter. Mobile Operating System Market Share in Finland - April 2021. viitattu 26.04.2021. saatavilla: https://gs.statcounter.com/os-market-share/mobile/finland/#monthly-201902-202102-bar.

14 Liikkanen L A. Native, hybrid or cross-platform? Choosing the right app technology goes a long way towards commercial success. viitattu 26.04.2021. saatavilla: https://qvik.com/news/native-hybrid-or-cross-platform-choosing-the-right-app-technology-commercial-success/.

15 Valdellon L. What Are the Different Types of Mobile Apps? And How Do You Choose? viitattu 26.04.2021. saatavilla: https://clevertap.com/blog/types-of-mobile-apps/.

16 Harrison K. Mobile App Development Options: Hybrid vs. Native vs. Web. viitattu 26.04.2021. saatavilla: https://www.digitaldoughnut.com/articles/2018/october/mobile-app-development-options-hybrid-vs-native/.

17 Manchanda A. Where Do Cross-Platform App Frameworks Stand in 2021? viitattu 27.04.2021. saatavilla: https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/.

18 Khandeparkar A, Gupta R, Sindhya B. An introduction to hybrid platform mobile application development. International Journal of Computer Applications. viitattu 27.04.2021. saatavilla: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.695.5170&rep=rep1&type=pdf.

19 Christina. Ohjelmistokehykset. viitattu 12.05.2021. saatavilla: https://www.edelphi.fi/ohjelmistokehykset/.

20 Budziński M. Netguru What Is React Native and How Is It Used? viitattu 08.05.2021. saatavilla: https://www.netguru.com/what-is-react-native.

21 W3schools. React JSX. viitattu. 08.05.2021. saatavilla: https://www.w3schools.com/react/react_jsx.asp.

22 Thomas G. What is Flutter and Why You Should Learn it in 2020. viitattu 08.05.2021. saatavilla: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/.

23 Nix-united. What is Flutter and Why Use Flutter for App Development. viitattu 08.05.2021. saatavilla: https://nix-united.com/blog/the-pros-and-cons-of-flutter-in-mobile-application-development/.

24 Flutter. Flutter architectural overview. viitattu 08.05.2021. saatavilla: https://flutter.dev/docs/resources/architectural-overview.

25 Microsoft. What is Xamarin? viitattu 10.05.2021. saatavilla: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin.

26 Altexsoft. The Good and The Bad of Xamarin Mobile Development. viitattu 10.05.2021. saatavilla: https://www.altexsoft.com/blog/mobile/pros-and-cons-of-xamarin-vs-native/.