Haaga-Helia
ammattikorkeakoulu Oy

Tietoturvaohjelmiston kehitysmallin päivittäminen paremmaksi ja ketterämmäksi

Kalle Verho

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

Haaga-Helia ammattikorkeakoulu Oy

Tiivistelmä

13.12.2019

Tekijä(t)
Kalle Verho

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Tietoturvaohjelmiston kehitysmallin päivittäminen paremmaksi ja ketterämmäksi

Sivu- ja liitesivumäärä
38 + 5

Opinnäytetyön otsikko englanniksi
Upgrading and Improving the Development Model of a Cybersecurity Product

Monimutkaisten tietojärjestelmien kehittäminen on erittäin vaikeaa työtä, koska etukäteen on yleensä mahdotonta tietää, mikä on paras lopullinen ratkaisu. Ketterä lähestymistapa on sekä tehokkain menetelmä kompleksisten suuren lisäarvon tuotteiden kehittämiseen että paras strategia niiden sisältämien riskien hallitsemiseen. Ketterän paradigman noudattaminen takaa parhaat edellytykset nykypäivän hektisillä markkinoilla, joilla yritysten pitää reagoida markkinoiden oikkuihin, viraali-ilmiöihin ja kulutustrendeihin.

Tämän opinnäytetyön tavoite on antaa lukijalle kuva ketterästä ohjelmistokehityksestä sekä lisätä ymmärrystä siitä, kuinka ohjelmistokehitystä tekevä yritys voi järjestää tiiminsä käyttämään ketteriä menetelmiä mahdollisimman hyvin – siten, että tiimi tuottaa hyvää laatua korkealla velositeetillä ja pitää sidosryhmät ja asiakkaat tyytyväisinä.

Opinnäytetyöni on laadullisin menetelmin toteutettu tutkimustyyppinen opinnäytetyö, jossa on käsitelty ketterään ohjelmistokehitykseen ja tarkasteltavan tuotteen ominaisuuksiin liittyvä olennainen teoria, tarkasteltu kehitysmallin parantamishankkeen läpivienti suomalaisessa tietoturvaohjelmistoja toimittavassa yhtiössä sekä toteutettu teemahaastattelu seitsemälle asiantuntijalle.

Tuloksien mukaan kehitysmallin parantamishankkeessa on onnistuttu joka suhteessa erinomaisesti ja kaikki sille asetetut tavoitteet on saavutettu. Tiimi toteuttaa ketterän paradigman periaatteita jokapäiväisessä toiminnassaan ja on sitoutunut kehitysmallin jatkuvaan parantamiseen.

Opinnäytetyötä voi hyödyntää samankaltaisten hankkeiden suunnitteluun ja toteuttamiseen. Haastattelukysymyksiä voi hyödyntää tuotetiimien haastatteluun etenkin, jos parannushankkeen onnistumista halutaan arvioida kriittisesti.

Asiasanat
Ketterä ohjelmistokehitys, ketterät menetelmät, ohjelmistokehitys, agile, Scrum, Kanban

Sisällys

1 Johdanto

1.1 Aihe ja tausta

Opinnäytetyöni käsittelee suomalaisen, melko perinteikkään tietoturvaohjelmistoja toimittavan pörssiyhtiön (myöh. Yhtiö) uusimman tietoturvaohjelmiston (myöh. Tuote) kehitysmallin muuttamista ketterämmäksi ja paremmaksi. Tuote on Yhtiön tulevaisuudelle erittäin tärkeä ja siihen investoidaan merkittävästi, ja tuotteen kehittämiselle on myös myönnetty EU:n tukea.

Aloitin tietojen keräämisen tammikuussa 2018 ja viimeiset huomiot olen tehnyt joulukuussa 2019. Kyseisenä aikana olen toiminut Yhtiössä kolmessa eri positiossa:

1. Tammikuu-joulukuu 2018: opinnäytetyössä tarkasteltavan Tuotteen tukipäällikkö, tehtävinä tuotteen tukimallin suunnittelu ja toteuttaminen. Kuuluin myös tuotteen lanseeraamistiimiin, joka julkaisi tuotteen maaliskuussa 2018, minkä jälkeen tiimi aloitti tuotteen markkinaseurannan.
2. Tammikuu-heinäkuu 2019: Yhtiön erään toisen tuotteen kehityspäällikkö ja scrummaster, tehtävinä kehitysmallin parantaminen ja ketterien menetelmien käyttöönotto. Tuote oli kriittisellä polulla ja kehitystiimi kriisin partaalla.
3. Elokuusta 2019 nykyhetkeen: opinnäytetyössä tarkasteltavan Tuotteen kehityspäällikkö ja scrummaster, tehtävinä kehitysmallin ylläpito ja jatkokehitys sekä muut Tuotteeseen liittyvät kehitys- ja parannushankkeet.

Pitkä tarkasteluaika, erilaiset positiot ja yksikkörajat ylittävä yhteistyö Yhtiössä ovat suoneet minulle hyvät edellytykset tarkastella Tuotetta, tiimiä ja kehitysmallin parantamishanketta objektiivisesti ja monesta erilaisesta näkökulmasta.

1.2 Menetelmät, tavoite ja rajaus

Opinnäytetyöni on laadullisin menetelmin toteutettu tutkimustyyppinen opinnäytetyö, jonka tavoite on antaa lukijalle kuva ketterästä ohjelmistokehityksestä sekä lisätä ymmärrystä siitä, kuinka ohjelmistokehitystä tekevä yritys voi järjestää tiiminsä käyttämään ketteriä menetelmiä mahdollisimman hyvin – siten, että tiimi tuottaa hyvää laatua korkealla velositeetillä ja pitää sidosryhmät ja asiakkaat tyytyväisinä. Tavoite saavutetaan käsittelemällä ketterään ohjelmistokehitykseen ja Tuotteen ominaisuuksiin liittyvä olennainen teoria, tarkastelemalla kehitysmallin parantamishankkeen läpivientiä Yhtiössä ja analysoimalla haastatteluvastaukset. Opinnäytetyöni vaatii lukijaltaan ohjelmistokehityksen menetelmien ja käytäntöjen perustuntemusta.

1.3 Rakenne

Teoriaosa alkaa luvusta 2, jossa esittelen ketterää ohjelmistokehitystä ja sen historiaa, nykypäivää ja tulevaisuutta yleisemmin ja kaksi Tuotteen kehitysmallissa käytettävää ketterää kehitysmenetelmää yksityiskohtaisemmin. Luvussa 3 käsitellään Tuotteen oleelliset ominaisuudet (esim. miksi valittu arkkitehtuuri vaikuttaa kehitysmallin valintaan), miten Tuotteen kehitysmalli on muuttunut sen elinkaaren eri vaiheissa, kuinka varsinainen kehitysmallin muutos toteutettiin ja millaiseksi kehitysmalli lopulta muodostui. Tutkimusosa alkaa luvusta 4, jossa käsittelen tutkimuksen tavoitteen, tuotoksen ja tutkimusmenetelmät. Luvussa käyn 5 käyn läpi tutkimuksen tulokset eli tekemäni haastattelut. Luvussa 6 pohdin työn tuloksista tehdyt johtopäätökset sekä työlle asetettujen tavoitteiden saavuttamisen.

1.4 Keskeiset käsitteet

Opinnäytetyön keskeiset käsitteet ovat ketterä ohjelmistokehitys, ketterät menetelmät, Scrum ja Kanban.

Ketterä ohjelmistokehitys on paradigma, joka perustuu Ketterän ohjelmistokehityksen julistukseen. Paradigmalle tyypillistä on suhtautua jatkuviin muutoksiin positiivisesti sekä toimia inkrementaalisesti ja iteratiivisesti, eli kehittää ja toimittaa aikarajatuissa pienissä palasissa, jatkuvasti tarkastellen ja oppien sekä toistaen sykliä koko tuotteen kehityksen ajan.

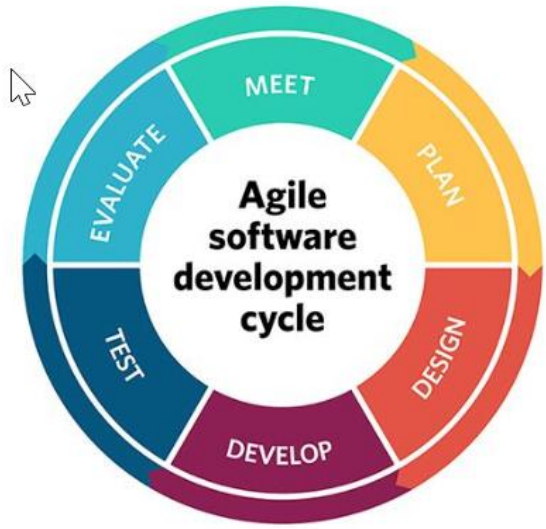
Ketterät menetelmät ovat Ketterän ohjelmistokehityksen julistuksen arvoille ja periaatteille perustuvia viitekehyksiä ja menetelmiä. Opinnäytetyössä esitellään tarkemmin ketterä viitekehys Scrum ja ketterä menetelmä Kanban.

Scrum on läpinäkyvyyteen perustuva ketterä viitekehys, jonka ytimessä on pieni, monitaitoinen ja itseohjautuva tiimi. Scrumin työkalujen avulla tiimi pystyy iteratiivisesti kehittämään ja toimittamaan suuren lisäarvon tuotteita luovasti ja tuottavasti.

Kanban on työnhallintamenetelmä, joka perustuu sekä itse prosessin eli työnkulun että varsinaisten työtehtävien visualisointiin. Kanbanin tavoite on prosessin pullonkaulojen havainnointi ja korjaaminen.

2 Ketterästä ohjelmistokehityksestä

Ketterässä ohjelmistokehityksessä oleellista on suhtautuminen jatkuvaan muutokseen: yhtäältä vaaditaan kykyä tehdä muutoksia itse, toisaalta tarvitaan myös kykyä reagoida toimintaympäristössä tapahtuviin muutoksiin. Ketterä ohjelmistokehitys on pohjimmiltaan paradigma, joka tarjoaa keinoja toimia menestyksekkäästi jatkuvasti muuttuvassa, turbulentissa ympäristössä. (Agile Alliance 2019a.)



Kuvio 1. Ketterä ohjelmistokehityssykli (Project-management.com 2019)

Termi itsessään on yläkäsite, joka sisältää useita Ketterän ohjelmistokehityksen julistuksen arvoille ja periaatteille perustuvia viitekehyksiä ja menetelmiä. Näille yhteistä on työskentelyyn osallistuvien ihmisten kommunikoinnin ja yhteistyön painottaminen, järjestäytyminen monitaitoisiin ja päätäntävaltaisiin pieniin tiimeihin sekä tapa kehittää ja toimittaa iteratiivisesti usein ja pieninä palasina kerrallaan. (kuvio 1; Agile Alliance 2019a.)

2.1 Ketterän ohjelmistokehityksen julistus

Ketterän ohjelmistokehityksen julistus (engl. The Agile Manifesto) luotiin Snowbirdin hiihtokeskuksessa Utahissa 11.-13.2.2001. Tuolloin 17 edustajaa eri ohjelmistokehityksessä käytettävistä (ja myös keskenään kilpailevista) viitekehyksistä ja menetelmistä kokoontui keskustelemaan vaihtoehtoisista ja paremmista tavoista tehdä ohjelmistokehitystä, jonka tuolloin katsottiin olevan aivan liian raskas ja dokumentointivetoinen prosessi. (Agilemanifesto.org 2001a.) Nämä keskustelut tiivistyivät Ketterän ohjelmistokehityksen julistukseen, jonka **arvot** ovat:

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. (Agilemanifesto.org 2001b.)

Jo varsinaista neljää arvoa edeltävä ingressivirke sisältää useita ketterille menetelmille keskeisiä dogmeja. Omasta ohjelmistokehityksen tekemisestä ja muiden auttamisesta puhutaan eksplisiittisesti me-muodossa, eli pohjimmiltaan kyse on ihmisten välisestä yhteistyöstä, jota työn kontekstissa tehdään koko ajan ja joka tasolla. Koska mikään ketterä viitekehys tai menetelmä ei ole sellaisenaan täydellinen ja jokaiseen käyttötapaukseen sopiva, parhaimmat työskentelytavat kyetään löytämään ainoastaan tekemällä asioita itse ja auttamalla muita, eli käyttämällä ja soveltamalla ketteriä menetelmiä jokapäiväisessä ohjelmistokehityksessä. Yhteistyön tekeminen, kommunikointi ja muiden auttaminen on tie lisääntyneeseen kokemukseen ja tietoon, joiden perusteella tekemistä, käytäntöjä ja menetelmiä voidaan alati kehittää ja jalostaa entistä paremmiksi. (Quickscrum 2016.)

Yksilöiden ja heidän välisen vuorovaikutuksensa painottaminen on kenties tärkein yksittäinen ketterän paradigman arvo. Kun tiimin yksilöt ymmärtävät toisiaan ja työskentelevät yhdessä päätettyjen tavoitteiden saavuttamiseksi, he kyllä pystyvät korjaamaan itse prosessissa ja työkaluissa ilmenevät ongelmat. (Software Testing Help 2019.) Prosessi ja työkalut ovat toki tärkeitä, mutta ne eivät saa olla itsetarkoituksellisia: niiden täytyy palvella tiimiä eikä päinvastoin, ne eivät saa pakottaa tiimiä työskentelemään jollain tietyllä tavalla vain siksi, että tämä tapa olisi yhteensopiva prosessin ja työkalujen kanssa. (KnowledgeHut 2019.)

Kuten yllä prosessien ja työkalujen tapauksessakin, niin ketterä paradigma ei suinkaan pidä dokumentaatiota arvottomana. Kattava dokumentaatio on kuitenkin toisarvoista toimivaan ohjelmistoon verrattuna, koska rajalliset resurssit kannattaa mieluummin suunnata lisäarvoa tuovan ohjelmiston kehittämiseen kuin siitä kertovan dokumentaation tuottamiseen. Ylenpalttinen dokumentaatio vie kaikkien aikaa, sekä dokumentaation tuottajien että sen lukijoiden. (Quickscrum 2016.) Yleensä toimivaa ohjelmistoa pääsee omin käsin ihan konkreettisesti käyttämään, jolloin ohjelmiston ja sen toiminnan ymmärtäminen on suu-

rimmalle osalle ihmisistä helpompaa kuin siitä kertovan monimutkaisen teknisen dokumentaation. (Ambler 2019.)

Asiakkaiden tarpeiden tyydyttäminen on tie menestykseen ja heiltä voi todennäköisimmin saada parhaat ideat ja ehdotukset menestyksekkään tuotteen rakentamiseen. Tuotteen ensisijaisina ostajina ja käyttäjinä he usein ovat myös tuotteen luontaisia asiantuntijoita, joten heidän mielipiteidensä ja ehdotustensa kuunteleminen tuotteen kehityksen aikana yleensä johtaa parempaan lopputulokseen. (KnowledgeHut 2019.) Kehitystiimiä kannattaa usein suojella suoralta asiakaskontaktilta, joten tuoteomistaja on avainasemassa asiakkaiden toiveiden välittämisessä kehitykseen. (Software Testing Help 2019.)

Viimeisin arvo oikeastaan tiivistää, mistä ketterässä ohjelmistokehityksessä pohjimmiltaan on kyse, eli jatkuvasta muutoksesta ja siihen reagoimisesta. Muutos on ohjelmistokehityksessä realiteetti, jota ei voi paeta, koska teknologia ja liiketoimintaympäristö muuttuvat jatkuvasti. Alati muuttuvassa ympäristössä on usein helpompaa vain hyväksyä asioiden muutokset ja valmistautua niihin kuin koittaa tehdä pitkän aikavälin muutokset kestäviä suunnitelmia. Projektisuunnitelma on toki ihan hyödyllinen dokumentti, mutta suunnitelman pitää olla muokattavissa ilmenevien muutosten mukaisesti. (Ambler 2019.) Kuuntelemalla asiakkaita kehitysprosessin aikana voi myös välttää isoja väärinymmärryksiä ja virheitä esim. puutteellisesti määriteltyjen käyttäjätarpeiden tai ominaisuuksien tapauksissa, jotka muuten voisivat aiheuttaa merkittäviä muutoksia aikatauluihin, kustannuksiin ja asiakastyytyväisyyteen. (Marjusaari 2017.)

Noudatamme seuraavia periaatteita:

1. *Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.*
2. *Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.*
3. *Toimitamme versioita toimivasta ohjelmistosta säännöllisesti, parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.*
4. *Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.*
5. *Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.*
6. *Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.*

7. *Toimiva ohjelmisto on edistymisen ensisijainen mittari.*
8. *Ketterät menetelmät kannustavat kestävään toimintatapaan.*
9. *Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.*
10. *Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.*
11. *Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.*
12. *Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.*
13. *Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.* (Agilemanifesto.org 2001c.)

Kuvio 2. Ketterän ohjelmistokehityksen julistuksen periaatteet (KnowledgeHut 2019)

Ketterän ohjelmistokehityksen julistuksen periaatteet ovat melko yksiselitteisiä (kuvio 2), ja ketterästi ajattelemaan tottuneille henkilöille suorastaan itsestään selviä. Periaatteissa on erittäin vahvoja keskinäisiä riippuvuuksia ja syy-seuraussuhteita, esimerkiksi:

- Tiimi on myötämielinen muuttuvien vaatimusten suhteen (periaate 2), jotka johtavat muutoksiin tuotteessa. Näistä muutoksista huolimatta ohjelmisto pysyy toimivana (periaate 7) ja uusia versioita toimitetaan nopeasti ja säännöllisesti (periaate 3). Kaikki edellä mainitut yhdessä johtavat tärkeimmän tavoitteen täyttymiseen, eli asiakkaan tarpeiden tyydyttämiseen (periaate 1).
- Kaikki työskentely tiimissä perustuu kunnioitukselle ja luottamukselle (periaate 5). Tiimi kommunikoi ja työskentelee kasvokkain (periaate 6) päivittäin koko projektin ajan (periaate 4). Tällainen toimintatapa johtaa parhaaseen lopputulokseen (periaate 12) ja tehokkuuteen (periaate 11) sekä ylipäätään mahdollistaa koko toimintatavan (periaate 10).

- Tiimi tarkastelee ja parantaa omaa toimintaansa jatkuvasti (periaate 13), minkä ansiosta toimintapa on kestävällä pohjalla (periaate 8), vaikka tulevaisuudessa tapahtuisi isojakin muutoksia (periaate 9).

Vaikka Ketterän ohjelmistokehityksen julistus onkin jo liki 20-vuotias, sen sisältö on edelleen kuranttia, nykyään kenties enemmän kuin koskaan aiemmin. (Drumond 2019.)

2.2 Ketterä viitekehys Scrum

Scrum on ketterä viitekehys, joka auttaa ihmisiä työskentelemään yhdessä. Se rohkaisee tiimejä oppimaan kokemustensa kautta, järjestäytymään itsenäisesti ongelmanratkaisutyöhön sekä tarkastelemaan ja parantamaan omaa toimintaansa jatkuvasti. Scrum on alun perin kehitetty erityisesti ohjelmistokehityksen tarpeisiin, mutta sen periaatteita ja työkaluja voidaan helposti soveltaa ja käyttää kaikenlaisessa ryhmätyöskentelyssä muillakin aloilla. (Atlassian 2019a.)

Hirotaka Takeuchi ja Ikujiro Nonaka kuvasivat ensimmäisinä Scrum-kehitysprosessin pääperiaatteet vuoden 1986 teoksessaan The New New Product Development Game, jossa he myös nimesivät konseptinsa Scrumiksi. Nykyisen Scrum-viitekehyksen kehittäjinä pidetään Jeff Sutherlandia, Ken Schwaberia, John Scumniotalesia ja Jeff McKennaa. (Kneafsey 2019.)

Scrumin ytimessä on pieni, tiivis, sopeutumiskykyinen ja monitaitoinen itseohjautuva tiimi. Scrumin työkalujen avulla tiimi pystyy ratkaisemaan monimutkaisia ongelmia kehittäessään suuren lisäarvon tuotteita luovasti ja tuottavasti. Scrum on kevyt ja yksinkertainen ymmärtää, mutta vaikea taitaa. Scrum tekee tuotehallinnan ja työmenetelmien suorituskyvyn näkyväksi, jotta tuotetta, tiimiä ja työskentely-ympäristöä voidaan jatkuvasti tarkastella parantaa. (Schwaber & Sutherland 2017, 3-4.) Läpinäkyvyys on erittäin tärkeää siksi, että arvoa optimoivat ja riskejä kontrolloivat päätökset perustuvat tarkastelussa havaittuun tuotosten tilaan. Näin ollen vain täydellisen läpinäkyvyyden vallitessa päätökset perustuvat todelliseen tietoon, kun taas heikon läpinäkyvyyden vallitessa päätökset voivat pohjautua virheellisille havainnoille ja täten vähentää arvoa ja kasvattaa riskejä. (Schwaber & Sutherland 2017, 17-18.)

Kuvio 3. Scrumin arvot (Scrum.org 2019b)

Scrumin arvot (kuvio 3) ovat sitoutuminen, rohkeus, keskittyminen, avoimuus ja kunnioitus. Scrumilla onnistuminen riippuu siitä, kuinka hyvin tiimin jäsenet pystyvät omaksumaan nämä viisi arvoa:

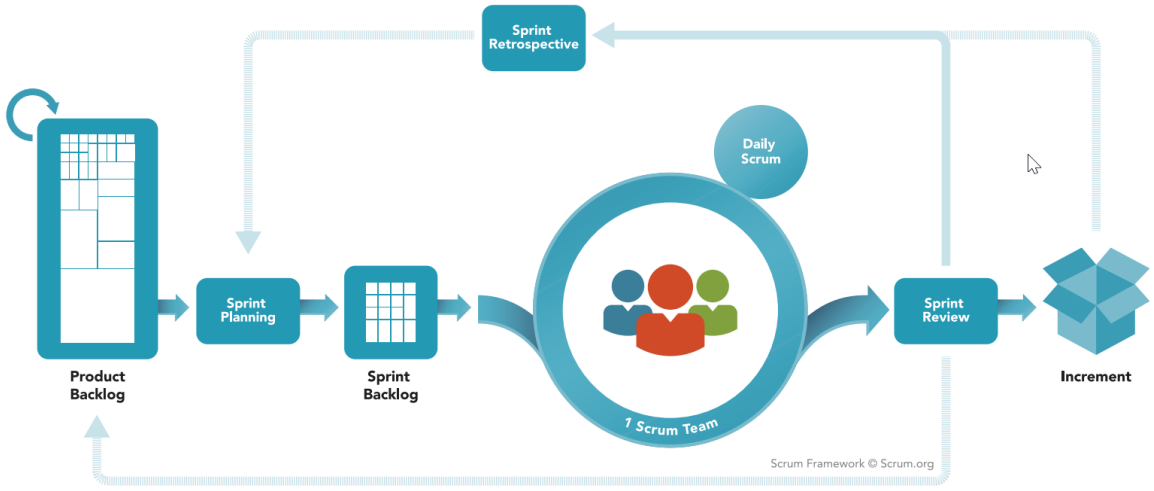
- Kaikilla on rohkeutta toimia oikein ja työstää vaikeita ongelmia.
- Kaikki keskittyvät pyrähdyksen työhön ja tavoitteisiin.
- Kaikki sitoutuvat henkilökohtaisesti pyrähdyksen työhön ja tavoitteisiin.
- Kaikki kunnioittavat toisiaan kyvykkäinä ja itsenäisinä ihmisinä.
- Tiimi ja sidosryhmät sopivat avoimuudesta liittyen kaikkeen työhön ja sen toteutuksen ongelmiin. (Schwaber & Sutherland 2017, 5.)

Scrum-tiimi koostuu tuoteomistajasta, kehitystiimistä ja scrummasterista. Tiimi on monitaitoinen, eli tiimissä on kaikki työn tekemiseen tarvittava osaaminen. Tiimi on myös itseohjautuva, eli se päättää itse, kuinka työ halutaan tehdä. Tiimi toimittaa tuotteita toistavasti ja lisäävästi, mikä varmistaa, että tuotteesta on aina saatavilla toimiva versio. (Schwaber & Sutherland 2017, 5-6.)

Tuoteomistaja on yksi henkilö, joka on vastuussa kehitystiimin työn tuloksena saatavan tuotteen arvon maksimoinnista sekä tuotteen kehitysjonon hallinnasta, priorisoinnista ja läpinäkyvyydestä. Tuoteomistajan päätökset ovat nähtävillä tuotteen kehitysjonon sisällössä ja järjestyksessä. (Schwaber & Sutherland 2017, 6.)

Kehitystiimi on monitaitoinen ja itseohjautuva joukko ammattilaisia, joka valitsee tuotteen kehitysjonosta kohdat pyrähdyksen tehtävälistaan ja tekee siitä julkaisukelpoisen inkrementin. Kehitystiimin sisällä ei ole alitiimejä ja vastuu kehityksestä kuuluu koko tiimille, vaikka tiimin sisällä voi tietysti olla erilaisia työn painopisteitä jäsenten osaamisen mukaan. Kehitystiimin optimaalinen koko on 3-9 henkilöä: liian pieni tiimi vähentää vuorovaikutusta ja johtaa pienempiin tuottavuushyötyihin, kun taas liian suuri tiimi lisää kompleksisuutta ja vaatii liikaa koordinointia. (Schwaber & Sutherland 2017, 6-7.)

Scrummaster on tiimin palveleva johtaja, joka vastaa Scrumin edistämisestä ja tukemisesta auttamalla kaikkia ymmärtämään sen käytännöt ja säännöt sekä fasilitoimalla sen tapahtumat. Scrummaster varmistaa, että koko Scrum-tiimi ymmärtää ja sitoutuu tavoitteisiin, ehdottaa tekniikoita kehitysjonon ja koko Scrum-prosessin hallintaan, valmentaa kehitystiimiä itseohjautuvuuteen ja moniosaamiseen, poistaa esteet kehitystiimin tieltä sekä suunnittelee Scrumin toteutusta organisaation sisällä (Schwaber & Sutherland 2017, 7-9.)



Kuvio 4. Scrumin tapahtumat, tuotokset ja työnkulku (Scrum.org 2019c)

Scrumin tapahtumat (kuvio 4) ovat pyrähdys, pyrähdyksen suunnittelupalaveri, päivittäispalaveri, pyrähdyksen katselmointi sekä retrospektiivi. Tapahtumia käytetään luomaan säännöllisyyttä ja minimoimaan muiden kuin Scrum-palaverien tarve, sekä tarjoamaan luonnollinen mahdollisuus tarkasteluun ja sopeuttamiseen. Kaikki tapahtumat ovat aikarajattuja, eli jokaisella on maksimipituus. (Schwaber & Sutherland 2017, 9.)

Pyrähdys on enintään kuukauden pituinen aikaraja, jonka sisällä tuotetaan julkaisukelpoinen ja toimiva inkrementti. Pyrähdyksillä on sama pituus koko kehityksen ajan ja uusi alkaa heti edellisen päätyttyä. Pyrähdyksen aikana ei tingitä laadusta eikä tehdä muutoksia,

jotka voisivat vaarantaa pyrähdyksen tavoitteet. Pyrähdyksen työlistaa voidaan tarkastella ja mukauttaa sitä mukaa kun opitaan lisää ratkaistavista ongelmista. Pyrähdykset lisäävät ennustettavuutta sekä rajoittavat riskin yhden pyrähdyksen pituuden kustannukseen. (Schwaber & Sutherland 2017, 9-10.)

Pyrähdyksen suunnittelupalaverissa koko Scrum-tiimi kokoontuu suunnittelemaan pyrähdyksen aikana tehtävän työn. Suunnittelun pitäisi vastata kahteen kysymykseen:

- *Mitä pyrähdyksessä tehdään?* Lähtökohtana on tuoteomistajan priorisoima tuotteen kehitysjono, viimeisin inkrementti, kehitystiimin kapasiteetti ja tiimin velositeetti. Kehitystiimi antaa ennusteen siitä, mitä pyrähdyksen aikana ehditään toteuttaa ja koko Scrum-tiimi työskentelee yhdessä ymmärtääkseen pyrähdyksen työmäärän ja tavoitteet.
- *Miten työ kannattaa toteuttaa?* Kun pyrähdyksen työmäärä ja tavoitteet on määritelty, niistä muodostetaan pyrähdyksen työlista eli suunnitelma inkrementin toteuttamisesta. Mikäli kehitystiimin mielestä inkrementissä on liian vähän tai liian paljon työtä, se voi neuvotella inkrementin sisällöstä tuoteomistajan kanssa. Kehitystiimi voi kutsua mukaan myös muita henkilöitä saadakseen esim. teknisiä tai liiketaloudellisia neuvoja. (Schwaber & Sutherland 2017, 10-12.)

Pyrähdyksen tavoitteet saavutetaan työlistassa olevien töiden tekemisellä. Työtä tehdessä ratkaistavista ongelmista opitaan koko ajan, joten on täysin mahdollista, että kesken pyrähdyksen työ paljastuu erilaiseksi kuin mitä kehitystiimi oletti – tällaisissa tapauksissa kehitystiimi voi neuvotella työlistasta uudelleen tuoteomistajan kanssa. (Schwaber & Sutherland 2017, 10-12.)

Päivittäispalaveri on enintään 15 minuutin pituinen palaveri, joka pidetään joka päivä samaan aikaan ja samassa paikassa (ja yleensä seisten). Kehitystiimi päättää palaverin avoimuudesta, ohjelmasta ja sen tavoista. Palaverissa tarkastellaan työn etenemistä ja tavoitteiden saavuttamista sekä tehdään yhteisiä päätöksiä. Palaverilla edistetään kommunikaatiota, päätöksentekoa, työn tarkastelua ja sopeutusta sekä vältetään muita palavereja. (Schwaber & Sutherland 2017, 12-13.)

Pyrähdyksen katselmointi pidetään pyrähdyksen lopussa ja siinä käydään läpi nyt kehitetty inkrementti ja tarvittaessa tarkastellaan ja muokataan tuotteen kehitysjonoa. Katselmointiin osallistuu koko Scrum-tiimi sekä mahdollisesti ne sidosryhmien edustajat, jotka tuoteomistaja on kutsunut tilaisuuteen. Tuoteomistaja selittää, mikä osa tuotteesta on valmista ja mikä ei. Kehitystiimi kertoo, mikä toteutuksessa meni hyvin, mitä ongelmia

kohdattiin ja miten ne ratkaistiin sekä esittelee valmiin työn, esim. uudet tuotteeseen kehitetyt ominaisuudet. Koko paikalla oleva ryhmä keskustelee seuraavista toimenpiteistä, markkinatilanteesta, budjetista, julkaisuaikataulusta ja potentiaalisista tuotteeseen tarvittavista uusista ominaisuuksista. Keskustelun perusteella tarkastellaan ja muokataan tuotteen kehitysjonoa, mikä on omiaan helpottamaan erityisesti seuraavia pyrähdyksen suunnittelupalavereita. (Schwaber & Sutherland 2017, 13-14.)

Pyrähdyksen retrospektiivi pidetään katselmoinnin jälkeen ja ennen seuraavan pyrähdyksen suunnittelupalaveria. Retrospektiivin tarkoituksena on tarkastella kulunutta pyrähdystä ja sen sujumista liittyen ihmisiin, yhteistyöhön, prosessiin ja työkaluihin, jotta voidaan tunnistaa sekä hyvin sujuneet asiat että tärkeimmät parannuskohteet, joiden toteuttamiseen luodaan suunnitelma seuraavan pyrähdyksen aikana. Vaikka Scrumissa parannuksia voidaan toteuttaa milloin tahansa, niin retrospektiivi tarjoaa muodollisen tilaisuuden tarkasteluun ja sopeuttamiseen. (Schwaber & Sutherland 2017, 14.)

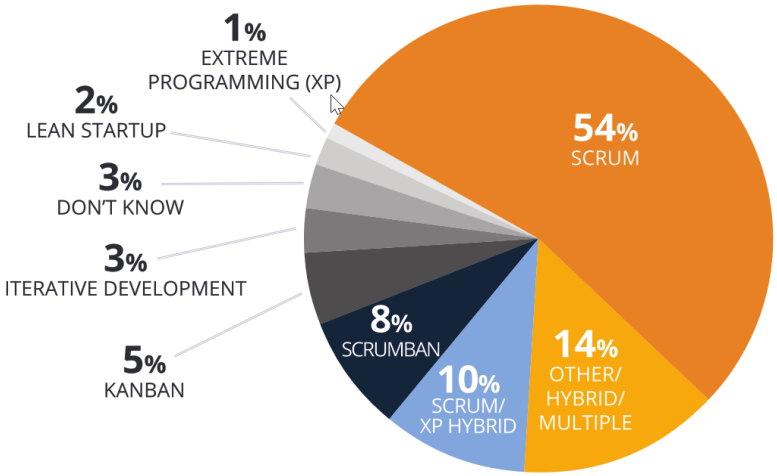
Scrumin tuotokset (kuvio 4) ovat tuotteen kehitysjono, pyrähdyksen tehtävälista ja tuoteversio eli inkrementti. Tuotokset kuvaavat työmäärää tai lisäarvoa ja niiden tarkoitus on lisätä läpinäkyvyyttä sekä tarjota mahdollisuuksia tarkastelulle ja sopeuttamiselle. (Schwaber & Sutherland 2017, 15.)

Tuotteen kehitysjono on tuoteomistajan priorisoima lista kaikista tuotteeseen tarvittavista vaatimuksista ja muutoksista. Se on olemassa niin kauan kuin tuotekin, eikä se ole koskaan valmis, vaan se muuttuu ja kehittyy dynaamisesti sen mukaan, kuinka tuote ja sen käyttöympäristö muuttuvat ja kehittyvät. Tuotteen kehitysjono listaa kaikki tulevissa inkrementeissä toteutettavat ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset. Kehitysjonon kohdat sisältävät järjestyksen, prioriteetin, kuvauksen ja usein myös työmääräarvion ja testikuvaukset. Yleensä korkealle priorisoidut kohdat ovat selkeämpiä ja yksityiskohtaisempia kuin matalammalle priorisoidut kohdat. Tuotteen kehitysjonon jalostaminen tarkoittaa yksityiskohtien, työmääräarvioiden ja keskinäisen järjestyksen lisäämistä. Kehitysjonon jalostaminen on toistuva prosessi, jonka toteutuksesta ja käytännöistä päätetään Scrum-tiimin sisällä. (Schwaber & Sutherland 2017, 15-16.)

Pyrähdyksen tehtävälista koostuu pyrähdykseen valituista tuotteen kehitysjonon kohdista sekä niiden toteuttamissuunnitelmasta. Tehtävälista on näkyvä ja reaaliaikainen kuva työstä, jonka kehitystiimi aikoo saada valmiiksi pyrähdyksen aikana; se konkretisoi ja tekee näkyväksi pyrähdykseen valitun työn ja mahdollistaa työn etenemisen seuraamisen. Tehtävälistan tulee olla riittävän yksityiskohtainen, jotta työn edistyminen voidaan havaita

kehitystiimin päivittäispalavereissa. Kehitystiimi muokkaa tehtävälistaa koko pyrähdyksen ajan sitä mukaa, kun työ edistyy, ratkaistavista ongelmista opitaan lisää tai havaitaan, että uutta työtä tarvitaan tehtävälistan toteuttamiseksi. (Schwaber & Sutherland 2017, 16-17.) Tuoteversio eli inkrementti on summa kaikista tuotteen kehitysjonon kohdista, jotka ovat valmistuneet kaikkien aiempien pyrähdysten aikana. Inkrementti on tarkasteltavissa oleva valmis kokonaisuus, jonka tulee olla käyttökelpoisessa kunnossa riippumatta siitä, päättääkö tuoteomistaja julkaista sen. (Schwaber & Sutherland 2017, 17.)

Scrum on nykyään ehdottomasti suosituin ketterä viitekehys, esim. uusimmassa State of Agile -raportissa (kuvio 5) 72 % vastaajista ilmoitti käyttävänsä Scrumia tai sen sisältävää hybridiä (CollabNet VersionOne 2019, 2).



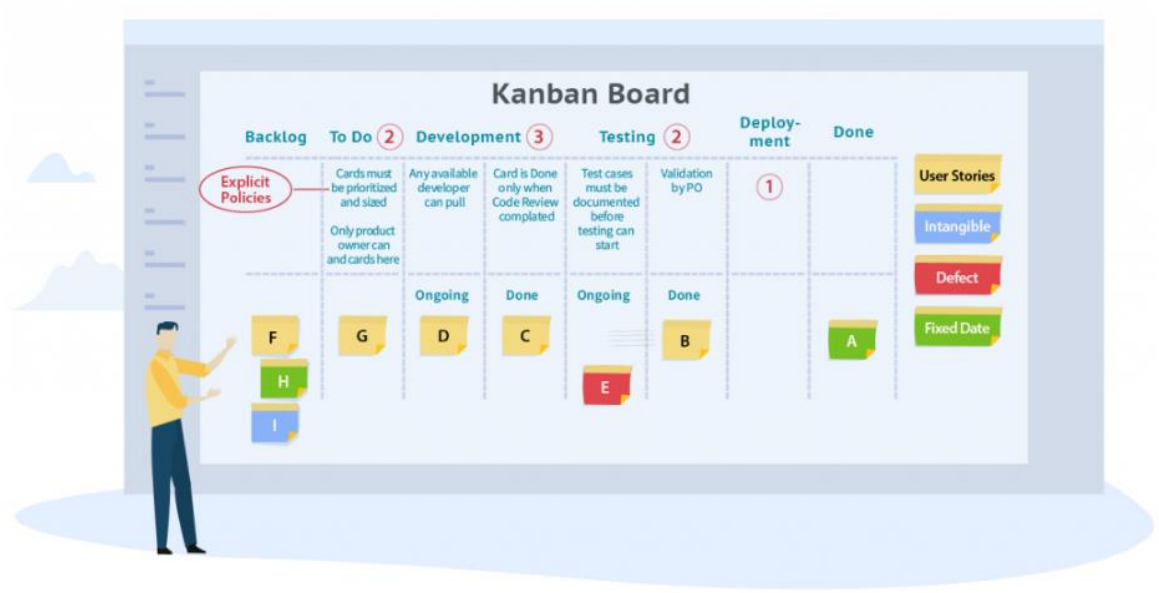
Kuvio 5. Ketterien menetelmien käyttö vuonna 2018 (CollabNet VersionOne 2019, 9)

Vankasta suosiostaan huolimatta Scrum ei aina ole oikea valinta. Ylläpitovaiheessa olevissa ohjelmistotuotteissa varsinaisia kehitystöitä voi olla hyvin vähän ja suurin osa töistä on suoraan asiakkaiden kohtaamista ongelmista johtuvia tehtäviä, esim. tukitikettejä. Tällaisessa tilanteessa Scrumin inkrementteihin on hyvin vaikea sitoutua, koska lisäarvon sijaan ne tuottavat pikemminkin lisää ongelmia. Tällaiseen käyttöön Kanban olisikin parempi valinta. (Kniberg 2009, 3.)

2.3 Ketterä menetelmä Kanban

Kanban on työnhallintamenetelmä, joka perustuu sekä itse prosessin eli työnkulun ja varsinaisten työtehtävien visualisointiin. Kanbanin tavoite on prosessin pullonkaulojen jatkuva havainnointi ja korjaaminen, minkä seurauksena työnkulku saadaan sujumaan optimaalisesti. (Digité 2019.)

Nykyisin ohjelmistokehityksessä hyvinkin suosittu Kanban perustuu Toyotan 1940-luvulla toteuttamiin prosessiparannuksiin, joissa sovellettiin ruokakauppojen käyttämiä varaston-hallinnan ja hyllytäytön mekanismeja autotehtaiden tuotantolinjoille. Kun tuotantolinjalla tarvitut osat olivat vaarassa loppua, varastoon lähetettiin kortti (japaniksi kanban), jossa kerrottiin mitä osia varastosta piti toimittaa, jotta tuotanto voisi jatkua katkotta. (Atlassian 2019b.)



Kuvio 6. Kanban-taulu (Digité 2019.)

Kanbanin käytännöt ovat:

- *Tee työnkulku näkyväksi*. Kaikkein tärkeintä on ymmärtää, mitä kaikkea muutos-pyynnön toteuttaminen valmiiksi tuotteeksi asti vaatii. Työnkulku visualisoidaan Kanban-taululla (kuvio 6), jonka sarakkeet esittävät työn eri vaiheita ja kortit eri työvaiheiden läpi kulkevia yksittäisiä työtehtäviä. Kortti kulkee taululla vasemmalta oikealle alkaen kehitysjonosta, kulkee kaikkien työvaiheiden läpi ja päätyy lopuksi valmiiden sarakkeeseen.
- *Rajoita käynnissä olevan työn määrää*. Koska fokuksen vaihtaminen kesken käynnissä olevan työn johtaa laadun heikkenemiseen ja tehottomuuteen, käynnissä olevan työn määrän rajoittaminen on yksi Kanbanin keskeisimmistä periaatteista. Rajoittaminen toteutetaan määrittelemällä sarakkeelle maksimiarvo sille, kuinka monta korttia sarakkeessa saa kerrallaan olla. Mikäli sarake on täynnä, siihen voidaan tuoda uusi kortteja vasta kun joku nykyisistä korteista siirretään seuraavaan sarakkeeseen.

- *Hallinnoi työnkulkua*. Tavoitetila Kanban-implementaatiossa on, että työt kulkevat nopeasti ja joustavasti sarakkeesta toiseen eli alusta loppuun. Mikäli yksi sarake täyttyy jatkuvasti, kyseessä on pullonkaula, joka Kanbanin ansiosta on helposti havaittavissa ja ratkaistavissa esim. siirtämällä lisää työvoimaa ruuhkautuvaan vaiheeseen.
- *Tee prosessikäytännöistä eksplisiittisiä*. Parantaminen on mahdotonta ilman asioiden perinpohjaista ymmärtämistä, joten käytetyn prosessin ja sen tavoitteiden tulee olla selkeästi määriteltyjä, hyvin dokumentoituja ja organisaatioon jalkautettuja.
- *Käytä palautesilmukoita*. Tärkein yksittäinen tilaisuus palautteen esittämiseen ja siitä keskustelemiseen on korkeintaan 15 minuutin pituinen päivittäispalaveri, joka pidetään joka päivä Kanban-taulun edessä. Kanban tunnistaa myös muita säännöllisiä kokouksia, joiden käytöstä ja frekvenssistä päätetään tiimin kesken.
- *Toteuta parantaminen yhteistyössä*. Jatkuva parantaminen on mahdollista vain organisaatioissa, joissa on yhteinen visio tavoitteista ja paremmasta tulevaisuudesta sekä kollektiivitason ymmärrys keinoista, joiden avulla näihin pääsee. (Kanbanize 2019.)

Kanban sopii erityisen hyvin tiimeille, jotka joutuvat jatkuvasti ottamaan vastaan suuren määrän muutospyyntöjä, joiden prioriteeteissa ja työmäärissä on isoja vaihteluita (Rehkopf 2019); tällaisessa tilanteessa Scrumin pyrähdyssykleihin on hyvin vaikeaa sitoutua. Menetelmänä Kanban sopii erittäin hyvin käytettäväksi myös Scrumin rinnalla, kuten 43 % vuoden 2015 State of Scrum -tutkimuksen vastaajista ilmoitti tekevänsä (Scrum Alliance 2015, 3).

2.4 Ketterä ohjelmistokehitys nyt ja tulevaisuudessa

Monimutkaisten tietojärjestelmien kehittäminen on erittäin vaikeaa työtä, koska etukäteen on yleensä mahdotonta tietää, mikä on paras lopullinen ratkaisu. Paras lähestymistapa tällaisen haasteeseen on empiirinen malli, eli pikkutarkan etukäteen tehdyn suunnitelman – joka on itse asiassa vain arvaus – sijaan visioidaan tavoitteet ja lähdetään rakentamaan ja toimittamaan niitä inkrementaalisesti pala kerrallaan ja iteratiivisesti työn aikana opitun perusteella koko ajan täsmentäen. Ketterä lähestymistapa on yksinkertaisesti sekä tehokkain menetelmä kompleksisten suuren lisäarvon tuotteiden kehittämiseen että paras strategia niiden sisältämien moninaisten riskien hallitsemiseen. (Marjusaari 2017.)

Ketterän paradigman noudattaminen takaa parhaat edellytykset toimimiseen nykypäivän hektisillä markkinoilla, joilla yritysten pitää reagoida hetkessä muuttuviin markkinoiden

oikkuihin, viraali-ilmiöihin ja kulutustrendeihin. Suurin osa menestyksekkäistä ohjelmistoyrityksistä, mm. maailman viisi markkina-arvoltaan suurinta yritystä Amazon, Apple, Facebook, Google ja Microsoft, toteuttavat ketterää paradigmaa. (Denning 2018.) Samalla kun ympäröivä maailma muuttuu, myös ketterät menetelmät ja niiden käyttötavat muuttuvat. Oleellista ei olekaan se, mitä nimenomaisia ketteriä menetelmiä käytetään, vaan se, että ketterän paradigman periaatteet ymmärretään ja niitä toteutetaan. (Denning 2019.)

Tarkastelkaamme esimerkiksi kuvitteellista ohjelmistotuotetta, joka on menestyksekkäästi kehitetty Scrumilla kypsäksi tuotteeksi. Kypsälle ohjelmistotuotteelle on ominaista, että varsinaisia isoja kehitystöitä ei enää tehdä, mutta maksavilta asiakkailta tulee jatkuvasti suuresti vaihtelevia muutospyyntöjä. Scrum sopii tällaiseen käyttöön huonosti, koska se perustuu inkrementtien etukäteissuunnitteluun. (Kniberg 2009, 3.) Toinen eri lähtökohdista juontuva, mutta samantyyppiseen ongelmaan johtava esimerkki on ohjelmistotuotteen jatkuva julkaisu, eli koodimuutoksia saatetaan tehdä ja viedä tuotantoon satoja kertoja päivässä, jolloin inkrementtien suunnittelu on kerta kaikkiaan mahdotonta (Karaivanov 2019).

Molemmissa tapauksissa looginen ratkaisu olisi yksinkertaisesti vaihtaa työkalut paremmin sopiviin, eli Scrumista Kanbaniin. Trendi on nähtävissä hyvin selvästi vertailemalla kahta viimeisintä State of Scrum -tutkimusta (taulukko 1), jossa Scrumin eksklusiivinen käyttö on laskenut merkittävästi vain noin kolmessa vuodessa.

Taulukko 1. Kahden viimeisimmän State of Scrum -tutkimuksen vertailu (Scrum Alliance 2015; Scrum Alliance 2018)

	2015	2018
Käyttää yksinomaan Scrumia	42 %	16 %
Scrum parantaa työelämän laatua	87 %	85 %
Projektien keskimääräinen onnistumisaste	62 %	63 %
Aikovat jatkaa Scrumin käyttöä	95 %	97 %

Ketterää paradigmaa vastaan esitetty kritiikki johtuu usein siitä, että sen periaatteet on ymmärretty väärin. Paradigma ei esim. ole missään nimessä dokumentaatiota vastaan, vaan se pyrkii priorisoimaan toimivan ohjelmiston dokumentaation edelle, ja yleensä toimiva ja asiakkaan käytettävissä oleva ohjelmisto vielä vähentää merkittävästi dokumen-

taation tarvetta. Ketterä toimintatapa ei myöskään suora tie onneen, pikemminkin päinvastoin: toimintatavan jalkauttaminen ja käyttäminen vaativat erittäin runsaasti suunnitelmallisuutta ja kurinalaista työtä. Epäonnistuminen ketterästi toimien on täysin mahdollista siinä missä perinteisemmillä menetelmilläkin, mutta ketterän toimintatavan läpinäkyvyys ja nopea iteraatiosykli yleensä todistaa epäonnistumisen nopeammin – mikä ei ole ollenkaan huono asia. (Agile in a Nutshell 2019.) Yksi merkittävä ongelmakohta on arkkitehtuurin valinta ja suunnittelu, mikä on loogista tehdä etupainotteisesti ja näin ollen soveltuu vesiputousmalliin myötäsyntyisesti; arkkitehtuurisuunnittelu ketterässä ympäristössä ei ole suinkaan mahdotonta, mutta siihen tulee kiinnittää erityistä huomiota (Morris 2018).

On tärkeää kuitenkin muistaa, että ketteryys sinänsä ei ole mikään taikakeino. Koska ketteryys on jo pitkään ollut muodikasta, monet yritykset kertovat hyvin mielellään markkinoille olevansa ketteriä. Ketterän paradigman hyötyjä ei kuitenkaan saavuteta ilmoitusluontoisesti, vaan sen periaatteet pitää todella sisäistää ja jalkauttaa yritykseen. Muutos on valtava eritoten isoille, perinteisempiä ohjelmistokehitysmenetelmiä käyttäville yrityksille, ja sen toteuttaminen vaatii kurinalaista suunnittelua ja sitoutumista yrityksen joka portaalla. Vastaavasti myös riskit ovat valtavia ja niiden toteutuminen hyvin todennäköistä. (AgencyAgile 2019.) Mikäli ketterän paradigman käyttöönotto tehdään puolivillaisesti eikä sen periaatteita ja toimintatapoja todellakin sisäistetä ja toteuteta, on hyvin todennäköistä, että kokonaisuutena katsottuna tilanne vain huononee entiseen verrattuna (Edwards 2017.)

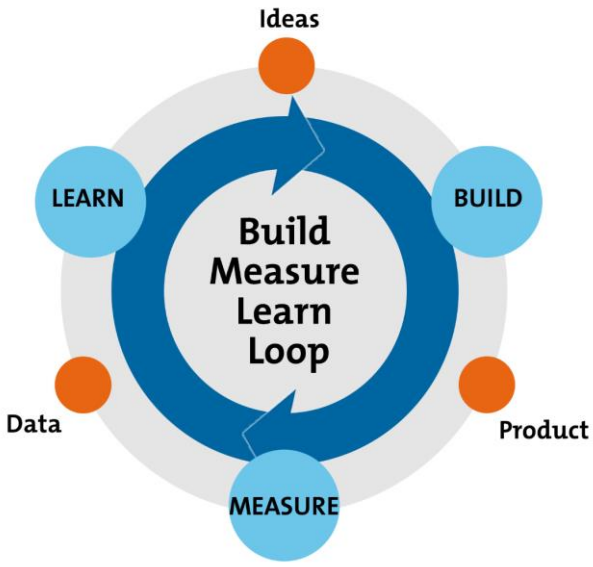
3 Tuotteesta ja sen kehitysmallista

Tuotteen elinkaari alkoi hyvältä kuulostavasta ideasta, jossa keskeistä oli kokonaan uudenlainen lähestyminen keskitettyyn käyttöoikeushallintaan: käyttäjien oikeustasojen ja sallittujen kohteiden hallinnointi pohjautuisi luontaisesti tietoturvallisiin, erittäin lyhytaikaisiin SSH-sertifikaatteihin. (SSH Communications Security Oyj 2019a.) Tällä lähestymistavalla voitaisiin välttää pysyvien SSH-avaimien sisäsyntyiset ongelmat, kuten käsitöinä tehtävä ylläpito, avainten holvauksen ja kierrättämisen vaikeudet sekä hirvittävät riskit kadonneiden tai kaapattujen pääkäyttäjätason avainten kanssa. (SSH Communications Security Oyj 2019b.) Lähestymistapa oli yhteensopiva myös modernin Zero Trust -arkkitehtuuriajattelun kanssa, jossa jokaisen käyttäjän jokainen operaatio verifioidaan joka kerta. (Fybish 2019.)

Idean validoimiseksi tuotteesta päätettiin ensin toteuttaa tekniikkademo, jolla haluttiin testata näitä uusia käyttöoikeushallinnan teorioita käytännössä. Tekniikkademo täytti sille asetetut odotukset, joten Yhtiössä päätettiin siirtyä seuraavaan vaiheeseen.

3.1 Pienin toimiva tuote

Pienin toimiva tuote eli MVP (engl. Minimum Viable Product) on versio tuotteesta, joka mahdollistaa maksimaalisen validoidun oppimisen minimaalisella vaivannäöllä (Ries 2009).



Kuvio 7. The Build-Measure-Learn Feedback Loop (Mindtools 2019)

MVP painottaa oppimisen merkitystä tuotekehityksessä ja se on erittäin hyödyllinen konsepti uuden tuotteen liiketoimintapotentiaalia arvioitaessa. MVP on mahdollisimman pienellä vaivalla toteutettu versio tuotteesta, joka mahdollistaa täyden Build Measure Learn -syklin kierroksen (kuvio 7), eli idean pohjalta rakennetaan tuote, jonka perusteella tehdään mittaukset ja saadaan niistä dataa, jonka perusteella opitaan jotain, esim. ostaisiko asiakas tuotteen vaiko ei. (Ries 2011, 81-83.) Tekemällä Tuotteesta MVP:n ja antamalla sen hyvin tuntemilleen pilottiasiakkaille Yhtiö halusi testata tuotteen kaupallista potentiaalia konkreettisesti ennen sitoutumista massiiviseen tuotekehityshankkeeseen. (Pendolin 2018.)

MVP-vaiheessa Tuotteen kehitystiimi koostui kolmesta ohjelmistoarkkitehdistä, joista kaikkia on haastateltu tähän opinnäytetyöhön (liite 2). Arkkitehdit ovat työskennelleet yhdessä useita vuosia ja tuntevat toisensa erittäin hyvin, joten tuotteen kehitysmalli oli erittäin vapaamuotoinen ja kevyt ja testaus oli minimaalista. Mitään tämän muodollisempaa ei yksinkertaisesti tarvittu, koska kehitystiimi oli niin verrattoman kokenut ja vuosien saatossa yhteen hitsautunut.

Jo MVP-vaiheessa tehtiin koko Tuotteen elinkaareen ja sen kehittämiseen vaikuttavia, erittäin tärkeitä valintoja. Aiemmin Yhtiön ohjelmistojen julkaisusykli on ollut verrattain pitkä, vuodessa on tyypillisesti julkaistu 1-2 major-versiota. Pitkään julkaisusykliin liittyy orgaanisesti useita ongelmia: pitkän aikavälin suunnitelmiin on vaikea sitoutua, suunnitelmat muuttuvat myynniltä ja asiakkailta tulevien toivomusten mukaan ja asiakkaat joutuvat odottamaan uusia ominaisuuksia ja bugikorjauksia tarpeettoman kauan. Tästä syystä jo Tuotteen arkkitehtuurivalinnoissa pidettiin mielessä, että Tuotteen on sovelluttava inkrementaaliseen kehitykseen, automatisoituun testaukseen ja usein tapahtuvaan julkaisuun. Tuote onkin toteutettu modulaarisella microservice-arkkitehtuurilla, joten se ei kärsi monoliittiselle arkkitehtuurille tyypillisistä ongelmista, kuten vaikeasta testattavuudesta, päivitettävuudestä ja regressioista. (Kharenko 2015.)

MVP-vaiheen päätteeksi tuote esiteltiin Yhtiön hyvin tuntemille pilottiasiakkaille ja vastaanotto oli niin positiivinen, että Yhtiö päätti aloittaa jatkokehityksen kaupalliseksi tuotteeksi.

3.2 Kehitysmallin päivittäminen

Kun päätös jatkokehityksestä oli tehty, kehitystiimiin siirrettiin lisää kehittäjiä muista tuotetiimeistä ja rekrytoitiin kokonaan uusia kehittäjiä. Kehitystiimin kasvettua nopeasti huomat-

tiin, että kehitystyön järjestelmällisyydessä oli selkeitä puutteita, siihen kaivattiin mm. lisää suunnitelmallisuutta ja struktuuria. Huomattiin, että itse asiassa koko kehitysmalli kaipasi päivittämistä.

Yhtiö päätti perustaa kokonaan uuden liiketoimintayksikön ja siirtää Tuotteen tämän yksikön alaisuuteen. Yksiköllä oli sekä valta että vastuu lähes kaikista Tuotteen kehitykseen, lanseeraamiseen, markkinointiin ja myyntiin liittyvistä asioista. Kehitysmallin päivitystä ja tiimin organisointia varten liiketoimintajohtaja palkkasi hankepäällikön ja antoi tälle hankkeen tavoitteet ja reunaehdot, joista tärkeimpiä olivat hyvän laadun toimittaminen ja usein tapahtuva julkaisu.

Tiimi kävi kehitysmallin parannustalkoisiin hankepäällikön johdolla. Aluksi identifioitiin useita kohtia, joihin haluttiin selkeitä parannuksia. Kun nämä kohdat oli tiedostettu, määriteltiin millainen prosessi toimisi tämän tiimin kanssa parhaiten; ei siis haluttu tuoda yhtä ja tiettyä ketterää kehitysmenetelmää käyttöön sellaisenaan, vaan haluttiin yksinkertaisesti poimia toimivat menetelmät ja käytännöt välittämättä siitä, missä ketterässä viitekehyksessä tai menetelmässä ne nyt sattuvatkaan olemaan. Varsinainen muutos aloitettiin kevyesti ja ensimmäisen kolmen kuukauden aikana oli vain tiimin kesken tehty lupaus siitä, että kaikki työ tehdään muille tiimin jäsenille näkyväksi, mikä koordinoitiin aina päivittäispalavereissa.

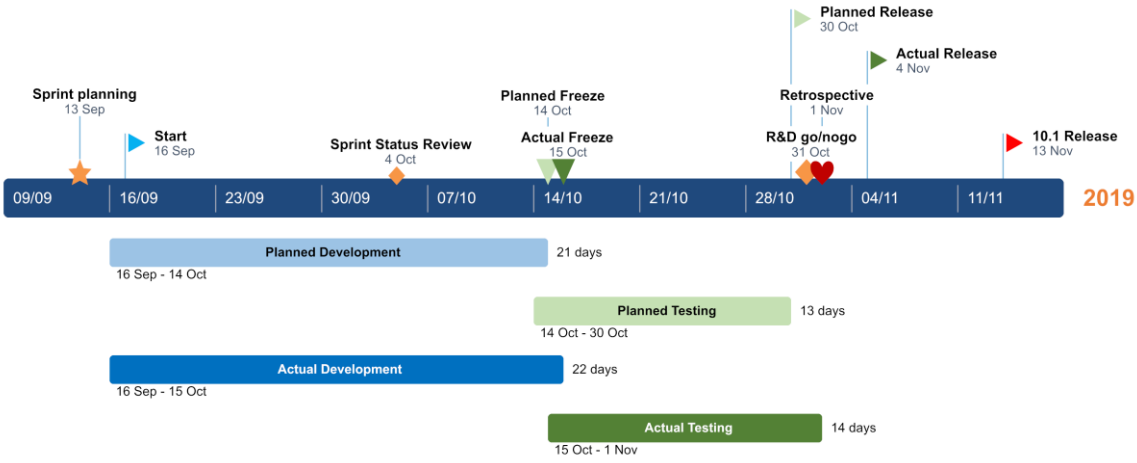
Vähitellen otettiin käyttöön suurin osa Scrumin käytännöistä ja siirryttiin käyttämään neljän viikon pituista pyrähdystä, joka jaettiin kolmen viikon kehitysjaksoon ja viikon testausjaksoon. Osa Scrumin vahvasti suosittelemista käytännöistä hylättiin suoraan: esim. työtikettejä ei ehdoin tahdoin pilkota päivän mittaisiin palasiin, vaan järkeviin ja luonnollisiin kokonaisuuksiin, koska tiimi koki pilkkomisen tuottavan merkittävästi tarpeettomia tikettejä ja turhanpäiväisiä transaktiokustannuksia. Samalla käyttöön otettiin myös uusi tikettijärjestelmä Jira ja sitä tukeva organisaatiowikiohjelmisto Confluence, jotka optimoitiin tiimin työnkuluille sopivaksi ja jossa käytettiin työn visualisoimiseksi Kanban-taulua. Näiden tietojärjestelmien käyttöönotto edesauttoi suuresti uuden kehitysmallin implementoimista, koska niiden työnkulut tukivat tavoiteltuja ketteriä menetelmiä saumattomasti ja työn visualisointi konkretisoi pyrähdyksen tehtävälistan kouriintuntuvasti.

Varsinainen ohjelmointityö toteutetaan jatkuvalla integraatiolla ja julkaisuputkella, jonka testit pyritään automatisoimaan niin pitkälle kuin mahdollista. Koodikatselmointi päätettiin tehdä vähintään kahden silmäparin käytännöllä, eli kukaan kehittäjä ei laita omaa ohjelmakoodiaan suoraan versionhallintajärjestelmän aktiiviseen kehityshaaraan, vaan pyytää

jotain työtoveriaan katselmoimaan ja liittämään koodinsa, mikäli siinä ei ole huomauttamista.

Käytäntöjä tarkasteltiin ja iteroitiin koko ajan ja muutoksia tehtiin aina kun ne katsottiin tarpeellisiksi. Isoista muutoksista mainittakoon pyrähdyksen pituuden muutos: ensin huomattiin, että viikon testausjakso on liian lyhyt, joten jaksoa pidennettiin kahteen viikkoon ja itse pyrähdyksiä näin ollen viiteen viikkoon. Myöhemmin tehtiin vastaava huomio kehitysjakson lyhyydestä, joten sitäkin jatkettiin viikolla, jolloin pyrähdyksien ohjeellinen pituus kasvoi kuuteen viikkoon, mikä on merkittävästi Scrum-oppaiden suosittelemaa korkeintaan neljää viikkoa pidempi.

Pienemmistä muutoksista mainittakoon muutokset retrospektiivien järjestämisessä. Aluksi retrospektiivit järjestettiin aina pyrähdyksen lopussa, koska niissä haluttiin keskustella kehitysmallissa tehdyistä muutoksista ja seurata muutosten vaikutuksia. Jossain vaiheessa pakollisista retrospektiiveistä luovuttiin, koska tässä vaiheessa tiimin jäsenet olivat jo oppineet puhumaan epäkohdista vapaasti, joten retrospektiivi päätettiin järjestää vain, mikäli sellainen katsottiin ehdottoman tärkeäksi – yleensä näin tehtiin vain silloin, kun pyrähdyksessä tai tuoteversion julkaisussa oli ollut ongelmia, esim. merkittävä myöhästyminen aikataulusta tai julkaisun jälkeen huomattiin, että Tuotteessa on regressio tai tietoturvaongelma. Aivan hiljattain kuitenkin havaittiin, että Scrum-oppaidenkin vahvasti suosittelema formaali retrospektiivi on erittäin hyödyllinen tilaisuus, joten niiden järjestäminen jokaisen pyrähdyksen lopussa otettiin takaisin kehitysmallin normaaliin ohjeistukseen.



Kuvio 8. Nykyisen kehitysmallin mukainen pyrähdys ja sen tapahtumat (suunnitellut ja toteutuneet eriteltyinä)

Kuviossa 8 on kaavio viimeisimmän kehitysmallin mukaisesta pyrähdyksestä, jonka kronologisesti katsottuna olisi tarkoitus sujua näin:

1. Pyrähdyksen suunnittelu on pyrähdyksen aloitusta edeltävällä viikolla. Koko Scrum-tiimi osallistuu kasvokkain pidettävään tapaamiseen, jossa esitellään pyrähdyksen tavoitteet, keskustellaan suunnitelmasta ja jaetaan pyrähdyksen työtiketit niin pitkälle kuin se on mahdollista. Näin muodostetaan pyrähdyksen tehtävälista. Kehitystiimille näytettävä suunnitelma on jo melko valmiiksi ajateltu, koska arkkitehdit, tuoteomistaja ja scrummaster ovat tehneet tarvittavan määrän esisuunnittelua, jotta tämä suunnittelupalaveri sujuisi helposti ja koko Scrum-tiimi pääsisi yhteisymmärryksen helposti. (ScrumDesk 2019).
2. Päivittäispalaveri on joka päivä ja se kestää korkeintaan 15 minuuttia. Päivittäispalaveri on avoin kaikille ja siinä käsitellään töiden edistyminen sekä tehdään tärkeät tiimitason päätökset.
3. Kehitysjakso alkaa seuraavana maanantaina ja kestää yleensä vähintään neljä täyttä viikkoa. Jakson aikana löydetään bugeja koodista ja mikäli niitä pidetään riittävän vakavina (= ohjelmiston julkaisun estävinä), ne lisätään suoraan pyrähdyksen tehtävälistaan pakollisiksi työtiketeiksi. Jakson puolivälissä järjestetään pyrähdyksen tilanteen katselmointi, jossa tarkistetaan, onko pyrähdyksen suunnitelma vielä kurantti, vai pitääkö tiimin ja sidosryhmien valmistautua viivästyksiin. Kehitysjakson aikana kehittäjät kokoontuvat viikoittain omaan arkkitehtuuripalaveriinsa, jossa käsitellään teknisiä ongelmia ja kysymyksiä. Kehitysjakso päättyy neljän täyden viikon jälkeen, kun tiimi yhteisesti päättää päivittäispalaverissa testausjaksoon siirtymisestä.
4. Testausjakso alkaa heti kun tiimi on niin päättänyt ja sen pituus pyritään pitämään vähintään kahdessa viikossa, mikäli se vain suinkin on mahdollista. Testaukseen osallistuvat sekä nimetyt testaajat että Tuotteen kehittäjät, mikä on jokseenkin poikkeuksellista. Testausjakson aikana Tuotteen koodiin tehdään minimaalinen määrä muutoksia, eli käytännössä vain sen verran kuin testaamisessa löytyneiden bugien korjaaminen vaatii. Testausjakso päättyy, kun kaikki vakavat bugit on korjattu, kaikki testit menneet läpi ongelmitta ja kun testauspäällikkö on tyytyväinen – testauksen päättyminen ei käytännössä siis ole tiimitason päätös.
5. Testauksen päätyttyä Scrum-tiimi kokoontuu taas kasvokkain ja äänestää ohjelmistoversion julkaisemisesta. Tämän palaverin asialista on formalisoitu ja ennen äänestämistä tiimi keskustelee ja käy läpi pyrähdyksen tavoitteet, tekemättä jääneet ja/tai uudelleen aikataulutetut tiketit, pyrähdyksen aikana löydetyt bugit ja julkaisutiedot (engl. release notes) sekä käsittelee ja päivittää riskimatriisin. Tarkka äänestystulos kirjataan kokouksen muistioon. Varsinaisen päätöksen ohjelmiston

julkaisusta tekee sidosryhmäkokous, jonka tuoteomistaja järjestää ja jossa scrummaster esittelee tämän kokouksen tulokset. Mikäli sidosryhmäkokous päättää julkaista, julkaisu tehdään Yhtiön julkaisupolitiikan mukaisesti ja sen käytäntöjä noudattaen.

6. Retrospektiivi on myös formalisoitu, tosin se on haluttu pitää melko kevyenä. Retrospektiivissä Scrum-tiimi tapaa jälleen kasvokkain ja keskustelee vapaamuotoisesti pyrähdyksestä: mikä meni hyvin, mikä huonosti, pitäisikö kokeilla jotain uutta ja onko kaikki sovitut tehtävät tehty. Formaalissa tutkiskelussa kirjataan ylös ne toimenpiteet, jotka täytyy aloittaa tai lopettaa ja ne, joiden tekemistä tulee jatkaa. Alkuperäinen lista on peräisin edellisestä retrospektiivistä, joten samalla tulee seurattua, onko kaikki sovitut asiat tehty ja toteutettu. Asiat ja toimenpiteet kirjataan ylös ja ne käsitellään viimeistään seuraavassa retrospektiivissä.

Tällä hetkellä identifioituja parannuskohteita on kaksi: kehitysjonoa pitäisi jalostaa hieman useammin ja suunnitelmia pitäisi pystyä tekemään muutaman pyrähdyksen päähän, kun tällä hetkellä oikeastaan eletään vain yksi pyrähdys kerrallaan. Nämä parannuskohteet ovat myös toisistaan riippuvaisia: suunnittelemalla tulevat pyrähdykset kattavammin tiedetään myös paremmin, millaisia kehityskohteita Tuotteeseen on tulossa lähiaikoina, mikä luonnollisesti ohjaa myös kehitysjonon jalostamista.

Kokonaisuutena kehitysmallin muutos oli todella iso ponnistus, jossa kaikkinensa meni melkein vuosi. Suurin vastuu hankkeen toteuttamisessa ja kehittämisessä oli ehdottomasti hankepäälliköllä, joka toisaalta palkattiin Yhtiöön nimenomaisesti tämä asia hoitamaan. Muutos ei olisi tietenkään onnistunut ilman sitoutunutta ja motivoitunutta tiimiä eikä avointa kommunikointia, jossa ei koskaan ollut mitään ongelmia, koska tiimi on niin tottunut avoimeen ja aktiiviseen keskusteluun Slackin välityksellä (Mehandru 2019).

Tulevaisuuden isoimmat haasteet liittyvät pilveen ja SaaS-malliin siirtymiseen ja sen mukanaan tuomaan jatkuvaan julkaisuun, mikä nostaa tuotantoon viennin ja aivan erityisesti testauksen vaikeustasoa massiivisesti (Huotarinen 2016). Koska asiakkaat voivat ostaa pilvikapasiteettia automaattisesti aina tarpeen mukaan, Tuotteen täytyy tukea skaalautuvuutta orgaanisesti, mikä sekin nostaa sekä arkkitehtuurisuunnittelun että varsinaisen ohjelmoinnin vaikeustasoa merkittävästi (Concepta 2019).

4 Tutkimuksen toteutus

4.1 Tutkimuksen tavoite ja tuotos

Tutkimuksen tavoite on selvittää:

1. Syyt, joiden takia kehitysmallin muutos haluttiin tehdä.
2. Keinot, joiden avulla muutos toteutettiin.
3. Lopputulos, eli kuinka muutos onnistui ja kuinka se vaikuttaa tiimiin ja Tuotteeseen.

Tutkimuksen tuotoksena syntyy dokumentointi yllä olevasta. Dokumentti on vapaasti käytettävissä ja sitä voi sellaisenaan hyödyntää vastaavien hankkeiden suunnitteluun ja toteuttamiseen. Opinnäytetyö hyödyttää myös minua itseäni suoraan ja merkittävästi, koska työskentelen parhaillaan Tuotteen kehityspäällikkönä ja scrummasterina.

4.2 Tutkimusmenetelmä

Tutkimus kuuluu laadullisiin tutkimuksiin. Toteutan tutkimuksen puolistrukturoituina haastatteluina valitulle joukolle, jolle esitetään samat haastattelukysymykset (liite 1), jotka toimivat keskustelun runkona (Saaranen-Kauppinen & Puusniekka 2006).

Valitsen haastateltaviksi (liite 2) Tuotteen parissa erilaisissa positioissa työskenteleviä henkilöitä, joilla on riittävän pitkä työkokemus ohjelmistokehityksestä, koska haluan heidän pystyvän keskustelemaan aiheesta monista eri näkökulmista.

Nauhoitan ja puran haastattelut jälkikäteen, jotta pystyn keskittymään haastattelutilanteeseen täysin.

5 Tutkimuksen tulokset

5.1 Yleistä

Kaikilla haastateltavilla on pitkä ura takanaan ja merkittävästi työkokemusta vaihtelevista ohjelmistoalan työtehtävissä. Kaikilla on kokemusta sekä lyhyt- että pitkäsyklisistä kehitysmalleista, vesiputousmallista sekä iteratiivisista ketteristä menetelmistä. Tämä varsin laaja-alainen kokemus tuli hyvin selvästi ilmi vastauksissa, mistä haastattelijana olen erittäin kiitollinen.

Haastattelukysymyksiin saamani vastaukset perusteltiin usein aiemmin koettujen ongelmien kautta, eli vastausten näkökulma oli usein jonkin ongelmaksi koetun ratkaiseminen. Näitä ongelmia oli koettu sekä muilla työnantajilla että Yhtiön muiden tuotteiden parissa työskenneltäessä.

5.2 Mitä nykyisellä kehitysmallilla pyritään saavuttamaan, ts. miksi juuri tätä kehitysmallia käytetään?

Kaikki haastateltavat olivat tästä aiheesta yksimielisiä ja tiivistivät vastauksensa hyvin samalla tavalla: käytetyn kehitysmallin pohjimmainen tarkoitus on toteuttaa hyvää laatua korkealla velositeetilla, julkaista uusia ohjelmistoversioita riittävän usein ja reagoida sidosryhmien ja asiakkaiden pyyntöihin ja vaatimuksiin nopeasti.

Tärkeimmäksi yksittäiseksi esiin tuoduksi tekijäksi muodostui laatu, koska se on samaan aikaan sekä tavoiteltu lopputulos että käytännössä kaikkien muiden kehitysmallilta odotettujen asioiden mahdollistaja: laadukas ohjelmisto pitää asiakkaat, sidosryhmät sekä itse kehitystiimin tyytyväisenä, ja tyytyväinen kehitystiimi tuottaa hyvää laatua, jota sidosryhmien on helppo markkinoida, myydä ja tukea sekä asiakkaiden ostaa. Hyvin toimiva kehitysmalli on kokonaisuus ja enemmän kuin osiensa summa; järjestämällä kehitysmalli hyvän laadun toimittamisen ehdoilla suurin osa muista halutuista ominaisuuksista saavutetaan – hieman yllättäenkin – tämän luonnollisena seurauksena.

Lyhyt julkaisusykli oli myös erittäin tärkeä kriteeri kehitysmallia mietittäessä. Julkaisusykli on edelleen melko lyhyt, mutta se on kuitenkin pidempi kuin mitä Scrum-ohjekirjat suosittelevat. Sykliä on myös pidennetty kahteen otteeseen kehitysmallia iteroitaessa, koska siten koettiin päästävän parempaan laatuun.

5.3 Mitkä tekijät, käytännöt ja tietojärjestelmät mahdollistavat nykyisen kehitysmallin käyttämisen ja tekevät siitä hyvän?

Tämän kysymyksen vastauksissa ilmeni jonkin verran eroja, näkökulma vaihteli lähinnä vastaajien työn sisältöön perustuen. Kaikki vastaajat olivat kuitenkin yhtä mieltä siitä, että tärkein yksittäinen nykyisen kehitysmallin käyttämisen mahdollistava tekijä on itse asiassa käytössä oleva ketterä kehitysmenetelmä, eli nimenomaisesti se tiimille valittu ja tiimin käyttöön muokattu Scrum-pohjainen hybridi. Kyseessä ei ole kehäpäätelmä, vaan tämä ajattelu itsessään on erittäin ketterää: paradigman ihanteita toteutetaan käytännössä, mutta varsinaisista ketteristä menetelmistä on sisällytetty tiimin käyttämään kehitysmalliin vain ne osat, jotka sopivat tiimin työskentelytapoihin ja tukevat niitä. Koko tiimi on sitoutunut tähän dogmittomuuteen sekä kehitysmallin jatkuvaan kehittämiseen ja parantamiseen. Kuka tahansa tiimin jäsenistä saa vapaasti ehdottaa muutoksia toimintamalleihin, joista sitten keskustellaan ja joita myös testataan käytännössä. Hyväksi koetut muutokset otetaan osaksi kehitysmallia ja huonoista luovutaan testin jälkeen.

Kaikkein tärkeimpänä yksittäisenä kehitysmallia tukevana tietojärjestelmänä pidettiin kommunikaatioalusta Slackia, jota on käytetty Yhtiössä ja usean vuoden ajan – Slackiä ei siis tarvinnut hankkia tätä hanketta varten. Slack mahdollistaa reaaliaikaisen keskustelun ja tehokkaan tiedonvälityksen, ja se on korvannut sähköpostin melkein kokonaan (ja puhelimet oikeastaan täysin). Slackin suurimpana ansiona pidettiin sen luonnetta orgaanisen keskustelun mahdollistajana ja rohkaisijana. Suurin osa tiimin välisestä kommunikoinnista käydään kokonaan Slackissa, joten etätöiden tekemistä pidettiin todella helppona: käytännössä ei ole mitään konkreettista eroa siinä, työskenteleekö toimistolla vaiko etänä. Slackin haittapuolina mainittiin keskustelun seuraamiseen kuluva aika, yksityiskohtaisten asioiden tarkka hakeminen jälkikäteen sekä se, että kommunikaation helppoudella on myös laiskistava vaikutus: tietoa ei läheskään aina viitsitä hakea itse, koska Slackissa on niin helppo ”kysellä tyhmiä” tai pyytää suoraa linkkiä. Tämä koettiin hieman oman työn sälyttämisenä muille, mutta ongelmaa ei pidetty mitenkään merkittävänä: itse asiassa suurin osa vastaajista myönsi itsekin toisinaan syyllistyvänsä tähän. Slackin hyvien puolien katsottiin ylittävän kaikki huonot puolet moninkertaisesti, eikä kukaan haastatelluista pystynyt edes kuvittelemaan tilannetta, jossa kommunikoitaisiin ”vanhanaikaisesti” eli pääasiassa sähköpostitse.

Jokapäiväisen työnkulun keskiössä on uusi tikettijärjestelmä Jira, jota käytetään kehitysprosessin kaikkien vaiheiden hallintaan: Jirassa on koko tuotteen kehitysjono, pyrähdyksen tehtävälista sekä työnkulun tehokkaasti visualisoiva Kanban-taulu. Tiimin käyttämän

Jira-instanssin työnkulkuihin on tehty merkittävästi muutoksia ja optimointeja tiimin työskentelytapojen mukaisesti.

Jiraa tukemaan ja täydentämään hankittiin myös organisaatiowikiohjelmisto Confluence. Käytännössä tämä toimii tiimitason intranetinä, jonne talletetaan kaikki tärkeät päätökset, palaverimuistiot, retrospektiivit, suunnitelmat jne. Kaikki haastatellut pitivät myös Confluencen hankkimista erittäin hyvänä uudistuksena, vaikka se onkin pääasiassa hankepäällikön ja liiketoimintajohtajan työkalu, eivätkä kehittäjät tarvitse sitä omassa työssään kovinkaan usein.

Molempien tietojärjestelmien optimoinnit on tehty siten, että niiden käyttäminen vaatii kehittäjiltä mahdollisimman vähän aikaa – tämä ei ole läheskään aina itsestäänselvyys, koska huonosti konfiguroitu Jira saattaa työn helpottamisen sijasta haitata työntekoa esim. epäloogisilla ja liian monimutkaisilla työnkuluilla. Meidän tapauksessamme kaikki työnkulut on kuitenkin sekä mietitty että toteutettu järkevästi ja loogisesti, ja kaikki haastatellut kehittäjät pitivät tiimin työtikettien valintaa, käsittelyä ja seurantaa hyvin helppona. Kaikki haastatellut kehittäjät ja liiketoimintajohtaja antoivat tästä ison tunnustuksen järjestelmän valinneelle ja pystyttäneelle hankepäällikölle, jonka ammattitaitoa ja merkittävää kokemusta vastaavista hankkeista pidettiin suurimpana yksittäisenä syynä tietojärjestelmäpuolen onnistumisiin.

Etätöiden helppoudesta huolimatta niitä kuitenkin tehdään verrattain vähän, keskimäärin noin yksi päivä viikossa per tiimin jäsen. Kaikki tärkeät päätökset tehdään yhdessä joko vartin päivittäispalaverissa tai erikseen aikataulutetuissa kokouksissa, joihin ei yleensä järjestetä etäosallistumismahdollisuutta, koska siihen ei ole tarvetta. Ensinnäkin tiimi arvostaa kasvokkain käytäviä keskusteluja ja toisaalta myös viihtyy erittäin hyvin yhdessä, mikä johtuu suurimmaksi osaksi tiimin kesken viljeltävästä, ajoittain myös varsin mustasta huumorista.

Kaiken tämän lopputuloksena on sitoutunut, motivoitunut ja suorastaan onnellinen tiimi, mikä näkyy erittäin konkreettisesti jokapäiväisessä työssä.

5.4 Mitä hyvää nykyisessä kehitysmallissa on ja vastaako se sille asetettuihin odotuksiin?

Haastatellut olivat hyvin yksimielisiä siitä, että kehitysmalli on kokonaisuutena erittäin hyvä, eikä siitä puutu mitään tarpeellista. Vastauksissa tuli myös selkeästi ilmi, että kehitys-

mallista ei puutu mitään siitä nimenomaisesta syystä, että siihen on aktiivisesti ja tiimityönä tuotu kaikki tarvittava – prosessin jatkuva iteroiminen ja parantaminen sai haastatelluilta runsaasti kiitosta. Yhden haastatellun arkkitehdin sanoin: ”Tämä on ehdottomasti parhaiten toimiva tekniikkatiimi, jossa koskaan olen työskennellyt”.

Haastatellut olivat myös täysin yksimielisiä siitä, että nykyinen kehitysmalli vastaa erittäin hyvin sille asetettuihin odotuksiin. Tähänkin sisältyi tärkeä huomio siitä, että mikäli kehitysmallissa havaitaan puutteita tai parannuskohteita, asioita voidaan aina kehittää ja muuttaa, eli muutokseen on ketterän paradigman mukaisesti todellakin sitouduttu.

Kehitysmallin toimivuudesta kaikki haastatellut kehittäjät ja liiketoimintajohtaja kiittelivät hankepäällikköä, joka on ollut tärkein yksittäinen henkilö kehitysmallia rakennettaessa. Hankepäällikkö puolestaan kiitteli motivoitunutta ja sitoutunutta tiimiä työnsä helpottamisesta ja kehitysmallin onnistumisesta. Kaikki olivat yhtä mieltä siitä, että merkittävä syy hankkeen onnistumiseen on myös erittäin kokenut ja ammattimaisesti toimiva tiimi.

5.5 Mitä parannettavaa nykyisessä kehitysmallissa on?

Itse kehitysmallissa ei koettu olevan parannettavaa, kaikki vastaajat olivat siihen hyvin tyytyväisiä. Vastaajat mainitsivat myös, että mikäli parannettavaa olisi, parannukset eittämättä tehtäisiin (kuten tähänkin asti on tehty). Parannusehdotuksia tuli kyllä useita, mutta ne menivät kehitysmallin ulkopuolelle ja koskivat enemmänkin vakiintuneita toimintatapoja ja Yhtiön nykyistä tilaa ylipäänsä.

Eniten mainintoja kehittäjiltä tuli testauksesta: automaattisten testien kattavuutta pitäisi kasvattaa koko ajan, painotusta pitäisi siirtää manuaalitestauksesta automatisoituun testaukseen, testisyklejä pitäisi pidentää ja suorituskertoja lisätä. Siirtyminen SaaS-malliin ja pilvipalveluihin nostaa testauksen tärkeyttä entisestäänkin.

Dokumentoinnista samaten tuli useampia mainintoja kehittäjiltä ja sen isoimmaksi ongelmaksi koettiin toteutus monen tuotetiimin kesken jaetuilla henkilöresursseilla, minkä vuoksi kehittäjien aikaa kuluu perustason dokumentoinnin kirjoittamiseen ja tikettien ”syöttämiseen” dokumentoijille. Dokumentoinnin työkuormaa pidettiin myös liian suurena nykyiselle dokumentointitiimille.

Liiketoimintajohtaja ja hankepäällikkö identifioivat myös henkilöriskeihin perustuvia ongelmia, sillä osa tuotteen toiminnan kannalta keskeisistä ja/tai suurta osaamista vaativista komponenteista tai tekniikoista on pahimmillaan vain yhden kehittäjän varassa.

Huomionarvoista on, että kaikkia kolmea yllä olevaa parannusehdotusta pidettiin pääosin melko suoraviivaisina resurssiongelmina, jotka voitaisiin suhteellisen helposti korjata lisäinvestoinneilla ja rekrytoinneilla.

Kaikki toivoivat parempaa tiedon jakamista Yhtiön eri yksiköiden ja osastojen välillä, siiloutumista pidettiin todellisena ja merkittävänä ongelmana. Vastaajien mielestä kuitenkin tiimin sisällä tieto kulkee ongelmitta.

Osa kehittäjistä mainitsi myös, että käytettävissä ei ole riittävää tietoa markkinoista ja asiakkaista, joten monet tuotteen kehityspäätöksistä tehdään puutteellisen tiedon valossa. Väärin arvaaminen on kallista ja rajatuilla resursseilla ei yksinkertaisesti voi tehdä kaikkea.

5.6 Vertaa nykyistä kehitysmallia muihin tuntemiisi kehitysmalleihin

Moni haastateltavista kehittäjistä on työskennellyt Yhtiön muissa tuotetiimeissä, joiden tekemät tuotteet ovat perinteisempiä monoliitteja, joista julkaistiin vuoden aikana 1-2 ohjelmistoversiota. Pidemmän julkaisusyklin katsottiin jo myötäsyntyisesti aiheuttavan merkittävästi ongelmia: pidemmän aikavälin suunnitelmiin on paljon vaikeampi sitoutua, kehitysjonon sisältö muuttuu usein myynniltä ja asiakkailta tulevien toivomusten mukaan ja asiakkaat joutuvat odottamaan bugikorjauksia tarpeettoman kauan, tai sitten bugikorjaukset pitää toimittaa asiakkaille erillisinä minor-julkaisuina, joiden toimittaminen vie paljon resursseja varsinaisesta kehitystyöstä sekä sotkee huolella tehdyt suunnitelmat ja aikataulutukset.

Tarpeeksi lyhyt julkaisusykli tavallaan korjaa kaikki nämä ongelmat luontaisesti: lyhyempi kehitysjono on helpompi suunnitella, sen toteuttamiseen on merkittävästi helpompi sitoutua, uudet ja toivotut ominaisuudet voidaan toteuttaa suunnitelmallisesti ja asiakkaat saavat sekä uudet ominaisuudet että bugikorjauksensa huomattavasti lyhyemmässä ajassa. Koska lyhyempi julkaisusykli on helpompi toimittaa annetussa aikataulussa, kehitystiimi kokee työn mielekkääksi ja onnistuminen motivoi tiimiä, mikä on myös merkittävä etu, koska tyytyväiset kehittäjät tuottavat parempaa laatua nopeammin. Usein tapahtuvan julkaisun seurauksena tiimi saa myös nopeammin palautetta ohjelmistoon jääneistä bugeista sekä toimitetuista ominaisuuksista, mikä mahdollistaa nopean reagoimisen ongelmiin ja

kehitysehdotuksiin, mikä puolestaan parantaa ohjelmiston laatua ja asiakaskokemusta entisestään. Kyseessä onkin itseään ruokkiva positiivinen kierre.

Tämän kysymyksen yhteydessä kaikki haastatellut kehittäjät toivat esiin, että Tuotteen arkkitehtuuri vaikuttaa asiaan merkittävästi. Koska Tuote on Yhtiön uusin ohjelmistohanke ja sen arkkitehtuuri noudattaa modernia microservice-mallia, uusien ominaisuuksien kehittäminen on huomattavasti helpompaa ja nopeampaa kuin monoliittisella arkkitehtuurilla toteutetulla tuotteella, jossa uusien ominaisuuksien ja tuoteversioiden kehittäminen ja julkaiseminen aiheuttaa lähes aina merkittäviä regressioriskejä. Yleinen näkemys oli myöskin, että monimutkaista monoliittista tuotetta ei voitaisi kehittää nykyisellä kehitysmallilla, koska uusia ohjelmistoversioita ei pystyttäisi julkaisemaan niin usein kuin nyt tehdään.

Monella vastaajalla oli myös kokemusta ohjelmistokehityksestä vesiputousmallilla, ja yhtä poikkeusta lukuun ottamatta kaikki kommentit olivat hyvin negatiivisia. Positiivinen kommentti tuli yhdeltä arkkitehdeistä, joka oli urallaan kehittänyt pelimoottoria vesiputousmallilla ja sellaiseen käyttöön se sopi kuulemma hyvin, moderniin ominaisuusvetoiseen ohjelmistotuotantoon sen sijaan ei hänenkään mielestään.

5.7 Millaisena näet tulevaisuuden Tuotteen ja sen kehittämisen osalta?

Kaikki haastateltavat olivat yhtä mieltä siitä, että kehitysmallin suhteen ei ole mitään tarvetta tehdä muutoksia ainakaan lähitulevaisuudessa. ”Ehjää ei tarvitse korjata”, sanoi yksi kehittäjistä.

Muutoksia kehitysmalliin pidettiin hyvin todennäköisinä siinä tapauksessa, että tiimin koko kasvaa merkittävästi, koska nykyisen mallin ei uskottu skaalautuvan siihen ja esim. päivittäispalavereista tulisi todennäköisesti liian pitkiä.

Muutoksia kehitysmalliin pidettiin väistämättöminä, mikäli tiimi jakautuu kahdeksi tai useammaksi tiimiksi tai jos tulevaisuudessa siirrytään täysipainoiseen jatkuvaan julkaisuun, jossa tuotantoon vienti tehdään esim. useita kertoja päivässä. Molemmissa tapauksissa toimintaympäristön muutoksia pidettiin niin suurina, että koko kehitysmalli pitäisi miettiä alusta lähtien kokonaan uusiksi, vaikka toki nykyiset hyvät käytännöt pyrittäisiin siirtämään uusittuunkin malliin niiltä osin kuin ne siihen sopisivat.

5.8 Millaisena näet tulevaisuuden ohjelmistokehityksen osalta yleisesti?

Ketterät kehitysmenetelmät ja lyhytsyklinen iteratiivinen prosessi nähtiin alan perustrendeinä, jotka ovat tulleet jäädäkseen. Tulevaisuudessa prosessoriaika on hyvin todennäköisesti yhtä ilmeinen ja helposti saatavilla oleva hyödyke kuin esim. sähkö, eli kuten yksi haastatelluista kehittäjistä asian kuvaili: ”prosessoriaikaa saa suoraan seinästä”.

Toinen vallitseva trendi on ohjelmistojen siirtyminen SaaS-malliin ja kokonaan pilvipalveluihin, minkä seurauksena ohjelmistojen saatavuus ja yleinen laatu paranee ja hinta laskee. Yksi arkkitehdeista sanoin näin: ”Kukaan ei enää ajattele palvelimia tai rautaa yleensä, vaan koodi ajetaan lambdafunktioina pilvessä ja kapasiteetin kasvattaminen hoituu aina tarpeen mukaan automaattisesti”.

SaaS-malliin ja pilveen siirtyminen tuottaa ohjelmistokehitykselle useita haasteita ja vaikeuttaa monia asioita merkittävästi: jatkuva julkaisu eli koodin siirtäminen suoraan pilveen ja tuotantoon vaatii kattavaa ja täysin automatisoitua testausta sekä viimeisilleen viilattua julkaisuputkea, koska uuden version julkaiseminen ei ole enää perinteinen pitkä lista manuaalisia tehtäviä.

6 Pohdinta

Tämän opinnäytetyön tavoitteena on ollut antaa lukijalle kuva ketterästä ohjelmistokehityksestä sekä lisätä ymmärrystä siitä, kuinka ohjelmistokehitystä tekevä yritys voi järjestää tiiminsä käyttämään ketteriä menetelmiä mahdollisimman hyvin. Omasta mielestäni tavoitteet on saavutettu hyvin: oleellinen teoria on käsitelty riittävän yksityiskohtaisesti, kehitysmallin parantamishankkeen syyt ja läpivienti Yhtiössä on kuvattu kattavasti sekä haastatteluvastaukset on analysoitu seikkaperäisesti.

On aivan kiistatonta, että kehitysmallin päivittämisessä on onnistuttu joka suhteessa erinomaisesti. Tuotteen parissa työskentelevän tiimin tyytyväisyys on huippuluokkaa ja tiimi tuottaa erittäin hyvää laatua nopeasti ja tehokkaasti. Tyytyväisyys näkyy erittäin konkreettisesti hyvänä ja vahvana me-henkenä tiimin kesken, runsaana huumorina ja aivan erityisesti siinä, että kaikki tiimin työskentelyyn liittyvä toimii erinomaisesti: jäsenien tekemiin lupauksiin voi luottaa, mitään ei tarvitse erityisesti valvoa eikä sovittujen asioiden perään kysellä ja vaikeidenkin asioiden toteuttajiksi ilmoittautuu useita vapaaehtoisia, kun näitä pitää etsiä. Pyrähdysten suunnittelu hoituu nopeasti ja vaivattomasti, koska arkkitehdit käyttävät merkittävästi aikaa kehitysjonon jalostamiseen ja vastaavasti itseohjautuva ja sitoutunut tiimi valitsee jonosta toteutettavat asiat ja toteuttaa ne yleensä aikataulussa. Työskentely tiimissä on helppoa ja mukavaa, eikä mitään ongelmia oikeastaan ole.

Tiimin nykyisellään käyttämään kehitysmenetelmään ei ole päädytty sattumalta, vaan kyseessä on aktiivisella työllä tehty muutos, joka on toteutettu tarkkaan suunnitellulla ja pitkäjänteisellä parantamishankkeella iteratiivisesti, mikä jo itsessään on ketterää. Scrumia ei valittu pohjaksi vain sen takia, että se on suosituin ketterä viitekehys, vaan siksi, että sen katsottiin parhaiten sopivan tämän tiimin työtapoihin. Scrumin rooleista, tuotoksista ja tapahtumista otettiin merkittävä osa sellaisenaan käyttöön, mutta niistä myös poikettiin silloin, kun se tiimin työtapoja noudattaen oli perusteltua: esim. pyrähdyksen kestoa pidennettiin kaksi eri kertaa, minkä seurauksena pyrähdyksen kesto meni jo selvästi Scrum-oppaiden suositusten yli.

Kehitysmallin päivittämisessä tärkeimmäksi tekijäksi muodostui laatu, koska se on sekä tavoiteltu lopputulos että monien muiden asioiden mahdollistaja. Järjestämällä kehitysmalli hyvän laadun toimittamisen ehdoilla suurin osa muista halutuista ominaisuuksista saavutettiin laatuun tähtäämisen luonnollisena seurauksena. Hyvä laatu myös vähentää tarvetta käyttää aikaa teknisen velan maksamiseen ja virheiden korjaamiseen, joten säästyneitä

resursseja voidaan käyttää tuotteen kehittämiseen. Laadun toimittaminen on näin ollen positiivinen itseään ruokkiva kierre.

Kehitystiimin tärkeimmät ohjenuorat ovat järjen käyttö ja kommunikointi: keskustellaan avoimesti siitä, pitääkö jotain tehdä, ja sitten tehdään kuten on keskusteluissa sovittu. Kuka tahansa saa koska tahansa ehdottaa parannuksia, ja tiimin jäsenet ovat hyvin tottuneita ilmoittamaan epäkohdista ja esittämään parannusehdotuksia aina moisia havaittuaan. Avoin kommunikointi ja muiden kunnioittaminen on jatkuvan kehittämisen elinehto.

Ketteriin menetelmiin sinänsä ei suhtauduta ideologisesti eikä niiden dogmeihin uskonnollisesti, vaan valitaan vapaasti kaikista menetelmistä juuri ne työkalut, jotka sopivat tämän nimenomaisen kehitystiimin työtapoihin. Prosessi ei saa koskaan olla itsetarkoituksellinen, vaan sen on palveltava tehtävää työtä. Tiimillä ei ole eksplisiittisiä arvoja, mutta implisiittiset arvot ovat mielestäni hyvin selkeät: tiimin toiminnassa ja ihanteissa näkyvät ketterän ohjelmistokehityksen julistuksen arvon ja periaatteet aivan konkreettisesti – niin selvästi, että eksplisiittisiä arvoja ei mielestäni välttämättä edes tarvita.

Koska opinnäytetyöni dokumentoi koko parantamishankkeen syyt ja seuraukset melko seikkaperäisesti, työtä voi sellaisenaan hyödyntää vastaavien hankkeiden suunnitteluun ja toteuttamiseen. Myös haastattelukysymyksiä voi käyttää muiden tuotetiimien haastatteluun, etenkin jos parannushankkeen onnistumista halutaan arvioida kriittisesti. Omasta mielestäni erittäin mielenkiintoinen jatkotutkimusidea olisi tutkia Scrumista Kanbaniin tai Scrumbaniin siirtyvää tiimiä, erityisesti jos tuote on kypsässä vaiheessa ja sen toteutustiimi on havainnut Scrumin käyttämisessä haasteita tai suoranaisia ongelmia.

Olen käytännössä koko 20-vuotisen ICT-urani ajan työskennellyt pienissä tiimeissä ketterästi ja ollut kohtalaisen tietoinen ketterän ohjelmistokehityksen teorioista ja periaatteista, vaikka monissa työpaikoissa ketterien menetelmien toimeenpano onkin jättänyt varsin paljon toivomisen varaa. Hieman häveten joudun kuitenkin myöntämään, että vasta tämän opinnäytetyön myötä perehdyin kunnolla näihin teorioihin ja periaatteisiin. Ensinnäkin havaitsin, että niissä on paljon järkeä, mutta todella pysäyttävää oli ymmärtää, kuinka hämmästyttävän hyvin ketterän ohjelmistokehityksen julistuksen periaatteet toteutuvatkaan toimivassa sitoutuneessa tiimissämme, ja vielä aivan kuin itsestään: ne manifestoituvat tiimin jokapäiväisen toiminnan kautta täysin luonnollisesti ja pakottamatta. Toinen itselleni erittäin merkittävä asia liittyy Scrumin suosittelemiin retrospektiiveihin: kun palasin nykyiseen tiimiini scrummasteriksi, huomasin että tiimi ei käytä formaaleja retrospektiivejä jokaisen pyrähdyksen päätteeksi. Minun aloitteestani otimme retrospektiivit jokaisen pyräh-

dyksen normaaliin ohjelmaan, ja tähän mennessä pitämissämme retrospektiiveissä onkin ehdotettu useita parannuksia, jotka on sitten seuraavien pyrähdysten aikana toteutettu. Tiimi on kokenut retrospektiivit erittäin hyödyllisiksi ja minulle on ollut äärimmäisen arvokasta, että yhtäältä koen tuottaneeni tiimille todellista arvoa ja toisaalta olen kouriintuntuvasti havainnoinut Scrumin suositteleman työkalun konkreettisen arvon.

Opinnäytetyön kirjoittaminen oli minulle itselleni valtava ponnistus ja siihen kului huomattavasti enemmän aikaa kuin mitä urakan alkupuolella kuvittelin. Yhtäältä tämä paransi stressinsietokykyäni ja toisaalta pakotti minut tarkastelemaan omia työskentelytapojani, joissa olenkin identifioinut selkeitä parannuskohteita. Opinnäytetyön kirjoittamiseksi olen joutunut havainnoimaan omaa työympäristöäni ja pohtimaan syvällisesti suhteitani työtovereihini, ja tajusinkin, että olen erittäin etuoikeutettu ja onnellisessa asemassa saadessani työskennellä näin pätevien, motivoituneiden ja mukavien ihmisten kanssa. Tämä on itselleni se kaikkein tärkein opinnäytetyöstä oppimani yksittäinen asia.

Lähteet

AgencyAgile 2019. Will Management Succeed at Killing Agile? Luettavissa: https://agencyagile.com/will-management-succeed-at-killing-agile/. Luettu: 9.10.2019.

Agile Alliance 2019a. Agile 101. Luettavissa: https://www.agilealliance.org/agile101/. Luettu: 4.10.2019.

Agile Alliance 2019b. Agile Glossary. Luettavissa: https://www.agilealliance.org/agile101/agile-glossary/. Luettu: 18.9.2019.

Agile in a Nutshell 2019. Agile Myths. Luettavissa: http://www.agilenutshell.com/agile_myths. Luettu: 7.10.2019.

Agilemanifesto.org 2001a. History: The Agile Manifesto. Luettavissa: https://agilemanifesto.org/history.html. Luettu: 4.10.2019.

Agilemanifesto.org 2001b. Ketterän ohjelmistokehityksen julistus. Luettavissa: https://agilemanifesto.org/iso/fi/manifesto.html. Luettu: 4.10.2019.

Agilemanifesto.org 2001c. Julistuksen takana olevat periaatteet. Luettavissa: https://agilemanifesto.org/iso/fi/principles.html. Luettu: 4.10.2019.

Ambler, S. 2019. Examining the Agile Manifesto. Luettavissa: http://www.ambysoft.com/essays/agileManifesto.html. Luettu: 7.10.2019.

Atlassian 2019a. What is Scrum? Luettavissa: https://www.atlassian.com/agile/scrum. Luettu: 8.11.2019.

Atlassian 2019b. What is Kanban? Luettavissa: https://www.atlassian.com/agile/kanban. Luettu: 12.11.2019.

CollabNet VersionOne 2019. 13th Annual State of Agile Report. Luettavissa: https://www.stateofagile.com/#ufh-c-473508-state-of-agile-report. Luettu: 8.10.2019.

Collin, N. 2019. Microservice-arkkitehtuuri. Luettavissa: https://www.collin.fi/~niklas/solita/training/microservices/#/2/1. Luettu: 11.12.2019.

Concepta 2019. The Importance of Scalability in Software Design. Luettavissa: https://conceptainc.com/blog/importance-of-scalability-in-software-design/. Luettu: 25.11.2019.

Denning, S. 2018. Why Agile Is Eating the World. Luettavissa: https://www.forbes.com/sites/stevedenning/2018/01/02/why-agile-is-eating-the-world%E2%80%8B%E2%80%8B/. Luettu: 8.10.2019.

Denning, S. 2019. Why Agile's Future Is Bright. Luettavissa: https://www.forbes.com/sites/stevedenning/2019/08/25/why-the-future-of-agile-is-bright/#50b7fa852968. Luettu: 24.11.2019.

Digité 2019. What is Kanban? Luettavissa: https://www.digite.com/kanban/what-is-kanban/. Luettu: 12.11.2019.

Drumond, C. 2019. Is the Agile Manifesto still a thing? Luettavissa: https://www.atlassian.com/agile/manifesto. Luettu: 24.11.2019.

Edwards, J. 2017. 7 simple ways to fail at agile. Luettavissa: https://www.cio.com/article/3234366/7-simple-ways-to-fail-at-agile.html. Luettu: 12.12.2019.

Eskelinen A., Kemppi, T., Laanti, M., Lekman, L., Lindström, J., Luoto, K., Toivola, T., Virtanen, P., & von Weissenberg, M. 2018. Suomenkielinen scrum-sanasto. Luettavissa: https://scrumwell.files.wordpress.com/2018/03/suomenkielinen-scrum-sanasto-2018-v1-0.pdf. Luettu: 18.9.2019.

Fybish, A. 2019. The Rise of Zero-Trust Architecture. Luettavissa: https://hackernoon.com/the-rise-of-zero-trust-architecture-17464e6cbf30. Luettu: 13.11.2019.

Huotarinen, J. 2016. Tiesitkö jatkuvan julkaisun olevan jo arkipäivää? Luettavissa: https://gofore.com/tiesitko-jatkuvan-julkaisun-olevan-jo-arkipaivaa/. Luettu: 25.11.2019.

Kanbanize 2019. Kanban Explained for Beginners. Luettavissa: https://kanbanize.com/kanban-resources/getting-started/what-is-kanban/. Luettu: 7.10.2019.

Karaivanov, D. 2019. Are We Seeing the End of Scrum? Luettavissa: https://kanbanize.com/blog/rise-kanban-scrum-world/. Luettu: 9.12.2019.

Kharenko, A. 2015. Monolithic vs. Microservices Architecture. Luettavissa: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59. Luettu 20.11.2019.

Kneafsey, S. 2019. A Short History of Scrum. Luettavissa: https://www.thescrummaster.co.uk/scrum/short-history-scrum/. Luettu: 8.11.2019.

Kniberg, H. 2009. Kanban vs Scrum – How to Make the Most of Both. Luettavissa: https://fenix.tecnico.ulisboa.pt/downloadFile/3779576751814/Kanban-vs-Scrum.pdf. Luettu: 12.11.2019.

KnowledgeHut 2019. The Ultimate Guide to the Agile Manifesto. Luettavissa: https://www.knowledgehut.com/blog/agile/agile-manifesto-principles-values. Luettu: 6.10.2019.

Marjusaari, V. 2017. Agile ja ketterät menetelmät – pelkkää sanahelinääkö? Luettavissa: https://www.solita.fi/blogit/agile-ja-ketterat-menetelmat-pelkkaa-sanahelinaako/. Luettu: 8.10.2019.

Mehandru, K. 2019. Slack and Zoom Have Proven That the Future of Work Is Agile. Luettavissa: https://www.forbes.com/sites/karanmehandru/2019/10/22/slack-and-zoom-prove-the-future-of-work-is-agile/#465727b85c46. Luettu: 14.11.2019.

Mindtools 2019. The Build-Measure-Learn Feedback Loop -kuvituskuva. Luettavissa: https://www.mindtools.com/pages/article/build-measure-learn.htm. Luettu: 10.10.2019.

Morris, B. 2018. Relax. There’s no conflict between architecture and agile. Luettavissa: https://www.ben-morris.com/relax-theres-no-conflict-between-architecture-and-agile/. Luettu: 22.11.2019.

Pendolin, H. 2018. Mikä on MVP – ja mitä se ei ole. Luettavissa: https://tuotejohtaminen.fi/mita-tarkoittaa-mvp-ja-mita-ei/. Luettu: 20.11.2019

Project-management.com 2019. Ketterä kehityssykli -kuvituskuva. Luettavissa: https://project-management.com/10-key-principles-of-agile-software-development/. Luettu: 5.10.2019.

Quickscrum 2016. Breaking Down the Agile Manifesto and Understanding It. Luettavissa: https://www.quickscrum.com/Article/ArticleDetails/4075/3/Breaking-Down-The-Agile-Manifesto-And-Understanding-It. Luettu: 6.10.2019.

Rehkopf, M. 2019. Kanban vs. Scrum. Luettavissa: https://www.atlassian.com/agile/kanban/kanban-vs-scrum. Luettu: 9.12.2019.

Ries, E. 2009. Minimum Viable Product: A Guide. Luettavissa: http://www.startuplessonslearned.com/2009/08/minimum-viable-product-guide.html. Luettu: 10.10.2019.

Ries, E. 2011. The Lean Startup. Crown Publishing Group. New York.

Saaranen-Kauppinen, A. & Puusniekka, A. 2006. 6.3.3 Strukturoitu ja puolistrukturoitu haastattelu. Luettavissa: https://www.fsd.uta.fi/menetelmaopetus/kvali/L6_3_3.html. Luettu: 27.11.2019.

Schwaber, K. & Sutherland, J. 2017. Scrum-opas – Scrumin määritelmä ja pelisäännöt. Luettavissa: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Finnish.pdf. Luettu: 2.10.2019.

Scrum Alliance 2015. The 2015 State of Scrum Report. Luettavissa: https://www.scrumalliance.org/ScrumRedesignDEVSite/media/scrumalliancemedia/files%20and%20pdfs/state%20of%20scrum/scrum-alliance-state-of-scrum-2015.pdf. Luettu: 9.12.2019.

Scrum Alliance 2018. State of Scrum 2017-2018. Luettavissa: http://info.scrumalliance.org/rs/510-STH-507/images/2017-SoSR-Final%20Version_sm.pdf. Luettu: 9.12.2019.

ScrumDesk. 2019. Sprint Preplanning. Luettavissa: https://www.scrumdesk.com/start/manual-for-scrumdesk-start/sprint-preplanning/. Luettu: 13.12.2019.

Scrum.org 2019a. Scrum Glossary. Luettavissa: https://www.scrum.org/resources/scrum-glossary. Luettu: 18.9.2019.

Scrum.org 2019b. Scrum Values Poster -kuvituskuva. Luettavissa: https://www.scrum.org/resources/scrum-values-poster. Luettu: 2.10.2019.

Scrum.org 2019c. The Scrum Framework Poster -kuvituskuva. Luettavissa: https://www.scrum.org/resources/scrum-framework-poster. Luettu: 2.10.2019.

Software Testing Help 2019. Agile Manifesto: Understanding Agile Values and Principles. Luettavissa: https://www.softwaretestinghelp.com/agile-manifesto/. Luettu: 7.10.2019.

SSH Communications Security Oyj. 2019a. What Are Ephemeral Certificates? Luettavissa: https://www.ssh.com/iam/ephemeral_access/. Luettu: 13.11.2019.

SSH Communications Security Oyj. 2019b. How to Kill a Fortune 500? Luettavissa: https://www.ssh.com/iam/how-to-kill-a-fortune-500. Luettu: 13.11.2019.

Liite 1. Haastattelukysymykset

1. Mitä nykyisellä kehitysmallilla pyritään saavuttamaan, ts. miksi juuri tätä kehitysmallia käytetään?
2. Mitkä menetelmät, käytännöt ja tietojärjestelmät mahdollistavat nykyisen kehitysmallin käyttämisen ja tekevät siitä hyvän?
3. Mitä hyvää nykyisessä kehitysmallissa on ja vastaako se sille asetettuihin odotuksiin?
4. Mitä parannettavaa nykyisessä kehitysmallissa on?
5. Vertaa nykyistä kehitysmallia muihin tuntemiisi kehitysmalleihin
6. Millaisena näet tulevaisuuden Tuotteen ja sen kehittämisen osalta?
7. Millaisena näet tulevaisuuden ohjelmistokehityksen osalta yleisesti?

Liite 2. Haastattelut

1. Hankepäällikkö; päävastuu muutoksen suunnittelusta, toteutuksesta, seurannasta ja jatkokehittämisestä sekä alkuvaiheessa myös scrummasterin tehtävät (haastateltu 2.11.2018).
2. Liiketoimintajohtaja; muutoshankkeen ideoija ja primus motor, joka palkkasi hankepäällikön ja antoi hänelle hankkeen tavoitteet ja reunaehdot (haastateltu 12.11.2018).
3. Seniorikehittäjä K; alkuvaiheessa ohjelmiston kehitys, sitten scrummasterin tehtävät ja lopulta koko kehitystiimin vetovastuu (haastateltu 29.3.2019).
4. Seniorikehittäjä J; ohjelmiston kehitys, erityisalueena erittäin vaikeat ja suurta osaamista vaativat kokonaisuudet (haastateltu 29.3.2019).
5. Ohjelmistoarkkitehti M; kehitystiimin johtohahmo, rajapinta tuotehallintoon ja asiakkaisiin (haastateltu 1.4.2019).
6. Ohjelmistoarkkitehti J; tuotekehitys, uusien teknologioiden tutkiminen, tulevaisuuden visiointi, asiakasrajapinnassa toimiminen (haastateltu 1.4.2019).
7. Ohjelmistoarkkitehti P; päävastuu käyttöliittymistä (haastateltu 2.4.2019).

Liite 3. Käsitteitä

Inkrementti, tuoteversio, engl. Increment – Summa kaikista tuotteen kehitysjonon kohdista, jotka ovat valmistuneet kaikkien aiempien pyrähdysten aikana.

Iteratiivinen – Toimintapa, jossa tehdään aikarajatuissa pienissä palasissa, toimitetaan usein ja sykliä toistetaan.

Jatkuva integraatio, engl. Continuous Integration, CI) – Ohjelmistotuotannon menetelmä, jossa useat kehittäjät työskentelevät samanaikaisesti ohjelmistoprojektissa lisäämällä omat lähdekoodimuutoksensa yhteiseen varantoon, jossa ne validoidaan, yhdistetään, käännetään ja testataan täysin tai osittain automaattisesti.

Jatkuva julkaisu, engl. Continuous Delivery, CD) – Ohjelmistotuotannon menetelmä, jossa useat kehittäjät työskentelevät samanaikaisesti ohjelmistoprojektissa lisäämällä omat lähdekoodimuutoksensa yhteiseen varantoon, jossa ne validoidaan, yhdistetään, käännetään ja testataan automaattisesti, ja josta ne julkaistaan tuotantoon automaattisesti.

Julkaisuputki, engl. Build Pipeline) – Kokoelma tietojärjestelmiä ja/tai prosesseja, joilla varantoon lisätty lähdekoodi voidaan validoida, yhdistää, kääntää, testata ja julkaista osittain tai täysin automaattisesti.

Kehitysjonon kohta, engl. Backlog Item – Yksittäinen tuotteeseen kehitettävä ominaisuus, parannus tai korjaus.

Kehitystiimi, engl. Development Team – Inkrementin toteutuksesta vastaava monitaitoinen ja itseohjautuva joukko ammattilaisia.

Microservice-arkkitehtuuri – Yksittäinen rajatun vastuualueen käsittävä sovellus, joka muodostaa oman itsenäisen kokonaisuutensa ja tarjoaa integraatiorajapinnan muille sovelluksille.

Monoliittinen arkkitehtuuri – Yksi iso sovellus, joka sisältää ja tekee kaiken koko järjestelmässä.

Pienin toimiva tuote, engl. Minimum Viable Product (MVP) – Versio tuotteesta, joka mahdollistaa maksimaalisen validoidun oppimisen minimaalisella vaivannäöllä.

Pyrähdyksen katselmointi, engl. Sprint Review – Pyrähdyksen lopussa pidettävä tilaisuus, jossa käydään läpi kehitetty inkrementti ja tarvittaessa muokataan tuotteen kehitysjonoa.

Pyrähdyksen retrospektiivi, engl. Sprint Retrospective – Katselmoinnin jälkeen pidettävä tilaisuus, jossa tarkastellaan kulunutta pyrähdystä ja sen sujumista liittyen ihmisiin, yhteistyöhön, prosessiin ja työkaluihin.

Pyrähdyksen suunnittelu, engl. Sprint Planning – Tilaisuus, jossa suunnitellaan pyrähdyksen aikana tehtävän työ.

Pyrähdyksen työlista, engl. Sprint Backlog – Pyrähdyksen suunnittelupalaverissa muodostettava työlista eli suunnitelma inkrementin toteuttamisesta.

Pyrähdys, engl. Sprint – Enintään kuukauden pituinen aikaraja, jonka sisällä tuotetaan julkaisukelpoinen ja toimiva inkrementti.

Päivittäispalaveri, engl. Daily Scrum – Enintään 15 minuutin pituinen palaveri, jossa tarkastellaan työn etenemistä ja tavoitteiden saavuttamista sekä tehdään yhteisiä päätöksiä.

Regressio – Ohjelman aiemmin toiminut komponentti tai osa lakkaa toimimasta osittain tai kokonaan esim. uusimman päivityksen jälkeen.

Software–as–a–Service (SaaS) – Ohjelmiston hankkiminen pilvipalveluna perinteisen ohjelmiston asentamisen sijaan, eli palveluntarjoajan ylläpitämä ohjelmisto on asiakkaan käytettävissä verkon yli selaimella tai sovelluksella, mistä asiakas maksaa tyypillisesti käyttönsä mukaan.

Scrummaster – Scrum–tiimin palveleva johtaja, joka vastaa Scrumin edistämisestä ja tukemisesta auttamalla kaikkia ymmärtämään sen käytännöt ja säännöt sekä fasilitoimalla sen tapahtumat.

Scrum-tiimi, engl. Scrum Team – Tuoteomistajasta, kehitystiimistä ja scrummasterista koostuva monitaitoinen ja itseohjautuva joukko ammattilaisia.

Sidosryhmä, engl. Stakeholder – Scrum-tiimin ulkopuolinen henkilö, jolla on intressi tuotteeseen, esim. asiakas tai yrityksen johtaja.

Tuoteomistaja, engl. Product Owner – Yksi henkilö, joka on vastuussa kehitystiimin työn tuloksena saatavan tuotteen arvon maksimoinnista sekä tuotteen kehitysjonon hallinnasta, priorisoinnista ja läpinäkyvyydestä.

Tuotteen kehitysjono, engl. Product Backlog – Tuoteomistajan priorisoima lista kaikista tuotteeseen tarvittavista vaatimuksista ja muutoksista.

Tuotteen kehitysjonon jalostaminen, engl. Product Backlog Refining – Toistuva prosessi, jossa kehitysjonon kohtiin lisätään yksityiskohtia, työmääräarvioita ja keskinäistä järjestystä.

Velositeetti, engl. Velocity – Käsitys tai numeerinen arvo sille, kuinka paljon kehitystiimi pystyy toteuttamaan yhden pyrähdyksen aikana.

Zero Trust Architecture – Jokaisen käyttäjän jokainen operaatio verifioidaan joka kerta.

(Agile Alliance 2019b; Collin 2019; Eskelinen ym. 2018; Fybish 2019; Ries 2009; Schwaber & Sutherland 2017; Scrum.org 2019a)