Opinnäytetyö (AMK)

Tietojenkäsittely

2021

Juhani Suominen

TOIMINTAMALLIN VALINTA PIENISSÄ OHJELMISTOALAN PROJEKTEISSA

– suun hyvinvointia edistävän mobiilisovelluksen beta-version kehitys

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2021 | 31 sivua

Juhani Suominen

TOIMINTAMALLIN VALINTA PIENISSÄ OHJELMISTOALAN PROJEKTEISSA

- suun hyvinvointia edistävän mobiilisovelluksen beta-version kehitys

Projektinhallinta on yleisesti hyvinkin laaja käsite, jonka toteutukseen on olemassa valtava määrä erilaisia työkaluja ja toimintamalleja. Tämän opinnäytetyön tavoitteena oli kartoittaa IT-alan projekteissa eniten käytettyjä toimintamalleja ja niiden soveltuvuutta erityisesti opinnäytetyön aikana toteutettavaan suun hyvinvointia parantamaan suunnitellun mobiilisovelluksen beta-version kehitykseen. Mobiilisovelluksen toteuttivat Turun ammattikorkeakoulussa toimivan theFIRMA-oppimisympäristön opiskelijat. Asiakas ja mobiilisovelluksen toimeksiantaja on Turun ammattikorkeakoulun YAMK-opiskelija.

Opinnäytetyössä esiteltiin mobiilisovelluksen kehitysympäristö ja annettiin katsaus yleisimpiin pienissä IT-alan projekteissa käytössä oleviin toimintamalleihin. Opinnäytetyön aikana hyödynnettiin tutkivaa havainnointia. Työn lopussa käytiin läpi projektin toteutus ja pohdittiin tehtyjen havaintojen perusteella mitä toimintamalleja käyttämällä projekti olisi ollut sujuvin.

Opinnäytetyössä selvisi, että mobiilisovelluksen beta-version kehityksessä olisi kannattanut soveltaa alusta alkaen jossakin määrin ketteriä menetelmiä. Näin tiimin resursseja olisi paremmin pystytty seuraamaan ja työtehtävien allokointi olisi pystytty toteuttamaan mahdollisimman tehokkaasti. Jos asiakkaan vaatimukset ja sovelluksen beta-version sisältö olisi jo projektin alkaessa määritelty ja rajattu tarpeeksi tarkasti, olisi kehitystyöhön jäänyt enemmän aikaa ja toteutus ollut hallitumpi.

ASIASANAT:

toimintamalli, projektinhallinta, menetelmät, sovelluskehitys

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2021 | 31 pages

Juhani Suominen

CHOOSING THE RIGHT PROCESS MODEL FOR SMALL SOFTWARE PROJECTS

- development of a beta-version for a dental healthcare mobile application

Project management is generally a very broad concept, for which there are many different process models. The purpose of this thesis was to first map out the most widely used ones in modern software development and then compare them to the mobile application developed for this thesis to see which one would most likely have yielded the best results. The mobile application was developed by students of theFIRMA, a learning environment of Turku University of Applied Sciences. The client was a Master's thesis worker also from Turku University of Applied Sciences.

First, the development environment of the mobile application was introduced, and the key features of the most used process models were showcased. At the end, the execution of the mobile application was evaluated. Based on the observations, the process models were compared to the development of the mobile application to find out which one would have been the most fitting.

By the end it was clear that an agile approach would most likely have resulted in the most successful outcome for the development of this mobile application. By adopting an agile approach, the development teams' resources would have been easier to manage, and the allocation of any given task would have been more efficient. By accurately specifying the requirements of the customer and defining the specific features of the beta-version, the project team would have had more time for the actual development and the execution would have been better controlled.

KEYWORDS:

process model, project management, method, software development

SISÄLTÖ

KUVAT

SANASTO

IT	Informaatioteknologia
Mockup	Verkkosivun tai sovelluksen suunniteltua ulkonäköä kuvastava malli.
UI/UX-suunnittelu	Käyttöliittymän (engl. user interface) ja käyttäjäkokemuksen (engl. user experience) sunnittelu.

1 JOHDANTO

Projektinhallintaan on olemassa monenlaisia työkaluja ja toimintamalleja. Nämä toimintamallit eivät kuitenkaan ole ”one-size-fits-all”-ratkaisuja, vaan ne soveltuvat vaihdellen erilaisiin projekteihin ja tilanteisiin. Jotkin toimintamallit soveltuvat paremmin suuren mittakaavan projekteihin, kun taas toiset ovat optimaalisempia pienempien projektien toteuttamiseen.

Toimintamallin valintaan vaikuttaa monikin asia. Muun muassa yrityksen tai projektitiimin koko, tiimin ja projektijohdon ennestään opitut tottumukset, sekä projektin luonne voivat vaikuttaa siihen, mitä toimintamallia missäkin projektissa olisi tehokkainta hyödyntää.

Opinnäytetyön aikana toteutettiin suun hyvinvointia parantamaan suunnitellun mobiilisovelluksen beta-versio. Asiakas ja mobiilisovelluksen toimeksiantaja oli Turun ammattikorkeakoulun YAMK-opiskelija.

Mobiilisovelluksen kehittivät Turun ammattikorkeakoulussa toimivan theFIRMA-oppimisympäristön opiskelijat. theFIRMA toimii pienen projektitoimiston tavoin, jossa opiskelijat pääsevät työskentelemään oikeissa asiakasprojekteissa jo opintojensa aikana ja kerryttämään opintopisteitä tekemästään työstä. Vaikka projektin toteuttivatkin theFIRMAn opiskelijat, toteutettiin se kuitenkin theFIRMAn ulkopuolisena projektina eikä näin ollen ollut suoraan yhteydessä theFIRMAn toimintaan.

Tässä opinnäytetyössä tavoitteena oli kartoittaa, mitkä toimintamallit nykypäivänä ovat IT-alalla etenkin pienemmissä projekteissa käytetyimpiä sekä verrataan näitä toimintamalleja opinnäytetyön aikana toteutettuun suun hyvinvointia edistävän mobiilisovelluksen beta-version toteutukseen. Lopuksi kartoitettiin, mitä toimintamalleja tai niiden osia kyseisessä projektissa olisi mahdollisesti kannattanut mukailla toteutuksen tehokkuuden maksimoimiseksi.

2 PROJEKTIN KUVAUS

2.1 Kehitystiimi ja ympäristö

Suun hyvinvointia edistävän mobiilisovelluksen beta-version toteutus tapahtui Turun ammattikorkeakoulun oppimisympäristön theFIRMAn opiskelijatyönä. theFIRMA toimii kuten pieni projektitoimisto, jossa opiskelijat suorittavat projektiopintojaan opintopisteitä vastaan. Oppimisympäristön toiminnasta vastaa opiskelijatoimitusjohtaja yhdessä theFIRMAn opettajien ja management-tiimin kanssa.

theFIRMAssa projektiopintojaan suorittaa opiskelijoita monelta eri vuosikurssilta ja näin ollen opiskelijoiden tietotaidossa on myös suurta hajontaa. Vasta-alkajat ja useamman vuoden työkokemuksen omaavat opiskelijat työskentelevät samoissa projekteissa ja toiminnassa pyritään, että kokeneemmat opiskelijat toimisivat kokemattomampien opiskelijoiden tukena projekteissa.

Mobiilisovelluksen kehitykseen valittiin neljä theFIRMAn opiskelijaa. Jokainen projektitiimin jäsen osallistui ohjelmointiin. Projektilla oli projektipäällikkö sekä UI/UX-suunnittelija.

Opinnäytetyön aikana toimin theFIRMAn opiskelijatoimitusjohtajana sekä projektityöryhmän projektipäällikkönä, ja opinnäytetyötä koskevat huomiot ja havainnot tehtiin näistä rooleista käsin.

2.2 Projektin aloitus

Projektin alussa kartoitettiin sopivia teknologioita mobiilisovelluksen toteutukseen. UI/UX-suunnitteluun työkaluksi valittiin Figma (www.figma.com), koska se oli projektitiimille entuudestaan jokseenkin tuttu. Ohjelmointikieleksi valittiin React Native (www.reactnative.dev) asiakkaan tarpeiden ja toiveiden perusteella. Kuten monesti theFIRMAn projekteissa, projektitiimin jäsenillä joko ei ollut ollenkaan tai oli vain minimaalinen tietotaito käytetyistä teknologioista ennen projektin alkua.

Suun hyvinvointi mobiilisovelluksen kehitystyö alkoi intensiivisellä React Native-ohjelmointikielen opiskelujaksolla. Jokainen projektitiimin jäsen opiskeli aihetta

itsenäisesti useamman viikon ajan, jonka aikana pyrittiin myös määrittämään asiakkaan vaatimuksia sovelluksen toimintaan ja ulkoasuun liittyen.

Varsinaiset vaatimukset määriteltiin tässä kohtaa vain karkeasti ja keskityttiin pääasiassa kokonaisuuden suunnitteluun. Sovelluksen sisältöä suunniteltiin siltä osin, mitä ominaisuuksia asiakas sovellukseen kehityksen alkuvaiheessa toivoi. Merkittävä osa näistä toiveista oli selvästi beta-vaiheen rajojen (engl. scope) ulkopuolella, mutta tässä kohtaa sisältöä ei kuitenkaan rajattu.

Sovelluksen visuaalisen ilmeen suunnittelu aloitettiin pian projektin alettua. Sovelluksen visuaalisen ilmeen suunnittelu toteutettiin Figma-ohjelmistolla. Asiakas toimitti projektitiimille useita erilaisia referenssejä sovelluksen visuaalisen ilmeen suunnitteluun. Näiden referenssien, sekä asiakkaan toiveiden pohjalta suunniteltiin alustava pohjapiirros, joka esitettiin asiakkaalle. Asiakkaan hyväksyttyä päänäkymän pohjapiirroksen, luotiin suunnitelmat muista näkymistä sen pohjalta.

2.3 Sovellus pähkinänkuoressa

Sovelluksen tarkoituksena oli edistää aikuisen käyttäjän, sekä tämän lapsen suun hyvinvointia ohjaamalla käyttäjää parantamaan totuttuja päivittäisiä tapojaan, sekä muistuttamalla tätä muun muassa säännöllisestä hampaiden pesusta, hammaslangan käytöstä ja hammaslääkärin tarkastuksessa käynnistä niin itsensä, kuin lapsensakin osalta. Ensisijainen kohderyhmä oli alustavasti tulevat sekä juuri ensimmäisen lapsensa saaneet uudet vanhemmat.

Sovelluksen käyttöönotossa tulisi käyttäjän täyttää alkukysely, jonka perusteella käyttäjän suun terveys, sekä hänen tottumuksensa pisteytettäisiin. Pisteytys tulisi käyttäjän näkyville sovelluksen päänäkymään.

Käyttäjän tottumuksia tultaisiin seuraamaan päivittäisellä tasolla sovellukseen rakennettavan tarkistuslistan avulla, jonka perusteella käyttäjän kokonaispistemäärä päivittyisi tämän toiminnan mukaisesti.

Päänäkymän lisäksi sovellukseen tulisi omat näkymänsä alkukyselylle, muistutuksille, tarkistuslistalle, käyttäjän suun hyvinvointia parantavalle ohjeistukselle sekä jonkin asteiselle asiakkaan määrittämälle terveyskirjastolle.

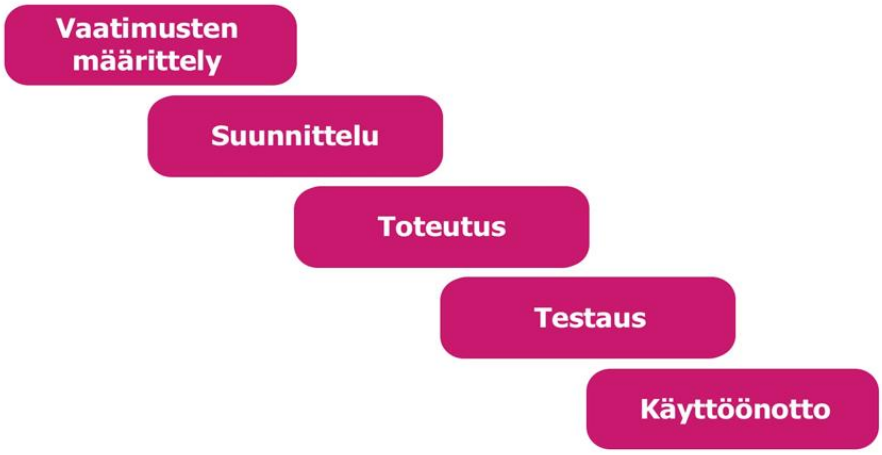
3 TOIMINTAMALLIEN ESITTELY

Ohjelmistotuotannossa on käytössä valtava määrä erilaisia toimintamalleja ja monesti myös erilaisia hybridiversioita niistä. Tästä syystä tässä opinnäytetyössä keskitytään vain muutamiin yleisimpiin menetelmiin.

3.1 Vesiputousmalli

Vesiputousmalli on yksi ensimmäisistä ohjelmistotuotannossa käytetyistä toimintamalleista. Alun perin vesiputousmalli otettiin ohjelmistotuotannossa käyttöön ensimmäisiä suurehkoja tietokoneohjelmia laadittaessa, kun havaittiin selkeä tarve systemaattisille työprosesseille. Ensimmäiset vesiputousmallit noudattivatkin yleisesti hyvin suoraviivaista järjestystä: asiakastarpeiden määritys, ohjelmiston suunnittelu, toteutus, testaus ja lopulta käyttöönotto. (Haikala & Mikkonen, 2011, s. 36)

Vesiputousmallissa prosessin yksittäiset osat suoritetaan järjestyksessä, vaihe vaiheelta, jolloin suoritetun vaiheen lopputulos toimii aina seuraavan vaiheen lähtökohtana. Mallin nimi juontaa juurensa sen rakenteeseen, sekä tapaan, jolla mallia monesti visualisoidaan, merkitsemällä prosessin päävaiheet laskujohteisesti vesiputousta muistuttavaan muotoon (Kuva 1). (Tutorialspoint 2021)



Kuva 1. Vesiputousmallin vaiheet (Macadamian, 2019).

Yhtenä ensimmäisistä ohjelmistotuotannossa käytetyistä toimintamalleista vesiputousmallia on käytetty alalla erittäin paljon. Etenkin sallimalla iteroinnin, pystytään useimpiin projekteihin ja tilanteisiin löytämään vesiputousmallia hyödyntäen soveltuva toimintamalli. Uudempia projektimallejakin tarkastellessa, löytyy niiden ytimestä lähes poikkeuksetta vesiputousmallin perusperiaate tai vesiputousmallia muistuttavia toimintatapoja. (Haikala & Mikkonen, 2011, s. 37)

Haikala ja Mikkonen huomauttavatkin jopa ketterän Scrum-mallin perustuvan käytännössä sarjaan lyhyitä vesiputouksia. (Haikala & Mikkonen, 2011)

Vesiputousmalli ei nykypäivänä kuitenkaan todennäköisesti ole ohjelmistotuotannossa paras ratkaisu. Vaikka vesiputouksia tehtäisiinkin useampia peräkkäin, saattaa täyden kierroksen suorittaminen hyvinkin kestää jopa useita kuukausia, jolloin sen aikana opittuja asioita päästään soveltamaan vasta pitkän ajan jälkeen. Tällöin edellisen vesiputouksen aikana opitut asiat ovat mahdollisesti päässeet jo unohtumaan tai projektihenkilökunta on saattanut joltain osin vaihtua, eikä opittuja asioita päästä projektissa enää hyödyntämään. (Haikala & Mikkonen, 2011, s. 41)

Vesiputousmallissa on myös muita huonompia puolia. Riskit saattavat realisoitua vasta järjestelmän testausvaiheessa, jolloin niihin reagointi saattaa tulle hyvinkin kalliiksi. Vaatimukset saattavat tarkentua, tai niitä voi tulla lisää projektin toteutusvaiheessa, jolloin niihin on hankala reagoida ja järjestelmä saattaa huonolla tuurilla olla jo valmistuessaan tavalla tai toisella ”vanhentunut”. Testaukset aloitetaan vasta projektin lopulla ja asiakaspalaute saadaan käytännössä vasta projektin käyttöönoton yhteydessä. Tällöin testauksessa tai asiakaspalautteessa ilmenneisiin ongelmiin on vaikeaa ja kallista reagoida. (Haikala & Mikkonen, 2011, s. 41)

Ongelmia saattaa aiheuttaa myös ihmisten erot ja projektitiimin jäsenet saattavatkin ymmärtää samat vaatimukset eri tavoin, josta luonnollisesti syntyy riski väärinymmärryksille. Projektiin käytetystä ajasta menee myös merkittävä osa dokumentaation luontiin, joka on kaikki pois itse ohjelmiston luomisajasta. (Macadamian, 2019)

3.2 Protoilu

Protoilulla tarkoitetaan ohjelmistotuotannossa prototyyppien rakentamista. Prototyypit ovat yleensä joltain osin vajaita kokonaisuuksia, joita käytetään monesti joidenkin ohjelmiston osien toiminnan tutkimiseen ja testaukseen.

Protoilussa käytetään pääasiassa kahta eri päätyyppiä: evoluutioprototyyppi ja kertakäyttöinen prototyyppi (voidaan käyttää myös termiä ”poisheitettävä prototyyppi”). Voidaan myös käyttää näiden kahden tyypin välimuotoja. Sen taustalla toimii vesiputousmallin perusidea sillä erolla, että protoilussa yksittäisissä vesiputouksissa on useita peräkkäin, jolloin jokaisessa yksittäisessä vesiputouksessa toteutetaan yleensä vain pieniä osia ohjelmistokokonaisuudesta. (Haikala & Mikkonen, 2011, s. 38)

3.2.1 Evoluutioprototyypit

Evoluutioprototyypit toimivat käytännössä todellisten komponenttien tapaan, jolloin muun muassa rajapintafunktiot palauttavat järkevän oloisia arvoja. Evoluutioprototyypit myös kuormittavat järjestelmää vähintään oikean komponentin tapaan, tai mieluusti jopa enemmän, jolloin voidaan varmistua komponentin soveltuvuudesta tuotantoon siirtyvän ohjelmiston osaksi. Periaatteeltaan evoluutioprototyyppi on erittäin lähellä iteratiivista kehitystä ja joissakin tapauksissa voidaan jopa kaikki ohjelmiston osat toteuttaa ensin prototyyppeinä, jonka jälkeen niitä voidaan alkaa yksitellen korvaamaan todellisilla toteutuksilla. (Haikala & Mikkonen, 2011, s. 38)

Vaikka evoluutioprototyypeillä voidaankin saavuttaa suuri hyöty ohjelmistotuotannossa, eivät ne kuitenkaan ole läheskään täydellinen ratkaisu. Suurimpana ongelmana näissä on monesti, että vaillinaisen luonteensa vuoksi, saattaa huonosti toimiva prototyyppi esimerkiksi projektin kiireellisen aikataulun vuoksi jäädä osaksi lopullista järjestelmää. Vaikka prototyypin ongelmana ei olisikaan sen suurempi asia, kuin esimerkiksi komponentin suoritusnopeus, voi se kuitenkin, etenkin useamman vastaavan ongelman kasaantuessa, vaikuttaa merkittävästikin lopullisen järjestelmän toimintaan. (Haikala & Mikkonen, 2011, s. 39)

Ominaisuuksiltaan rajallisen prototyypin ongelmana voidaan myös pitää sen aiheuttamia harhaluuloja lopullisen tuotteen analysoinnissa, johtaen kehitystyön harhautumiseen ja

mahdollisesti vaatimusten vajaaseen määrittämiseen. (Rapids Reproductions, Inc., 2021)

3.2.2 Kertakäyttöiset prototyypit

Kertakäyttöisillä prototyypeillä viitataan usein yksinkertaisempiin, tarvittaessa jopa vain paperille piirrettyihin kuviin, joiden avulla pystytään havainnollistamaan suunnitellun ohjelmiston ulkonäköä ja toiminnallisuutta. Esimerkiksi mobiilisovelluskehityksessä kertakäyttöinen prototyyppi voi olla muun muassa Figma- tai Adobe XD-sovelluksilla rakennettu interaktiivinen mockup, jolla asiakkaalle pystytään esittelemään sovellukselle suunniteltu rakenne ja jolla asiakas pystyy itse kokeilemaan sovelluksen sisällä navigointia ennen varsinaisen sovelluksen rakennusta.

Kuten Haikala ja Mikkonen toteavat, kertakäyttöisten prototyyppien ongelma on, että nähtyään hyvin toimivan prototyypin, asiakas voi luulla, että järjestelmä on jo lähes valmis, vaikka siitä on olemassa vain kuoret. Tästä syystä kehittäjät saattavatkin käyttää tarkoituksen mukaisesti ”käsinpiirretyn” näköisiä mockupeja, jotta vastaavia epäselvyyksiltä ei pääse syntymään. (Haikala & Mikkonen, 2011, s. 39)

3.3 Iteratiivisuus

Iteratiivisuudella tarkoitetaan ohjelmistotuotannossa toimintatapaa, jossa prosessin vaiheita toistetaan tarvittaessa useita kertoja. Näin toimimalla pystytään tarkentamaan kyseisissä vaiheissa ilmentyneitä ongelmia ja helpottamaan niiden ratkaisua. Taustalla iteratiivisessa kehityksessä toimii vesiputousmallia muistuttava rakenne, jossa tuotekehityksen syklit ovat vesiputousmallia huomattavasti lyhyemmät. Jokaisen syklin lopussa siihen asti toteutettuja vaatimuksia voidaan myös demonstroida asiakkaalle.

Iteratiivisessa kehityksessä voidaan myös päästä lähelle ketterien menetelmien toimintaperiaatteita, hyväksymällä ettei kaikkia vaatimuksia tunneta vielä kuin muutaman seuraavan syklin osalta ja uusia vaatimuksia tarkennetaan jatkuvasti kehitysprojektin edetessä. (Sininen Meteoriitti, 2013)Tuotekehityssyklien ollessa vesiputousmalliin verraten huomattavasti lyhyemmät, kasvaa testauksen määrä kehitysprojektin aikana merkittävästi. Vaikka testauksen määrällä saavutetaankin huomattava etu lopputuotteen laadussa, se myös hankaloittaa projektin aikataulutusta. Yleisesti iteratiivisessa

kehityksessä vaaditaankin testaukseen jonkinlaista automaatiota vähentämään projektihenkilökunnan työkuormaa testausten osalta. Toimiva vaatimushallinta ja tuotteenhallinta nousevat myös erittäin tärkeään rooliin. (Haikala & Mikkonen, 2011, s. 42)

3.4 Ketteryys

Ketterät (agile) menetelmät saivat alkunsa vuonna 2001 joukon kevyiden menetelmien kehittäjiä, kannattajia ja konsultteja kokoonnuttua kehittämään etenkin Yhdysvalloissa paljon sovelletuille suunnitelmaohjatuilla kehitysprosesseille vaihtoehtoisia, kevyempiä menetelmiä. (Haikala & Mikkonen 2011, 43)

Kokouksessa perustettiin myös Agile Alliance-järjestö, jonka tavoitteena on ketterien menetelmien edistäminen, sekä järjestön ”peruskirja”, Agile Manifesto.

Agile Manifesto luettelee järjestön arvot seuraavasti:

> “*Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:*
>
> ***Yksilöitä ja kanssakäymistä** enemmän kuin menetelmiä ja työkaluja.*
>
> ***Toimivaa ohjelmistoa** enemmän kuin kattavaa dokumentaatiota.*
>
> ***Asiakasyhteistyötä** enemmän kuin sopimusneuvotteluja.*
>
> ***Vastaamista muutokseen** enemmän kuin pitäytymistä suunnitelmassa.*
>
> *Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän.*” (Beck, ym., 2001)

Kuten järjestö arvoissaan painottaa, ovat muun muassa prosessit, työkalut ja kattava dokumentaatio arvokkaita ohjelmistokehityksen kannalta. Ketterässä toimintamallissa kuitenkin arvotetaan tyytyväinen asiakas ja toimiva ohjelmisto projektin kannalta näitä tärkeämmäksi.

Manifestissa (Beck, ym., 2001) luetellaan ketterän toimintamallin kaksitoista perusarvoa, jotka ovat vapaasti suomennettuna seuraavat:

1. Tärkeintä on asiakastyytyväisyys projektin alusta asti tasaiseen tahtiin toimitetuilla toimivilla ohjelmistoversioilla.

2. Sallitaan vaihtuvat vaatimukset jopa kehityksen myöhäisessä vaiheessa.
3. Toimitetaan jatkuvasti toimivia ohjelmistoversioita, muutamien viikkojen tai kuukausien välein, pyrkien aina tiheämpään toimittamiseen.
4. Liiketoiminta- ja kehityshenkilöiden tulee päivittäin työskennellä yhdessä.
5. Rakennetaan ohjelmistoja motivoituneiden yksilöiden ympärille. Annetaan heille tarvittava ympäristö ja tuki ja luotetaan, että he saavat tehtävänsä suoritettua.
6. Tehokkain tapa kommunikoida ja välittää tietoa kehitystiimissä on keskustelu kasvokkain.
7. Toimiva ohjelmisto on ohjelmistokehityksen edistymisen tärkein mittari.
8. Ketterät menetelmät edistävät kestävää kehitystä. Projektihenkilökunnan tulisi kyetä jatkamaan tasaiseen tahtiin koko projektin ajan.
9. Jatkuva teknisen erinomaisuuden ja hyvän suunnittelun tarkkailu tehostaa ketteryyttä.
10. Tekeminen tulisi pyrkiä tekemään mahdollisimman yksinkertaisesti: ylimääräisen työn minimointi on äärimmäisen tärkeää.
11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itsenäisesti ohjautuvissa tiimeissä.
12. Tiimin tulee tarkistella toimintaansa säännöllisin väliajoin ja hienosäätää toimintatapojaan sen mukaisesta parantaakseen työtehokkuuttaan.

Vaikka monet näistä perusarvoista vaikuttavatkin pääasiassa järkeviltä ja ovat yleisesti hyväksyttävissä, vaikuttaa perusajatus näiden taustalla olleen soviteltu lähinnä pienempiin asiakasprojekteihin. Etenkin suuremmissa projekteissa, joissa ohjelmistokokonaisuudet on jaettu osiin, joita toteuttavat useat yritykset mahdollisesti ympäri maailmaa, voi moni näistä olla hyvinkin vaikeasti saavutettavissa. Muun muassa päivittäin kasvokkain työskentely projektihenkilökunnan kesken saattaa olla käytännössä mahdotonta projektia työstettäessä samanaikaisesti eri puolilta maailmaa. Myös asiakkaan jatkuva läsnäolo kehityshankkeessa todennäköisesti lähinnä hidastaisi kehitystä ja laskisi projektihenkilökunnan työtehokkuutta, sekä jopa moraalia.

Kuten Haikala ja Mikkonenkin (Haikala & Mikkonen, 2011, s. 46) kirjassaan mainitsivat, on mielenkiintoista, ettei näitä manifestin kirjoittajien kehittämiä menetelmiä Scrumia lukuunottamatta enää nykypäivänä juurikaan sellaisenaan käytetä. Näihin menetelmiin kehitettyjä ideoita, kuten lyhyempien iteraatioiden käyttöä, kuitenkin hyödynnetään nykypäivänäkin, osana Scrumia.

3.5 Scrum

Vaikka Scrumin keskeisimmät periaatteet onkin listattu jo vuonna 1986 Harvard Business Review -lehdessä (Takeuchi & Nonaka, 1986, s. 138), sai se varsinaisesti alkunsa 1990-luvun alkupuolella Jeff Sutherlandin ja Ken Schwaberin innostuttua kehittelemään ohjelmistokehitysmenetelmää Sutherlandin ohjelmistokehitystiimille näiden perusteiden pohjalta. Scrum sai näin ollen alkunsa siis jo noin 10 vuotta ennen ketterien menetelmien yleistymistä muotitermiksi. (Haikala & Mikkonen, 2011, ss. 46-47)

Kuten kehittäjänsä Schwaber ja Sutherland mainitsivat, ”Scrum perustuu empirismiin ja lean-ajatteluun. Empirismin mukaan tieto tulee kokemuksesta ja päätösten tekemisestä havaintojen perusteella. Lean-ajattelu vähentää hukkaa ja keskittyy olennaiseen.” (Schwaber & Sutherland, 2020, s. 3)

Viimeisten vuosien saatossa Scrum on saavuttanut valtavan suosion ohjelmistokehitysprojekteissa, ja siitä onkin tullut yksi yleisimmin käytetyistä toimintamalleista. Scrumin suosio on noussut jopa niinkin valtavaksi, että sitä saatetaan hieman virheellisesti pitää jopa sanan ketterä synonyymina. Tämä ei tosin ole varsinaisesti aivan oikein, vaan tarkemmin ajateltuna sitä voidaan pitää jopa jonkinlaisena ketteryyden ”jäykistäjänä”. (Haikala & Mikkonen, 2011, s. 47)

3.5.1 Scrum-roolit

Toisin kuin esimerkiksi vesiputousmallissa, jossa projektihenkilökunnalla on useita eri rooleja, on Scrumissa rooleja vain kolme: tuotteen omistaja (Product Owner), Scrum-mestari (Scrum-master) sekä projektitiimi. Tuotteen omistaja on vastuussa projektin taloudellisesta tuloksesta, toimii rajapintana sidosryhmiin, kartoittaa sidosryhmien avulla järjestelmän vaatimukset, dokumentoi nämä vaatimukset tuotteen työlistaksi, sekä ylläpitää tätä työlistaa (product backlog) prioriteettijärjestyksessä. Scrumissa vaatimusten hallinta on siirretty käytännössä täysin tuotteen omistajan harteille. (Sininen Meteoriitti, 2013)

Scrum-mestari toimii projektissa osittain projektipäällikön tapaan, mutta kuten Haikala ja Mikkonen (Haikala & Mikkonen, 2011, s. 49) mainitsivat, Scrum-mestari on kuin ”projektipäällikkö ilman valtaa”. Scrum-mestarin tärkeimpiä tehtäviä ovat muun muassa Scrum-prosessin noudattamisen, sekä pyrähdysten tuloksen varmistaminen, tehtävien

valmistumisen toteaminen, kun määritellyt ehdot on täytetty (definition-of-done, DoD), sekä niiden valmiiksi merkitseminen. Näiden lisäksi Scrum-mestari vastaa tiimin toimintakyvystä ja tarvittaessa poistaa tiimin työskentelyä häiritseviä tai estäviä tekijöitä, kuten tiimiin sopimattomia jäseniä, tai ulkopuolisia haittoja, kuten tiimin jäsenten ylityöllistäminen toisissa projekteissa.

Tiimin koko projekteissa voi vaihdella, mutta lähtökohtana yleensä on, että tiimi koostuu erilaisista osaajista, joiden yksilöllistä osaamista pyritään hyödyntämään. Perusteena Scrumissa on, että tiimi on itseorganisoituva, eikä varsinaista projektipäällikköä ole. Tällöin myös Scrum-mestari toimii samalla tavoin tiimin jäsenenä, kuten kaikki muutkin. Projektin työskentelykäytännöt tiimi määrittää itsenäisesti vastaten samalla pyrähdyksen työlistan osittamisesta tehtäviksi, sekä näiden jakamisesta tiimin jäsenten kesken. (Haikala & Mikkonen, 2011, s. 49)

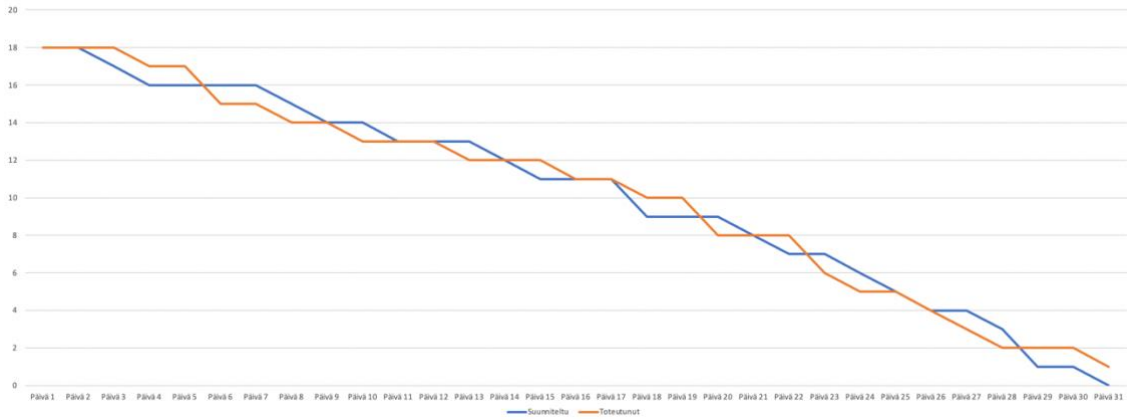
3.5.2 Scrum-prosessi

Scrum – kuten muutkin ketterät mallit – perustuu erimittaisten syklien ympärille, joista tärkeimpinä niin sanottu pyrähdys (sprint) sekä päivä. Pyrähdyksellä tarkoitetaan yleisesti yksittäistä ajanjaksoa, jonka aikana projektia työstetään. Jokaisen pyrähdyksen jälkeen siinä toteutettavan ohjelmiston tai ohjelmistokokonaisuuden osan tulisi ainakin periaatteessa olla julkaisuvalmis. Pyrähdysten pituus itsessään vaihtelee monesti organisaatioiden tarpeiden mukaan. Tyypillinen kesto pyrähdykselle on kuukausi, mutta voi tarpeista riippuen vaihdella esimerkiksi kahdesta viikosta kahteen kuukauteen. (Haikala & Mikkonen, 2011, s. 48)

Pyrähdyksen alussa pidetään pyrähdyksen suunnittelukokous (engl. sprint planning meeting), johon osallistuvat tiimi, Scrum-mestari, sekä tuotteen omistaja. Suunnittelukokouksessa tuotteen omistaja esittelee ensin tuotteen työlistan (product backlog), jonka jälkeen valitaan seuraavassa pyrähdyksessä toteutettavat tehtävälistan alkiot (product backlog item) sen mukaan, mihin tiimi pystyy sitoutumaan. Valitut alkiot pilkotaan yksittäisiin pienempiin tehtäviin, joille jokaiselle annetaan suhteellinen aika-arvio. Näiden aika-arvioiden tarkoituksena on auttaa tiimiä arvioimaan tuleviin pyrähdyksiin valittujen tehtävien määrää. (Haikala & Mikkonen, 2011, s. 50)

Aika-arvioiden yksiköllä ei ole käytännön merkitystä, joten yksikkönä voidaan käyttää esimerkiksi Fibonaccin lukujonoa, jonka avulla tiimi pystyy helposti hahmottamaan yksittäisistä tehtävistä syntyvän suhteellisen työkuorman.

Pyrähdyksen aikana tilannetta voidaan seurata edistymiskäyrällä (sprint burndown chart, Kuva 2). Edistymiskäyrää päivitetään jatkuvasti pyrähdyksen aikana, aina kun yksittäisiä tehtäviä saadaan valmiiksi. Näin pystytään seuraamaan pyrähdykseen valittujen tehtävien edistymistä. Vastaava edistymiskäyrä voidaan luoda myös projektikohtaisesti, jolloin pystytään seuraamaan lopputuotteen kehitystä. Yhtenä Scrumin suuren suosion syistä pidetään tehtävälistojen muuttumattomuutta pyrähdysten aikana. Tällä varmistutaan siitä, ettei tiimin tarvitse varautua vaatimusmuutoksiin päivittäin, vaan mahdolliset muutokset kohdistuvat aina seuraavan pyrähdyksen alkuun. (Haikala & Mikkonen, 2011, s. 50)



Kuva 2. Esimerkki pyrähdyksen edistymiskäyrästä (Haikala & Mikkonen, 2011).

Jatkuvaa seurantaa tehdään lyhyiden, päivittäisten kokousten (daily scrum) yhteydessä. Näissä kokouksissa käydään läpi mitä tiimin jäsenten ovat tehneet edellisen kokouksen jälkeen, mitä he tulevat tekemään ennen seuraavaa kokousta, sekä kartoitetaan mahdolliset työtä hidastavat seikat. Päivittäisiin kokouksiin osallistuvat lähtökohtaisesti vain tiimi ja Scrum-mestari. Myös tuotteen omistajan on mahdollista osallistua, mutta hänellä ei ole oikeutta vaikuttaa kokouksen sisältöön, eikä määrätä lisävaatimuksia tiimille. (Sininen Meteoriitti, 2013)

Pyrähdyksen lopussa pidetään katselmointipalaveri (sprint review), jossa tarkistetaan pyrähdyksen aikana suoritetut tehtävät ja esitellään tuotokset tuotteen omistajalle, sekä sidosryhmien edustajille. Samalla kerätään mahdollinen palaute sidosryhmien

edustajilta. Kuten Schwaber ja Sutherland Scrum-ohjeessaan (Schwaber & Sutherland, 2020, ss. 9–10) toteaa, on pyrähdyksen katselmointi työpajamainen kokous, jossa osallistujat keskustelevat menneestä pyrähdyksestä sekä siitä, mitä kannattaisi tehdä seuraavaksi.

Pyrähdyksen lopussa pidetään myös arviointipalaveri, jossa tiimi, Scrum-mestari ja tuotteen omistaja käyvät läpi edellisen pyrähdyksen ajalta, miten pyrähdys meni, saavutettiinko tavoitteet ja mitä toimintatapoja mahdollisesti tulisi kehittää seuraaviin pyrähdyksiin.

3.5.3 Työkalut

Yksin Scrumin oleellisimmista työkaluista on tehtävätaulu (Kuva 3). Tehtävätaulu koostuu useasta sarakkeesta, eli listasta, jotka monesti nimetään esimerkiksi tehtävien vaiheiden mukaan, kuten *ei aloitettu*, *kesken*, *tarkastuksessa* ja *valmis*. Näihin listoihin lisätään kortteja, jotka sisältävät yksittäisiä tehtäviä, joille määritetään työmääräarviot. Tehtäväkortteja siirretään listasta toiseen aina tehtävän tilan muuttuessa. Esimerkiksi alettaessa työstämään uutta tehtävää siirretään kyseinen tehtäväkortti listasta *ei aloitettu*, listaan *kesken*. (Haikala & Mikkonen, 2011, s. 55)

Tehtävän siirtyessä listaan *tarkastuksessa* tarkastaa Scrum-mestari sen tilan ja tehtävän ollessa asianmukaisesti toteutettu, siirtää sen listaan *valmis*. Jos tehtävän toteutuksessa havaitaan puutteita, siirretään se takaisin kehitykseen, esimerkiksi listaan *kesken*. (Haikala & Mikkonen, 2011, s. 55)

Kuva 3. Esimerkki tehtävätaulun rakenteesta.

3.6 Lean-ajattelu

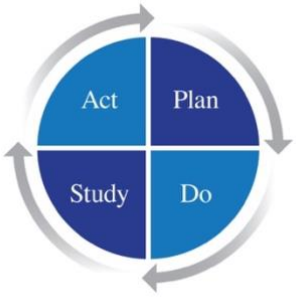
Lean itsessään ei varsinaisesti ole toimintamalli, kuten esimerkiksi Scrum tai vesiputousmalli, vaan ennemminkin prosessin kehityksessä käytetty ajattelutapa.

Lean-ajattelu perustuu pääasiassa kahteen peruspilariin: eliminoidaan hukka (engl. eliminate waste) ja ihmiskeskeisyys (engl. center on the people who add value) (Haikala & Mikkonen, 2011, s. 55) tai kuten Lean Enterprise Instituten verkkosivuilla on mainittu: ”*Simply, lean means creating more value for the customer with fewer resounces*.” (Lean Enterprise Institute, 2021)

Hukka Lean-ajattelussa tarkoittaa käytännössä kaikkea, joka ei tuota näkyvää ja välitöntä lisäarvoa asiakkaalle. Näin ollen esimerkiksi asioiden tekeminen ennakkoon tai ”varastoon” lasketaan hukaksi. Myös muun muassa täydellisen määrittelydokumentin luominen voidaan laskea hukaksi, jos kaikkia dokumentissa määriteltyjä asioita ei lopulta toteutetakaan. Ihmiskeskeisyydellä tarkoitetaan lisäarvoa tuottaviin yksilöihin keskittymistä. Pyritään siis panostamaan toiminnassa erityisesti ”ruohonjuuritasolla” vaikuttaviin työntekijöihin, eikä niinkään esimerkiksi johtoportaaseen. (Haikala & Mikkonen, 2011, ss. 54–55)

3.7 Kanban

Kanban on Scrumin tavoin kokemusperäinen toimintamalli, jossa oletuksena on aloittaa projekti aina toimintamallin muokkaamattomalla perusmallilla, jota projektin edetessä kehitetään juuri kyseiseen projektiin sopivammaksi. Mallissa seurataan Demingin laatuympyrää (Kuva 4), jonka englannin kielinen nimi – PDSA Cycle – tulee sen vaiheista Plan, Do, Study ja Act, jotka ovat vapaasti suomennettuna suunnittele, toteuta, arvioi ja vakiinnuta. (Gofore, 2020)



Kuva 4. Demingin laatuympyrä (The W. Edwards Deming Institute, 2021).

Yksinkertaisuudessaan Kanban on kolmea periaatetta noudattava työnhallintatapa. Nämä periaatteet ovat *aloita millä pystyt*, *hae vähittäistä muutosta* ja *kunnioita nykyisiä prosesseja, rooleja ja velvollisuuksia*. Näiden lisäksi tärkeissä rooleissa ovat työn kulun visuaalinen seuranta, yhtäaikaisen tekemisen rajoittaminen niin sanotuilla kaistanrajoittimilla, sekä läpimenoajan, eli työn yksittäisissä vaiheissa käytetyn ajan mittaaminen. (Lehtonen, ym., 2014)

Työtä seurataan yleisesti vastaavanlaisella tehtävätaululla, kuin Scrumissa ja kaistanrajoittimet varmistavat, että samassa listassa, eli työvaiheessa olevien tehtävien määrä on rajoitettu, jotta projektitiimi pystyy paremmin keskittymään työn alla oleviin tehtäviin. Vaikka päällisin puolin Kanban muistuttaakin Scrumia, on näissä toimintamallissa kuitenkin selkeitä eroja. Päällimmäisenä kuitenkin pyrähdyksien puuttuminen Kanbanissa. Tämä mahdollistaa uusien tehtävien aloittamisen heti kun kaistalla on tilaa. (Haikala & Mikkonen, 2011, s. 55)

4 PROJEKTIN LÄPIKÄYNTI

4.1 Toteutusvaihe

Tässä luvussa esitellään mobiilisovelluksen beta-version kehitysprojektia vaihe vaiheelta ja käydään läpi projektin aikana havaittuja sudenkuoppia.

4.1.1 Edistymisen seuranta

Projektin alkaessa ja kehitystiimin aloittaessa React Nativen opiskelu, ei edistystä seurattu lainkaan. Varsinaisen kehitystyön alkaessa alettiin edistymistä seuraamaan kaksi kertaa viikossa pidettävissä verkkokokouksissa. Kokouksissa tiimin jäsenet esittelivät edellisen kokouksen jälkeen toteuttamiaan ominaisuuksia, sekä kartoitettiin ongelmakohtia ja pyrittiin yhdessä ratkaisemaan ne kokousten aikana.

Kokousten lisäksi kehityksen seurannassa oli käytössä Microsoft Teams, sekä projektin myöhemmässä vaiheessa Planner-ohjelmistot. Teams toimi tiimin pääasiallisena kommunikaatiokanavana. Planneriin, joka on käytännössä Microsoftin versio Scrumissa ja Kanbanissakin käytetystä tehtävätaulusta, tehtävät jaettiin yksittäisiksi pienemmiksi tehtäviksi, jotka allokoitiin yksittäisille projektitiimin jäsenille.

4.1.2 Sovelluksen ohjelmointi

Toteutusvaiheeseen edettyään, tiimi aloitti ohjelmiston kehityksen sovelluksen päänäkymästä. Tämän jälkeen muita sovelluksen näkymiä alettiin toteuttamaan yksi kerrallaan asiakkaan ilmaiseman tärkeysjärjestyksen mukaisesti. Tässä vaiheessa projektin eteneminen oli melko hidasta projektitiimin jäsenten muiden opintojen, harjoitteluiden, sekä töiden viedessä ison osan heidän ajastaan.

Myös projektitiimin toistaiseksi suhteellisen vähäinen React Native osaaminen vaikutti luonnollisesti osaltaan kehityksen nopeuteen.

Projektitiimi kohtasi useampia tilanteita, joissa sovelluksen kehitys lähes pysähtyi ongelmatilanteiden selvittelyn vuoksi. Koska projekti toteutettiin theFIRMAn

ulkopuolisena projektina, ei siihen ollut mahdollista saada lisää tekijöitä theFIRMAsta, vaan projektitiimin piti itsenäisesti ratkaista ongelmat.

4.1.3 Beta-version toiminnallisuuden rajaus

Projektin ollessa jo hyvin käynnissä ja asiakkaan toimittaessa jatkuvasti lisävaatimuksia sovelluksen toiminnallisuuteen liittyen, kävi selväksi, beta-version sisältöä olisi pakko rajata. Tiimillä oli toki jo tiedossa, että ominaisuuksia tulisi rajata beta-versioon, mutta konkretisoitui asia vasta asiakkaan toimittamien sovelluksen alkukyselyyn tarkoitettujen kysymyslistojen laajuudesta.

Kysymyslistat sisälsivät huomattavan määrän kysymyksiä aiheisiin, joita ei ollut beta-versioon mahdollista toteuttaa rajallisen aikataulun, sekä käytössä olevien työvoimaresurssien puitteissa. Asiasta ilmoitettiin asiakkaalle ja sovittiin asiakkaan kanssa kokous. Kokouksessa asiakkaalle esitettiin tiimin kanta siitä, mitkä ominaisuudet tiimi koki sovelluksen toiminnan kannalta tärkeimmiksi, ydinominaisuuksiksi, sekä perusteltiin suositukset aikatauluun ja resursseihin vedoten. Yhdessä asiakkaan kanssa päädyttiin keskittymään beta-versiossa vain aikuiseen pääkäyttäjään, jolloin käyttäjän lapsen suun hyvinvointiin liittyvät ominaisuudet jäisivät sovelluksen kehityksen myöhempään vaiheeseen.

Beta-versioon toteutettavaksi karsittiin alkukyselyn toteutus, kyselyn pisteytyksen suunnittelu ja toteutus, pisteiden visualisointi sovelluksen päänäkymässä, sekä toiminnallisuus pisteiden päivityksen mahdollistamiseksi.

4.2 Testaus

Sovelluksen beta-version testaus suunniteltiin toteutettavaksi asiakkaan kokoaman testiryhmän toimesta projektin lopulla. Testauksen toteutukseen valittiin React Nativen Expo-ohjelmistokehys, hyödyntäen siihen yhdistettävää Expo Go -mobiilisovellusta, jolla kehitysvaiheessa oleva sovellus on mahdollista saada toimimaan testiryhmän mobiililaitteilla. Tarkkaa testaussuunnitelmaa ei kuitenkaan tehty, eikä testauksen ajankohtaa sovittu etukäteen.

4.3 Projektin reflektointi

Vaikka sovitut tavoitteet saavutettiinkin mobiilisovelluksen kehitysprojektissa lähes täysin, jäi prosessiin myös parantamisen varaa.

4.3.1 Vaatimusten määrittely

Projektin aikana useampikin asia olisi voitu hoitaa toisin. Vaatimusten määrittely ja rajaus toteutettiin selkeästi liian myöhään projektin ollessa jo melko pitkällä. Tämä johti projektin toteutusvaiheessa siihen, ettei aikaa pystytty hyödyntämään niin tehokkaasti kuin olisi ollut mahdollista. Jos vaatimukset olisi rajattu heti projektin alkaessa sovelluksen beta-version aikataulun ja resurssit huomioon ottaen, olisi projektin toteutusvaiheessa ollut merkittävästi reilummin aikaa itse sovelluksen ohjelmointiin.

Asiakkaalla itsellään ei ollut sovelluskehityksestä entuudestaan kokemusta, eikä hän näin ollen luonnollisestikaan osannut varautua sovelluskehitysprojekteissa ilmeneviin vastoinkäymisiin.

Monesti vastaavissa projekteissa asiakas saattaa ilmaista haluavansa jotain, jota tämä ei kuitenkaan teknisesti ottaen todellisuudessa tarvitse. Vastaavasti tilanne saattaa olla myös päinvastainen, jolloin asiakas ei halua ominaisuuksia, jotka kuitenkin ehdottomasti tulee toteuttaa sovelluksen toimivuuden takaamiseksi. Yhtenä suurimpana syynä tähän saattaa monesti olla, ettei asiakas halua maksaa jostain, mitä ei itse sillä hetkellä koe tarvitsevansa. (Kähönen, 2016)

Kuten monesti vastaavissa projekteissa, ei tässäkään projektissa asiakas osannut itse määritellä haluamiaan ominaisuuksia muuten kuin suuntaa antavasti, jolloin vaatimusten määrittely jäi projektitiimin vastuulle.

4.3.2 Projektin seuranta

Vaikka projektin etenemistä alettiinkin seuraamaan projektin aikana, olisi tämä ehdottomasti pitänyt aloittaa välittömästi projektin käynnistyttyä. Koska käyttöön otettu Microsoft Planner-ohjelmisto vastaa Scrumissa ja Kanbanissa käytettyjä tehtävätauluja,

olisi projektissa ollut helppoa hyödyntää kyseisten toimintamallien perusteita joko sellaisenaan tai projektiin sovitetulla hybridimallilla, kuten esimerkiksi Scrumbanilla.

4.3.3 Beta-version testaus

Sovelluksen beta-version testaus sovittiin suoritettavaksi ennen projektin luovutusta sitä jatkavalle kehitystiimille. Vaikka beta-version testaus saatiinkin toteutettua, ei aikataulusyistä käyttäjän tietojen keruuta ehditty beta-vaiheen testaukseen mennessä toteuttamaan.

4.4 Toimintamallien sopivuuden vertailu

Tässä luvussa kartoitetaan tutkimustulosten perusteella, mitä toimintamallia mobiilisovelluksen beta-version kehityksessä olisi optimaalisesti ollut kannattavaa hyödyntää.

4.4.1 Vesiputousmalli

Perinteinen vesiputousmalli olisi saattanut olla hyvinkin perusteltu vaihtoehto kyseiselle projektille. Tällöin vaatimukset ja rajaukset olisi kartoitettu tarkasti heti projektin alussa, jonka jälkeen projektitiimi olisi päässyt toteuttamaan muuttumatonta visiota beta-versiosta koko projektin ajan. Myöskään työskentelyyn ei olisi oletettavasti tullut katkoksia. Tosin tällöin asiakkaalta olisi vaadittu huomattavasti tarkemmat ennalta tehdyt määrittelyt toivotuista ominaisuuksista, eikä asiakas ollut kykeneväinen tekemään niitä ilman kehittäjien teknistä tietämystä.

4.4.2 Protoilu

Koska kyseessä oli sovelluksen beta-versio, voidaan projektia toki periaatteessa ajatella eräänlaisena prototyyppinä. Koska evoluutioprototyyppien ei kuitenkaan oletusarvoisesti ole tarkoitus päätyä osaksi lopullista sovellusta, toisin kuin kyseisessä projektissa toteutetun sovelluksen beta-version, ei protoilu todennäköisesti olisi ollut projektiin optimaalisin toimintamalli.

4.4.3 Iteratiivisuus

Vaikka projektin alussa ei vaatimuksia määritettykään niin tarkkaan kuin olisi pitänyt, oli kuitenkin itsestään selvää, ettei kaikkia asiakkaan sovellukseen toivomia ominaisuuksia pystyttäisi toteuttamaan aikamääreiden puitteissa. Tämän huomioon ottaen, olisi iteraatioita varmasti kannattanut hyödyntää projektin aikana. Jos vaatimusten määrittely olisi tehty projektin alussa optimaalisella tavalla, olisi sovelluksen kehitys voinut olla mahdollisesti hyvinkin tehokasta iteroimalla yksi ominaisuus kerrallaan.

4.4.4 Ketteryys

Ketterien menetelmien ollessa yleisesti etenkin pienempiin projekteihin hyvinkin perusteltu toimintamalli, voidaan olettaa, että seuraamalla Agile Alliance-järjestön perusarvoja, olisi projektin kehitys voinut sujua ainakin jossakin mittakaavassa paremmin.

4.4.5 Scrum

Scrumin ollessa yksi käytetyimpiä toimintamalleja IT-alalla nykypäivänä, on selvää, että siihen on myös syynsä. Tarkkojen roolikuvausten ja niihin liitettyjen vastuiden määrittely, prosessin järjestelmällisyys ja säännöllisyys, sekä muun muassa tehtävätaulun käyttö olisivat varmasti helpottaneet ja tehostaneet työskentelyä myös tässä projektissa. Koska Scrum-aiheista opiskelumateriaalia löytyy verkosta erittäin paljon, josta valtava määrä myös ilmaiseksi, olisi Scrumia käytettäessä ollut helppo hakea tietoa, jos toimintamallin toimintaan liittyen olisi ilmentynyt ongelmatilanteita tai epäselvyyksiä.

4.4.6 Lean

Vaikka Lean onkin enemmänkin ajattelutapa, kuin varsinainen toimintamalli, kuvastaa se tässä työssä mainituista malleista varmasti parhaiten projektin aikana käytettyjä menetelmiä. Etenkin Lean-ajattelun peruspilarit, hukan eliminointi ja ihmiskeskeisyys, näkyivät projektin aikana. Projektiryhmä kehitti vain oleellisia ominaisuuksia, eikä hukkaa näin ollen päässyt niiden osalta syntymään. Takautuvasti projektin edistystä arvioidessa

on helppo havaita tehdyt virheet, kuten jo mainittu vaatimusten määrittelyn lykkääminen projektin myöhempään vaiheeseen. Toinen Lean-ajattelun peruspilareista – hukka – ei siis tässä mielessä toteutunut. Lean-ajattelua alusta asti hyödyntäen olisi tämä mahdollisesti kuitenkin toteutettu jo projektin alussa.

4.4.7 Kanban

Koska projektissa otettiin käyttöön Microsoft Teams, sekä sen Planner-lisäosa, oli käytössä Kanbanin tehtävätaulua vastaava työkalu. Myös yhtäaikaista tekemistä rajoitettiin määrittämällä aktiivisia tehtäviä vain rajoitettu määrä kehittäjää kohden. Näiden jälkeen ainoana Kanbanin pääpiirteistä hyödyntämättä jäi vain yksittäisissä vaiheissa käytetyn ajan mittaaminen. Vaikka aikaa mittaamalla olisikin periaatteessa ollut mahdollista käyttää aikaa tehokkaammin, on epätodennäköistä, että näin tekemällä olisi kyseisessä projektissa juurikaan tehokkuus parantunut. Projektiryhmän jäsenillä oli jatkuvasti heille allokoituja tehtäviä ja tehtävien allokointi tehtiin jokaisen jäsenen tietotaidon mukaan. Näin ollen, vaikka aikaa olisi mitattu, ei se todennäköisesti olisi tuottanut juurikaan lisäarvoa sovelluksen kehitykseen.

4.5 Projektin jatko

Mobiilisovelluksen kehitysprojekti jatkui uuden projektiryhmän toimesta. Uusi projektiryhmä aloitti beta-version testauksella.

5 YHTEENVETO JA POHDINTAA

Tämän opinnäytetyön tavoitteena oli kartoittaa erityisesti pienissä projektitiimeissä käytetyimmät projektinhallinnan toimintamallit sekä takautuvasti kartoittaa, mitä toimintamallia, tai niiden yhdistelmää käyttämällä suun hyvinvointia edistävän mobiilisovelluksen beta-version kehityksessä olisi suoriuduttu tehokkaimmin.

Tutkimuksen aikana selvisi, että jotkin yritykset ovat selvästi erikoistuneet tiettyihin toimintamalleihin, mikä on varmasti toimiva ratkaisu tiettyyn pisteeseen saakka. Mutta mitä sitten, kun ennalta opiskeltu toimintamalli ei sovikaan projektin toteutukseen? Vaikka yksittäisiin toimintamalleihin erikoistumisesta varmasti saadaankin merkittävä hyöty kyseisten yritysten toimintaan projekteissa, joissa näitä toimintamalleja on järkevää ja tehokasta hyödyntää, saattavat vaihtoehtoiset toimintamallit jäädä tämän takia jopa kokonaan pois yrityksen toimintamallivalikoimasta. Tällöin vaihtoehtoisten toimintamallien ominaisuuksia ei välttämättä päästä hyödyntämään, vaikka joku näistä olisikin selkeästi kannattavampi vaihtoehto projektin tehokkaan toteutuksen kannalta.

Uskonkin, että toimintamallin valinta tulisi aina tehdä projektikohtaisesti, vaikka yrityksellä tai yksittäisellä projektitiimillä olisikin erityisen vahva osaaminen jostain tietystä toimintamallista.

Tämän opinnäytetyön aikana toteutetun suun hyvinvointia edistävän mobiilisovelluksen beta-version kehityksessä olisi ehdottomasti kannattanut soveltaa alusta alkaen jossakin määrin ketteriä menetelmiä. Näin tiimin resursseja olisi paremmin pystytty seuraamaan ja työtehtävien allokointi jokaisen projektitiimin jäsenen projektiin käytettävissä olevan työskentelyajan mukaan olisi pystytty toteuttamaan mahdollisimman tehokkaasti.

Koska asiakkaan vaatimuksia ei projektin alkaessa määritelty tarpeeksi tarkasti eikä sovelluksen beta-version sisältöä rajattu, jouduttiin näihin käyttämään aikaa varsinaisessa kehitysvaiheessa. Luomalla tarkat määritykset ja rajaamalla beta-vaiheessa toteutettavat ominaisuudet perusteellisesti ja tarkasti projektin alkaessa olisi varsinaiseen kehitystyöhön jäänyt merkittävästi enemmän aikaa ja toteutus olisi ollut hallitumpaa. Näin olisi myös vältytty kehitystyön rikkonaisuudelta, joka aiheutui kehitystiimin joutuessa väliaikaisesti keskeyttämään kehitystyön vaatimusten ja niiden rajauksen määrittämisen ajaksi.

Koska projektissa oli tarkoitus toteuttaa ainoastaan rajattu määrä sovelluksen ydinominaisuuksia, uskon tämän kaltaisten projektien kohdalla kannattamattomaksi hyväksyä ainakaan kovin merkittäviä uusia vaatimuksia kesken projektin. Tämän vuoksi perinteisempää vesiputousmallia vähintään soveltaen ja iteraatioita hyödyntäen olisi mahdollisesti saavutettu hyvinkin suoraviivainen ja selkeä tavoite projektille. Näin olisi vältytty vähintäänkin kesken projektin syntyneeltä kehitystauolta, joka taas olisi mahdollistanut paremman jatkumon kehitysprosessissa.

LÄHTEET

Beck, K.;Beedle, M.;van Bennekum, A.;Cockburn, A.;Cunningham, W.;Fowler, M.;. . . Sutherland, J. (2001). *Manifesto for Agile Software Development*. Haettu 22. huhtikuu 2021 osoitteesta http://agilemanifesto.org/

Gofore. (2020, maaliskuu 18). *gofore.com*. Retrieved huhtikuu 29, 2021, from https://gofore.com/mika-on-kanban/

Haikala, I., & Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt.* Helsinki: Talentum Media Oy.

Kähönen, P. (2016). *pasaati.com.* Retrieved toukokuu 1, 2021, from https://www.pasaati.com/hubfs/Konvertoivat_materiaalit/2016_11_Vaatimusmaarittely/Projektin_vaatimusmaarittely.pdf

Lean Enterprise Institute. (2021). *Lean Enterprise Institute*. Retrieved huhtikuu 29, 2021, from https://www.lean.org/whatslean/

Lehtonen, T., Tuomivaara, S., Rantala, V., Känsälä, M., Mäkilä, T., Jokela, T., . . . Isomäki, M. (2014, syyskuu). *Sulautettujen järjestelmien ketterä käsikirja*. Retrieved toukokuu 24, 2021, from https://tt.utu.fi/embedded_kasikirja/1/4/index.html

Macadamian. (2019, toukokuu 17). *macadamian.com*. Retrieved huhtikuu 29, 2021, from https://www.macadamian.com/learn/when-to-use-waterfall-vs-agile/

Rapids Reproductions, Inc. (2021). *Rapids Reproductions, Inc.* Retrieved toukokuu 24, 2021, from https://rapidsrepro.com/advantages-disadvantages-prototyping/

Schwaber, K., & Sutherland, J. (2020). *Scrumwell.* Retrieved huhtikuu 29, 2021, from https://scrumwell.files.wordpress.com/2021/02/2020-scrum-guide-finnish.pdf

Sininen Meteoriitti. (2013, kesäkuu 6). *Sininen Meteoriitti*. Retrieved toukokuu 24, 2021, from https://meteoriitti.com/2013/06/06/ketteryys-haltuun-scrum-pahkinankuoressa/

Takeuchi, H., & Nonaka, I. (1986, helmikuu). *The new new product development game.* Retrieved from

http://damiantgordon.com/Methodologies/Papers/The%20New%20Product%20Development%20Game.pdf

The W. Edwards Deming Institute. (2021). *PDSA Cycle - The W. Edwards Deming Institute*. Retrieved toukokuu 30, 2021, from https://deming.org/explore/pdsa/

Tutorialspoint. (2021a). *Tutorialspoint - SDLC - Iterative Model*. Retrieved toukokuu 24, 2021, from https://www.tutorialspoint.com/sdlc/sdlc_iterative_model.htm

Tutorialspoint. (2021b). *Tutorialspoint - SDLC - Waterfall Model*. Retrieved toukokuu 24, 2021, from https://www.tutorialspoint.com/sdlc/sdlc_waterfall_model.htm