HAAGA-HELIA
ammattikorkeakoulu

Keinoja ketteryyden lisäämiseen yrityksen sovelluskehityksessä

Pyry Simula

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

Tekijä(t)
Pyry Simula

Koulutusohjelma
Tietojenkäsittely

Raportin/Opinnäytetyön nimi
Keinoja ketteryyden lisäämiseen yrityksen sovelluskehityksessä

Sivu- ja liitesivumäärä
63 + 2

Tämä laadullinen tutkimustyyppinen opinnäytetyö tutkii erään kansainvälisen tietojärjestelmiin erikoistuneen yrityksen osaston ketteryyttä lisääviä keinoja. Työssä tutkitaan haastattelujen ja lähdekirjallisuuden avulla ketteryyttä lisääviä keinoja ja niiden sisällyttämistä toimeksiantajan sovelluskehitykseen.

Osaston projektien määrä on ollut nopeassa kasvussa, jonka takia nykyiset menetelmät eivät riitä enää hallinnolliseen työhön tai projektien poikkeamien optimaaliseen hallitsemiseen. Osasto tarvitsee tämän takia lisäkeinoja ketteryyden lisäämiseen.

Opinnäytetyö on toimeksianto osastolta, jossa keskitytään ketterien menetelmien tuomiin hyötyihin osaston sovelluskehityksen näkökulmasta. Osasto on myös toivonut ketteryyttä lisäävien keinojen lisäksi mittaamisehdotuksia annetuista ketteryyttä lisäävistä keinosta sekä sisällyttämiseen vaikuttavien tekijöiden läpikäyntiä.

Vaikka opinnäytetyössä käsitellään ketteryyttä lisääviä keinoja, niin opinnäytetyö on rajattu koskemaan pääsääntöisesti Scrum-viitekehystä.

Opinnäytetyön teoria keskittyy vesiputousmalliseen sovelluskehitykseen ja ketterään sovelluskehitykseen. Teoriaosuudessa esitetään Scrum-viitekehyksen käytäntöjä ja haasteita. Teoriaosuuden lopussa käydään läpi perinteisiä ohjelmistokehityksen mittareita ja ketteriin menetelmiin kuuluvia mittareita.

Opinnäytetyö toteutetaan kahdella asiantuntijahaastattelulla, jonka perusteella selvitetään osaston nykyinen sovelluskehitysmalli ja sen tuomat haasteet. Haastatteluihin osallistui yhteensä kuusi eri osaston asiantuntijaa, jotka vastasivat avoimiin kysymyksiin osaston sovelluskehitykseen liittyen. Haastatteluissa selvisi, että osasto toimii vesiputousmallin mukaisesti ja sen suurimpia haasteita ovat työn vähäinen organisointi, muutoshallinta ja kommunikointi.

Opinnäytetyö aloitettiin tammikuussa 2019. Lähdekirjallisuuteen perehtyminen alkoi helmikuussa 2019 ja osaston haastattelut toteutettiin maaliskuun ja huhtikuun 2019 aikana.

Osaston ketteryyttä lisääviä keinoja ovat sprinttien ja Scrum-viitekehyksen roolien käyttöönotto. Osaston ketteryyttä lisää myös ketteryyteen sitoutunut johtotiimi ja sähköisten kommunikaatiosovellusten, kuten Slackin käyttö. Osaston ketteriä mittareita ovat itsearviointiin pohjautuva suhteellisen ketteryyden mittaristo ja iteraatio -ja julkaisumittarit.

Ketteryyttä lisäävien keinojen sisällyttämisessä osaston kannattaa hankkia Agile Coach ainakin sovelluskehityksen muutosprosessin ajaksi. Osaston pitää myös mahdollistaa sitoutuneiden tiimien käyttöönotto ja auttaa kaikkia osaston työntekijöitä saavuttamaan ketterä ajatusmaailma.

Osaston kannattaa tulevaisuudessa ottaa käyttöön edistynyt ketterä sovelluskehitysmalli.

HAAGA-HELIA
ammattikorkeakoulu

Tiivistelmä

Asiasanat
Agile, Scrum, mittaaminen, vesiputousmalli

Sisällys

1 Johdanto

Opinnäytetyö tehtiin yritykselle, jonka sovelluskehitys on voimakkaassa kasvussa. Toteutettavien projektien määrä on kasvanut viime vuosina paljon ja projektiryhmät ovat suurentuneet. Projekteista on tullut kasvun myötä vaikeammin hallittavia ja niiden seuranta vaatii valtavan määrän hallinnollista työtä. Tästä johtuen projektien poikkeamiin ei ole ehditty reagoida riittävän nopeasti ja muutoinkin päätöksenteko on hidastunut. Osaston projektitehtävien toteutuksen tehokkuus hukkuu myös virheherkkyyteen, koska projektien seuranta ja suunnittelu eivät ole riittävällä tasolla. Osasto tarvitseekin lisäkeinoja ketteryyteen, joille on yhteistä toimivan ohjelmiston ensisijaisuus, suora viestintä ja nopea muutoksiin reagointi.

Opinnäytteen tehtävänä on siis antaa ketteryyttä lisääviä keinoja yrityksen yhden osaston sovelluskehityksen toteutukseen ja hallintaan. Opinnäytetyön tarkoituksena on myös antaa mittaamisehdotuksia ketteryyttä lisääviin keinoihin, joiden avulla osasto pystyy kehittämään omia palveluitaan ja toimintatapojaan.

Opinnäytetyössä kerrotaan konkreettisia keinoja parantaa ketteryyttä ja annetaan ehdotuksia keinojen toimeenpanosta. Opinnäytetyö keskittyy ketterien (Agile) menetelmien hyödyntämiseen sovelluskehityksessä ja projektinhallinnassa, mutta metodeista ja viitekehyksistä keskitytään ainoastaan Scrumiin. Opinnäytetyössä ei siis käydä syvemmin läpi esimerkiksi XP-, Kanban- tai Lean-viitekehyksiä. Opinnäytetyön tutkimuksen työkaluna käytetään kirjallisuutta ja haastatteluja. Kaikki ketteryyttä lisäävät keinot ilmenevät haastattelussa havaittujen ongelmien ja oman empiirisen havainnointini perusteella.

Opinnäytetyö on toimeksianto, jossa toimeksiantaja on asettanut opinnäytetyön tavoitteeksi löytää konkreettisia keinoja ja toimenpide-ehdotuksia sovelluskehityksen laadun parantamiseksi. Osasto myös toivoo, että opinnäytetyössä käydään läpi mittareita, joiden avulla konkreettisia ketteryyttä lisääviä keinoja voidaan mitata. Tavoitteena on myös ottaa huomioon yrityksen osaston nykytilanne ja olemassa oleva kulttuuri.

Opinnäytetyön lopputuloksena syntyy itse opinnäytetyö, nykytilahaastattelu, ketteryysasiantuntijahaastattelu ja mittarointiehdotukset. Opinnäytetyön kautta osasto saa selkeitä ohjeita osaston työskentelytapojen parantamiseen ja strategian kehittämiseen. Opinnäytetyö lisätään osaston sovelluskehityksen tiekarttaan, jossa sitä tullaan suoraan hyödyntämään osaston strategian toteuttamiseen.

Opinnäytetyöni koostuu johdannosta, ohjelmistokehityksen sovelluskehitysmalleista, haastattelujen läpikäynnistä, ketteryyttä lisäävistä keinoista, keinojen sisällyttämisestä ja pohdinnasta. Ohjelmistokehityksen sovelluskehitysmalleissa kerron teoriaa vesiputousmallista, ketteristä menetelmistä, Scrumista ja mittaamisesta. Haastattelujen läpikäyntiosuudessa kerron tutkimusmenetelmästä lisää ja haastattelujen taustan ja tulokset. Kehitystoimenpiteissä kerron haastatteluihin ja tietoperustaan pohjautuen tarkemmin keinoista lisätä ketteryyttä ja siitä, miten niitä voisi sisällyttää osaston sovelluskehitykseen. Pohdinnassa analysoin tutkimuksen tuloksia ja mahdollista jatkokehittämistä.

Opinnäytetyön keskeiset käsitteet ovat:

Vesiputousmalli - Perinteinen sovelluskehitysmenetelmä, jossa kehitystyö etenee lineaarisessa järjestyksessä vaatimustenmäärittelystä käyttöönottoon.

Agile - Ketterä kehittäminen - Sovelluskehitysmenetelmä, joka pyrkii vastaamaan tehokkaasti muuttuvan ympäristön vaatimuksiin.

Scrum - Ketterän sovelluskehityksen malli, jonka keskiössä on empiirinen lähestyminen ja mahdollisimman suuren lisäarvon tuottaminen.

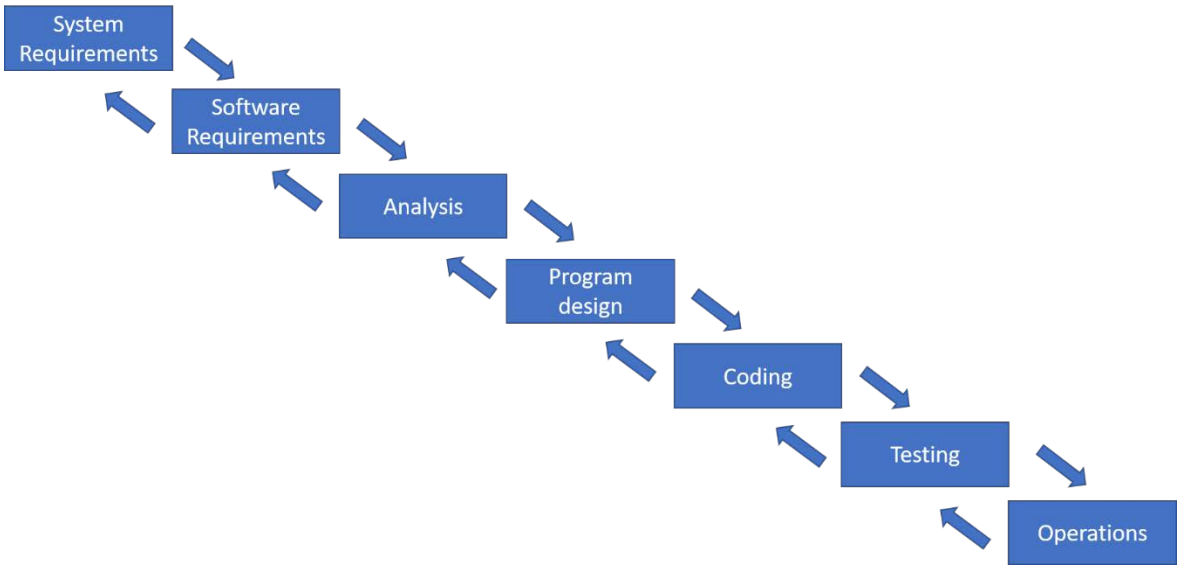
Iteraatio - Toistuva, aikarajattu tapahtuma, jossa kehitystyö tapahtuu. Iteraation pituus vaihtelee projektien mukaan, mutta sen pituus pysyy samana projektin sisällä.

Inkrementti - Iteraation aikana rakennettu versio, joka on täysin testattu ja potentiaalisesti julkaisukelpoinen. Uusi inkrementti rakennetaan aina vanhan inkrementin päälle.

2 Ohjelmistokehityksen sovelluskehitysmallit

2.1 Vesiputousmalli

Ennen ketterää ohjelmistokehitystä lineaariset suunnitteluteknikat olivat yleisessä käytössä ohjelmistokehityksessä. Näistä lineaarisista tekniikoista käytetään yleisnimitystä vesiputousmalli. Vesiputousmallin ideana on ratkaista suurten ohjelmistojärjestelmien kehittäminen niin, että aluksi määritellään järjestelmän vaatimukset, sitten järjestelmä toteutetaan ja testataan ja lopuksi järjestelmä otetaan käyttöön. Vesiputousmallia käytetään niin, että ohjelmistokehityksen jokainen vaihe tehdään ensin loppuun ja sen jälkeen vasta siirrytään seuraavaan vaiheeseen. Vesiputousmallin kehittäjä Winston W. Royce ei kuitenkaan tarkoittanut alun perin vesiputousmallin toteuttamista sellaisenaan, koska se tekee siitä riskialttiin ja helposti epäonnistuvan kriittisen suunnittelun takia. Hän suosittelikin, että vesiputousmallia muokattaisiin käytännössä iteratiivisemmaksi ja inkrementaalisemmaksi. Vesiputousmalli siis etenisi alaspäin, mutta se voisi palata uuden inkrementin valmistuttua ylempiin kohtiin ja iteraatioiden kohdalla edellisiin iteraatioihin. Kuva yksi esittää mukaillun mallin Roycen alkuperäisestä vesiputousmallista.



Kuva 1. Vesiputousmalli (mukaillen Royce 1970.)

Vesiputousmalli koostuu vaatimustenmäärittelystä, analyysistä, järjestelmän suunnittelusta, kehitystyöstä, testaamisesta ja järjestelmän luovuttamisesta ja ylläpitämisestä. Vaatimustenmäärittelyosiossa ohjelmistokehittäjät tai arkkitehdit kirjoittavat kaikki mahdolliset vaatimukset järjestelmälle dokumenttiin, joka kertoo mitä järjestelmän kuuluisi tehdä. Ohjelmistokehittäjät tulevat käyttämään dokumenttia pohjana kaikelle tulevalle kehitystyölle.

Analyysivaiheessa ohjelmistokehittäjät käyttävät vaatimustenmäärittelydokumenttia tutkiakseen ja rajatakseen sitä joko loogisen tai teoreettisen järjestelmän suunnittelua varten. Tämän jälkeen projekti etenee järjestelmän suunnitteluvaiheeseen, jossa ohjelmistokehittäjät muokkaavat järjestelmän loogista rakennetta, jotta se toimii järjestelmän fyysisten osien ja ohjelmistoteknologioiden kanssa. Kun järjestelmän fyysinen suunnittelu on valmis, projektissa alkaa kehitystyövaihe. Kehitystyövaiheessa ohjelmistokehittäjät rakentavat järjestelmän vaatimustenmäärittelydokumentin ja fyysisten vaatimusten perusteella. Sen jälkeen siirrytään testausvaiheeseen, jossa järjestelmä testataan ja siinä ilmenevät virheet raportoidaan ja korjataan. Tämän jälkeen järjestelmä luovutetaan käyttöönottoa varten ja asiakkaan järjestelmää ylläpidetään ja mahdollisesti jatkokehitetään. (Chi 31.10.2018).

Vesiputousmalli on tehokas työkalu tiettyjen reunaehtojen yhteydessä. Vesiputousmallin on todettu toimivan hyvin, kun vaatimukset on hyvin ymmärretty ja kehitettävä tuote on todistetusti jo pitkään toiminut järjestelmä. Vesiputousmalli suosii vakautta ennemmin kuin ketteryyttä, suunnittelua ennemmin kuin tutkimista ja dokumentointia ennemmin kuin keskustelua. (Resnick, Bjork & De la Maza 2011, 18).

Vesiputousmalli suosii vakautta, koska vakaille vaatimuksille on helppo suunnitella toimiva ratkaisu. Jos esimerkiksi rakennusyritys rakentaa siltaa, yrityksen on hyvä tietää mistä sillan pitää alkaa ja loppua ja millaisen painon sen pitää kestää. Vesiputousmalliin ei myöskään sovellu hyvin dynaamiset vaatimukset, koska niitä on vaikea budjetoida ja aikatauluttaa. Jos vaatimuksena on esimerkiksi siirtää ihmisiä Bostonista Cambridgeen mahdollisimman tehokkaasti, sen aikataulutusta ja budjettia on lähes mahdoton tehdä mahdollisten kuljetusvälineiden ja reittien takia. Ohjelmistokehityksessä asiakkaat kuvaavat yleensä ratkaisun (ihmisten siirtämisen) ennemmin kuin heidän ongelmiinsa sopivan tuotteen (silta). Tämä tekee vesiputousmallista hankalan sovelluskehitysmallin dynaamisessa ympäristössä. (Resnick ym. 2011, 18).

Vesiputousmallin yksi suurimpia haasteita on kriittinen suunnittelu. Jokainen vaihe on yleensä jo ennustettu toimivan valmiiksi ennen niiden toteuttamista. Alussa tehdyillä päätöksillä on siis suuri merkitys viimeisiin vaiheisiin. Siksi vesiputousmallin ensimmäisessä vaiheessa keskitytään vaatimustenmäärittelyyn ja järjestelmän suunnitteluun. Vesiputousmallin keskeinen idea on se, että virheiden löytäminen projektin alkuvaiheissa on paljon helpompi korjata kuin projektin loppuvaiheessa. Siltaesimerkissä sillan paikasta pitää olla 100% varma ennen kuin yhtäkään kiveä aletaan raivata. (Resnick ym. 2011, 19).

Dokumentaatio on pääkommunikointityökalu eri vesiputousmallien välillä. Suuri riippuvaisuus dokumentaatiosta sallii projektin eri vaiheiden aloituksen, asiantuntijoiden resurssoinnin eri vaiheille, tuloksen tuottamisen ja siirtymisen sulavasti seuraavaan vaiheeseen ennustettavalla tavalla. Dokumentaation avulla isot tiimit voivat vaihtaa työntekijöitä vaiheesta toiseen maksimoidakseen heidän työpanoksensa. Dokumentointi tuottaa usein kirjallisen raportin edistymisestä, jolloin projektista löytyy läpinäkyvyys päätöksenteon syihin, päätöksenteon aikaan ja päätöksen tekijään. (Resnick ym. 2011, 19).

Kun vaatimukset on kerätty projektin sidosryhmiltä, niistä tehdään dokumentti, joka määrittelee onnistumisen. Tämä dokumentti tulee olemaan ”sopimus” asiakkaan liiketoiminta-käyttäjien ja toimittajan teknisen sisällyttämistiimin kesken. Jos toimitettu järjestelmä täyttää tämän dokumentin vaatimukset, se yleensä todetaan silloin valmiiksi. Tämän takia kaikkien sidosryhmien tulisi ymmärtää täysin dokumentin sisältö. (Resnick ym. 2011, 19-20).

Kun tämä dokumentti on käyty läpi asiakkaan kanssa ja hyväksytty, siitä muokataan toiminnallinen vaatimusdokumentti (spec). Toiminnallinen vaatimusdokumentti kuvaa kaikki järjestelmän toiminnot ja sen pohjalta kehitystiimi alkaa rakentamaan tuotetta. Toiminnallista vaatimusdokumenttia seuraa myös vielä yksityiskohtainen suunnitteludokumentti, joka kuvaa teknologian suunnittelua. Sen tehtävä on kuvata toiminnallinen vaatimusdokumentti järjestelmän rakenteeseen. Tämän jälkeen vasta yleensä alkaa toteutustyö. (Resnick ym. 2011, 19-20).

Vesiputousmallin raskaan dokumentoinnin etuja ovat läpinäkyvyys ja jäljitettävyys, mutta siitä on myös haittaa. Raskas dokumentointi olettaa, että kaikki sidosryhmät lukevat kaikki dokumentit läpi, jota ei käytännössä koskaan tapahdu. Raskas dokumentointitapa olettaa myös, että sidosryhmien kaikki työntekijät ymmärtävät täysin näiden dokumenttien sisällön. Sidosryhmien työntekijät tunnetusti allekirjoittavat dokumentit siksi, että he luottavat dokumenttien kirjoittajiin. Tämä kuitenkin mahdollistaa kommunikointivirheestä johtuvia virheitä, jotka huomataan myöhemmässä vaiheessa projektia. (Resnick ym. 2011, 19-20).

2.2 Ketterä malli

Ketterällä kehityksellä tarkoitetaan kykyä luoda ja vastata muutokseen. Sen avulla pystytään hallitsemaan ja lopulta onnistumaan epävakaassa ympäristössä. Ketterä kehitys on sateenvarjotermi viitekehyksille ja käytännöille, jotka pohjautuvat ketterän ohjelmistokehityksen julistuksen ja sen 12 periaatteen arvoihin. (Agile Allience 2019). Ketterä sovelluskehitys on virallisesti luotu vuonna 2001, jolloin 17 eri alojen asiantuntijaa loivat ketterän ohjelmistokehityksen julistuksen (Fowler 9.7.2006).

Ketterän kehityksen tapaisia käytäntöjä näkyi ensimmäisen kerran 1990-luvulla. Eri organisaatioiden ohjelmistokonsultit ja ohjelmistokehittäjät olivat tyytymättömiä heidän tapaansa luoda tuotteita ja he rupesivat yhdistelemään vanhoja ja uusia ideoita kunnes niistä löytyi toimiva kokonaisuus oman tiiminsä sovelluskehityksen parantamiseen. Näitä keinoja olivat esimerkiksi läheisen yhteistyön painotusta kehitystiimin ja liiketoiminnan sidosryhmien kanssa, toistuvaa liiketoiminta-arvon toimittamista, tiiviiden itseohjautuvien tiimien käyttöä ja järkevämpiä tapoja rakentaa, hyväksyä ja toimittaa ohjelmakoodia. Kun asiantuntijat huomaisivat näiden keinojen toimivan omassa sovelluskehityksessään, he alkoivat jakaa niitä muille tiimeille viitekehyksinä. Näiden viitekehysten avulla nykyisin tunnetut ketterät viitekehykset, kuten Scrum, XP ja FDD, alkoivat hiljalleen muotoutua. (Agile Allience 2019).

Vuonna 2000 järjestettiin XP:n eli Extreme Programming-viitekehyksen omaaville asiantuntijoille oma tapaaminen, jossa keskusteltiin XP-viitekehyksestä ja muiden vastaavien viitekehysten tulevaisuudesta. Tapaamisen järjestäjä pyysi mukaan myös asiantuntijoita, jotka eivät ymmärtäneet XP-viitekehystä täysin, mutta jotka olivat omaksuneet muita vastaavia ketteriä viitekehyksiä. Tapaamisessa asiantuntijat päätyivät johtopäätökseen, että XP-viitekehyksessä ja muissa viitekehyksissä on paljon samaa ja että niiden yhdistämisestä olisi hyvä pitää laajempi kokous. (Fowler 9.7.2006).

Vuonna 2001 pidettiin kokous Snowbirdissä Yhdysvalloissa, johon oli kutsuttu kaikki ketteriin menetelmiin erikoistuneet aktiiviset asiantuntijat. Heistä 17 pääsi paikalle, jonka jälkeen he keskustelivat eri ketterien viitekehysten yhdistämisestä. Erimielisyyksiä oli paljon, mutta muutamasta asiasta päästiin yksimielisyyteen. Näistä yksimielisistä arvoista muodostui kokouksen jälkeen ketterän ohjelmistokehityksen julistus ja hieman myöhemmin asiantuntijat loivat julistuksen lisäksi 12 periaatetta. XP-viitekehyksestä ja muista ketteristä viitekehyksistä käytettiin ennen kokousta nimeä ”kevyet metodit”, mutta kokouksessa todettiin, että ketteryys toimisi parempana nimenä. Ketteryys eli Agile valittiin nimeksi siksi, että se vangitsi parhaiten ajatuksen sopeutumisesta ja muutokseen vastaamisesta. (Fowler 9.7.2006).

Vuoden 2001 jälkeen perustettiin Agile Allience, jossa kaikki sovelluskehityksen ammattilaiset pystyivät jakamaan omia käytäntöjään ja ideoitaan ja sen seurauksena organisaatiot ja tiimit alkoivat omaksua itselleen ketteriä kehityskäytäntöjä. Jotkut organisaatiot alkoivat kuitenkin omaksua omia periaatteita ja rituaaleja julistuksen ja 12 periaatteen sijaan, koska niiden toteuttaminen sellaisenaan on haastavaa. Tästä syystä nykyään lähes jokai-

sessa organisaatiossa on ainutlaatuinen tapa käyttää ketteriä viitekehyksiä. Agile Alliancen mukaan parhaassa tilanteessa organisaatiot eivät näe ketterää kehittämistä uutena ja erilaisena työkaluna, vaan yksinkertaisesti tapana lähestyä työtä. (Agile Allience 2019).

Ketterän ohjelmistokehityksen julistus ilmaisee ketterän ohjelmistokehityksen ytimen. Julistuksessa käsitellään neljää eri väittämää, joissa vasemmanpuolisia väittämiä arvostetaan enemmän kuin oikeanpuoleisia väittämiä. Tämä ei kuitenkaan tarkoita, että oikealla puolella olevia väittämiä ei arvostettaisi ollenkaan. Julistuksen väittämien mukaan ketterässä ohjelmistokehityksessä arvostetaan:

- Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja
- Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota
- Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja
- Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa

(Agilemanifesto 2001a).

Yksilöillä ja kanssakäymisillä on enemmän arvoa kuin menetelmillä ja työkaluilla. Sokea menetelmän noudattaminen ja hyvän ja nopean, mutta väärän työkalun käyttäminen ei tuo sovelluskehitykselle yhtä paljon arvoa kuin tiimin jäsenten ymmärtäminen. Jos esimerkiksi tiimin sisäistä tehokkuutta halutaan parantaa järkevällä uudistuksella, se pitää ensin myydä muulle tiimille ennen sen toteuttamista. Jos kehitystiimin jäsenet noudattavat uutta menetelmää ja työkalua ymmärtämättä sen merkitystä, he saattavat saada aikaiseksi ristiriitaisia ja sekavia tuloksia. Jokaisella tiimin jäsenellä on omat motivaationlähteensä, ideansa ja mieltymyksensä, jonka takia jokaisen tiimin jäsenen yhteistyö ja työpanoksen suhde muun tiimin tekemiseen pitää selvittää ennen kuin aletaan pohtimaan sopivia menetelmiä ja työkaluja. (Stellman & Greene 2015, 34).

Toisen julistuksen kohdan mukaan toimivassa ohjelmistossa on enemmän arvoa kuin kattavassa dokumentaatiossa. Toimivalla ohjelmistolla tarkoitetaan ohjelmistoa, joka tuottaa arvoa organisaatiolle. Toimivan ohjelmisto voi olla asiakkaille myytävä ohjelmisto tai oman yrityksen sisällä toimiva ohjelmisto, joka auttaa työntekijöitä tekemään työnsä tehokkaammin. Arvon tuotto ohjelmistossa tulee joko pääoman säästämisestä tai sen kasvattamisesta suhteessa ohjelmiston käyttöönottokuluihin. Dokumentointi on kuitenkin myös tärkeää. Dokumentoinnista on silloin eniten hyötyä ja lisäarvoa, kun se auttaa kehitystiimiä

ymmärtämään ratkaistavan ongelman, kommunikoimaan helposti ongelmasta järjestelmän käyttäjien kanssa ja ennaltaehkäisemään ja ratkaisemaan ongelmat ennen niiden sisällyttämistä ohjelmistoon. (Stellman & Greene 2015, 35-36).

Kolmannen julistuksen kohdan mukaan asiakasyhteistyö on arvokkaampaa kuin sopimusneuvottelut. Kaikkia projektin osapuolia pitäisi siis kohdella kuin he olisivat yhtä isoa tiimiä. Asiakasyhteistyö ulottuu sopimusten parissa olevien työntekijöiden lisäksi esimerkiksi ohjelmistokehittäjiin, testaajiin ja projektipäälliköihin. Kun nämä henkilöt eivät työskentele samassa tiimissä ja eivät tee töitä samaa tavoitetta eli toimivan ohjelmiston rakentamista kohti, he kohtelevat toisiaan ikään kuin näkymättömän sopimuksen mukaan ja eivät jaa silloin tärkeää tietoa tiimin- ja projektinrajan yli. Myös erilaiset palvelusopimukset lisäävät eroavuutta ohjelmistokehittäjien ja testaajien ja kehitystiimin ja asiakkaan välille. Jos esimerkiksi ohjelmistokehittäjä on yksin vastuussa jostain palvelusopimuksen yksityiskohdasta, on epätodennäköistä olettaa hänen samanaikaisesti kehittävän uusia ratkaisuja ja innovaatioita rakennetun ohjelmiston käyttäjille. (Stellman & Greene 2015, 36-37).

Viimeisessä julistuksen kohdassa muutokseen vastaamista pidetään arvokkaampana kuin suunnitelmassa pitäytymistä. Suunnitelmassa pysyminen ei ole muuttuvassa ympäristössä järkevää, koska silloin väärästä suunnitelmasta rakentuu väärä tuote. Tiimien pitäisi jatkuvasti etsiä muutostarpeita ja olla valmiita reagoimaan asiakkaan tarpeiden muutokseen. Muutos ei kuitenkaan ole usein miellyttävää ja se voi esimerkiksi viedä paljon projektipäällikön turhaa aikaa suunnitella koko loppuprojekti uusiksi. Muutos on kuitenkin hyvä tunnistaa mahdollisimman ajoissa, koska mitä myöhemmin asioita halutaan muuttaa, sitä hankalampaa se on. Kiteytettynä siis toimivan ohjelmiston toimittaminen on aina tärkeämpää kuin suunnitelman toteutus. (Stellman & Greene 2015, 37-38).

Ketterän ohjelmistokehityksen julistukseen kuuluu myös 12 periaatetta. Periaatteet ovat:

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. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.

9. Teknisen laadun ja ohjelmiston hyvän rakenteen jatkuva huomiointi edesauttaa ketteryyttä.

10. Yksinkertaisuus - tekemättä jätettävän työn maksimointi - on oleellista.

11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.

12. Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti.

(Agilemanifesto 2001b).

2.3 Scrum-viitekehys

Ketteriin menetelmiin lukeutuu useita eri viitekehyksiä, mutta opinnäytetyössä niistä tarkastellaan ainoastaan Scrum-viitekehystä. Scrum päätyi tarkastelun kohteeksi siksi, että osastolla on ollut ketteryyskoulutusta, joka on pohjautunut pääasiassa Scrum-viitekehykseen. Osastolla on myös alkamassa muutos ketterämpään suuntaan ja tähän ketterämpään suuntaan on toivottu nimenomaan Scrumin käytäntöjä, kuten tuoteomistajaa, sprinttejä ja läpinäkyvyyttä. Jotta opinnäytetyö pysyy kompaktina ja johdonmukaisena, on parempi keskittyä laajemmin Scrum-viitekehykseen.

2.3.1 Yleistä

Scrum on ketterien menetelmien viitekehys, jonka avulla voidaan ratkaista monimutkaisia ongelmia ja kehittää mahdollisimman suurta lisäarvoa tarjoavia tuotteita tuottavalla ja luovalla tavalla. Scrum on kevyt, helppo ymmärtää ja vaikea taitaa. Scrum ei ole prosessi, tekniikka tai yksityiskohtia ohjaava menetelmä, vaan viitekehys, jonka sisällä näitä voidaan hyödyntää. Scrum-viitekehys koostuu Scrum-tiimeistä ja niiden rooleista, tuotoksista, tapahtumista ja säännöistä. (Schwaber & Sutherland 2017, 3).

Scrum perustuu empirismiin eli kokemukseen ja päätöksentekoon vaikuttavien tunnettujen tosiasioiden kautta saatuun tietoon. Scrumissa hyödynnetään iteratiivisinkrementaalista

lähestymistapaa, jonka avulla pyritään riskien kontrolloimiseen ja ennustettavuuden optimoimiseen. Scrumin empiristiseen prosessinhallintaan kuuluu kolme tukipylvästä, jotka ovat läpinäkyvyys, tarkastelu ja sopeuttaminen. (Schwaber & Sutherland 2017, 3-4).

Läpinäkyvyys ilmenee Scrumissa niin, että prosessien merkittävät tekijät tulisi olla helposti saatavilla kaikille niille osapuolille, jotka vastaavat lopputuloksesta. Prosessien merkittävät tekijät pitää määritellä ja sopia yhdessä, jotta niiden tarkastelijoilla on yhteinen näkemys siitä, mitä tarkastellaan. Huono läpinäkyvyyden vallitessa päätökset huonontavat todennäköisemmin tuotoksen arvoa, ovat virheellisiä ja lisäävät riskejä. Esimerkkejä läpinäkyvyydestä ovat kaikkien osallisten yhteinen sanasto ja kaikkien työhön osallistuvien yhteinen ”valmiin” määritelmä. (Schwaber & Sutherland 2017, 4-5, 17).

Tarkastelu ilmenee niin, että käyttäjien tulee tarkastella Scrumin tuotoksia ja työn etenemistä kohti sprintin tavoitteita, jotta ongelmalliset poikkeamat voidaan havaita ajoissa. Tarkastelua ei saa kuitenkaan tehdä liian tiheästi, jotta se ei häiritse varsinaista työtä. (Schwaber & Sutherland 2017, 5).

Sopeuttaminen ilmenee niin, että tarkastelijan päätellessä yhden tai useamman prosessin osan olevan hyväksyttävän rajan ulkopuolella, työstettävää ainesta tai prosessia tulee muokata. Tämän avulla myöhemmät poikkeamat saadaan minimoiduiksi. Scrumin päätapahtumat tarkasteluun ja sopeuttamiseen ovat sprintin suunnittelu, päivittäispalaveri, sprintin katselmointi ja retrospektiivi. (Schwaber & Sutherland 2017, 5). Päätapahtumista kerrotaan tarkemmin luvussa 2.3.3.

Scrumia käytetään pääsääntöisesti teknologioiden ja tuotteen toimintojen empiiriseen havainnointiin, tuotekehitykseen ja niiden jalostamiseen, tuotteiden ja päivitysten julkaisemiseen monia kertoja päivässä, tuotantoympäristöjen hallintaan ja tuotteiden ylläpitoon. Scrumin arvoihin kuuluvat kunnioitus, keskittyminen, avoimuus, sitoutuminen ja rohkeus. Nämä vahvistavat toteutuessaan Scrumin kolmea tukipylvästä ja kasvattavat luottamusta tiimin sisällä. Scrumin onnistuminen on riippuvainen näiden viiden arvon omaksumisen onnistumisesta. Scrum-tiimin jokaisen jäsenen pitää sitoutua Scrum-tiimin tavoitteisiin, löytää rohkeutta tarttua vaativien ongelmien ratkaisemiseen, keskittyä sprintin työhön ja tavoitteisiin, sopia kaiken työn ja ongelmien avoimuudesta ja kunnioittaa kaikkia työntekijöitä kyvykkäinä ihmisinä. (Schwaber & Sutherland 2017, 3-5).

2.3.2 Scrum-tiimi

Scrum-tiimiin kuuluu tuoteomistaja, Scrummaster ja kehitystiimi. Scrum-tiimit ovat pieniä ja tiiviitä ja tiimejä on yleensä useita riippuen projektien koosta. Scrum tiimi ei ota ohjeita ohjaukseen tiimin ulkopuolelta ja kaikki tarvittava osaaminen löytyy työn suorittamiseen tiimin sisältä. Yksittäiset tiimit ovat myös sopeutuvia ja joustavia. Tiimimallin idea on siis luovuuden, joustavuuden ja tuottavuuden optimoiminen. (Schwaber & Sutherland 2017, 5).

Tuoteomistajan tehtävänä on vastata kehitystiimin työn tuloksena saatavan tuotteen kehitysjonon hallitsemisesta ja tuotteen arvon maksimoinnista. Tuoteomistaja pitää huolen tuotteen kehitysjonon selkeästä ilmauksesta ja tavoitteiden mukaan sen uudelleenjärjestämisestä. Tuoteomistaja pyrkii varmistamaan tuotteen kehitysjonon avoimuuden, läpinäkyvyyden ja ymmärrettävyyden koko Scrum-tiimille ja varmistaa sen, että kehitystiimi ymmärtää kehitysjonon riittävällä tarkkuudella. Tuoteomistaja vastaa myös kehitystiimin työn arvon maksimoimisesta ja vahvistamisesta. Tuoteomistajan ei itse ole välttämättä pakko tehdä näitä kaikkia töitä, mutta hän on kuitenkin aina vastuussa niistä. Scrum-viitekehyksessä kukaan ei saa pakottaa kehitystiimiä työskentelemään minkään muiden tavoitteiden kuin tuoteomistajan asettamien tavoitteiden kanssa. (Schwaber & Sutherland 2017, 6).

Kehitystiimi koostuu tuotteen kehittämisestä vastaavista ammattilaisista, jotka muuttavat tuotteen kehitysjonon sisällön mahdollisesti julkaisukelpoiseksi “valmiiksi” inkrementiksi jokaisessa iteraatiossa. Kehitystiimit ovat täysin itseohjautuvia eli kukaan muu ei saa kertoa kehitystiimille, kuinka kehitystiimi luo julkaisukelpoisen inkrementin tuotteen kehitysjonosta. Kehitystiimi koostuu 3-9 asiantuntijasta ja kehitystiimin pitää pystyä itsenäisesti toteuttamaan potentiaalisesti julkaisukelpoinen inkrementti. Kehitystiimin jäsenillä voi olla tiettyjä erikoisosaamisia ja painopisteitä, mutta kehitystiimi ei tunnusta alitiimejä, jotka vastaisivat erityisistä osa-alueista. Scrumissa ei myöskään kehitystiimin sisällä ole varsinaisia titteleitä. (Schwaber & Sutherland 2017, 6-7).

Scrummasterin tehtävänä on vastata ketteryyden ja varsinkin Scrumin edistämisestä ja tukemisesta auttamalla kaikkia ymmärtämään Scrumin teorian, säännöt ja käytännöt. Scrummaster pyrkii myös tuottamaan Scrum-tiimille mahdollisimman suuren arvon muuttamalla kehitystiimin ulkoisten sidosryhmien toimintatapoja. (Schwaber & Sutherland 2017, 7).

Scrummasterin rooli vaihtelee sen mukaan, ketä hän palvelee. Jos Scrummaster palvelee tuoteomistajaa, niin silloin hän auttaa tuotteen kehitysjonon tehokkaassa hallinnassa, kehitysjonon selkeyden tärkeyden ymmärtämisessä ja tuoteomistajan opastuksessa tuotteen kehitysjonon järjestämisestä sellaiseksi, että se tuottaa mahdollisimman paljon arvoa. Tämän lisäksi Scrummasterin pitää ymmärtää tuotesuunnittelua kokemuksiin ja havaintoihin pohjautuvassa ympäristössä ja varmistaa tavoitteiden, julkaisujen laajuuden ja tuotteen liiketoiminta-alueen mahdollisimman hyvän ymmärtämisen Scrum-tiimin kaikille jäsenille. Ketteryyden harjoittamisen lisäksi Scrummasterin pitää pystyä tarpeen mukaan johtamaan Scrumin eri tapahtumia. (Schwaber & Sutherland 2017, 7).

Scrummasterin rooli kehitystiimin kannalta on valmentaminen ja työn lisäarvon maksimoiminen. Scrummaster valmentaa kehitystiimiä Scrumin käytännöissä tarpeen mukaan ja pyrkii auttamaan tiimiä muuttumaan itseohjautuvaksi ja moniosaavaksi. Scrummaster pyrkii myös poistamaan haitallisia esteitä kehitystiimin työn edistymiseltä ja johtamaan tarpeen mukaan Scrumin tapahtumia. (Schwaber & Sutherland 2017, 9).

Organisaation kannalta Scrummaster pyrkii johtamaan ja valmentamaan eri sidosryhmiä Scrumin käyttöönotossa ja parantaa jatkuvasti Scrumin käytön tehokkuutta. Scrummaster suunnittelee Scrumin toteutusta organisaatiossa ja työskentelee muiden Scrummasterien kanssa maksimoidakseen Scrumin käytön hyödyt. Scrumin käytön opastaminen ja muutosten tekeminen Scrum-tiimin tuottavuuden lisäämiseksi ovat myös Scrummasterin oleellisia tehtäviä. (Schwaber & Sutherland 2017, 9).

2.3.3 Tapahtumat

Scrumin tapahtumien tarkoituksena on tarjota mahdollisuus sopeuttaa ja tarkastella eri asioita eli lisätä läpinäkyvyyttä. Tapahtumilla pyritään myös luomaan säännöllisyyttä ja minimoimaan palaverien tarvetta lukuun ottamatta Scrum-palavereja. Tapahtumat ovat myös aikarajattu ja niiden keskiössä on hukan minimointi. (Schwaber & Sutherland 2017, 9).

Scrumin päätapahtuma on sprintti, joka kestää aina kerrallaan 1-4-viikkoa. Sprintin aikana tuotetaan ”valmiin” määritelmän täyttävä, käyttökelpoinen ja potentiaalisesti julkaisukelpoinen inkrementti. Sprintin aikana voidaan luoda useampia versioita, joista inkrementti koostuu. Kun sprintin koko on määritelty, niin se on koko kehitysajan samanpituinen ja seuraava sprintti aloitetaan heti edellisen sprintin päätyttyä. Sprintin aikana sprintin tavoitetta vaarantavia muutoksia ei tehdä, laadun tulee pysyä tavoitteen mukaisena ja sisältöä voidaan tarkentaa opittujen asioiden pohjalta. (Schwaber & Sutherland 2017, 10).

Sprintin aikana tapahtuvat vaiheet ovat sprintin suunnittelupalaveri, päiväpalaveri, kehitystyö, sprintin katselmointi ja sprintin retrospektiivi. Sprintin pitää sisältää aina sprintin määritelmä, mitä sprintin aikana toteutetaan, sprintin toteutuksen suunnittelu, joustava suunnitelma, joka ohjaa toteutusta ja varsinaisen työn ja työn tuloksena syntyvän inkrementin. (Schwaber & Sutherland 2017, 10).

Sprinttejä ei yleensä keskeytetä, mutta jos tuoteomistaja näkee sprintin tavoitteen muuttuneen tarpeettomaksi tai että sen toteuttaminen ei ole muuten kannattavaa, se yleensä keskeytetään. Keskeytyneen sprintin tuotteen kehitysjonon valmiiksi saadut kohdat yleensä hyväksytään osaksi inkrementtiä. Muut kehitysjonon kohdat arvioidaan uudelleen ja siirretään sen jälkeen joko osaksi inkrementtiä tai takaisin kehitysjonoon. (Schwaber & Sutherland 2017, 10).

Sprintin suunnittelupalaverissa suunnitellaan sprintissä tapahtuva työ. Koko Scrum-tiimi osallistuu suunnitteluun ja sen kesto on korkeintaan kahdeksan tuntia riippuen sprintin kestosta. Sprintin suunnittelussa keskitytään kahteen asiaan: mitä sprintin aikana tehdään ja miten tämä työ tullaan toteuttamaan. Ensin kehitystiimi antaa arvion toiminnallisuudesta, joka ehdittäisiin toteuttaa sprintin aikana. Tämän jälkeen tuoteomistaja keskustelee tiimin kanssa sprintin tavoitteista ja tuotteen kehitysjonon mahdollisista kohdista, jotka voitaisiin toteuttaa sprintin aikana. Suunnittelupalaverissa keskustellaan myös kehitystiimin edellisen sprintin suorituskyvystä ja kapasiteetista, jotka vaikuttavat seuraavan sprintin työmäärään. Kehitystiimi päättää kuitenkin työmäärän kokonaisuudessaan itse ja asettaa koko Scrum-tiimin kanssa yhteisen tavoitteen sprintille. (Schwaber & Sutherland 2017, 11).

Tämän jälkeen kehitystiimi suunnittelee, miten se saa sprintin työmäärän toteutettua ”valmiiksi” inkrementiksi. Sprintin kehitysjonon kohdat ja suunnitelma niiden toteuttamiseen muodostavat sprintin kehitysjonon. Kehitystiimi suunnittelee aluksi työn ja toteutettavat toiminnallisuudet. Työn määrä voi riippua sprintistä, mutta sitä tulisi aina olla riittävä määrä, jotta koko sprintti saadaan hyödynnettyä. Sprintin ensimmäisten päivien työ pilkotaan enintään päivän kestäviin yksiköihin, jotta sprintti saadaan aloitettua tehokkaasti. Tuoteomistaja on myös usein organisointiapuna suunnitteluvaiheessa. (Schwaber & Sutherland 2017, 11).

Sprintin ja tuotteen kehitysjonon suunnittelussa käytetään yleensä käyttäjätarinoita eli User storeja. Käyttäjätarina on kuvaus tietystä tavasta, millä käyttäjä tulee käyttämään tuotetta. Ne ovat yleensä muutaman lauseen pituisia ja ne vastaavat kysymyksiin kuka

käyttäjä on, mitä käyttäjä haluaa tehdä ja miksi käyttäjä haluaa tehdä niin. Käyttäjätarinoiden peruskuvaus kirjoitetaan ”muistilapun” etupuolelle ja ”muistilapun” toiselle puolelle kirjoitetaan tyytyväisyysehdot tai hyväksymiskriteerit. Kehitystiimi tekee yleensä tuoteomistajan kanssa hyväksymiskriteerit peruskuvauksen jälkeen ja niihin kirjoitetaan muutama tarkka asia, mitä käyttäjän pitäisi pystyä tekemään inkrementillä sen jälkeen, kun käyttäjätarina on valmis. Tämän avulla kehitystiimillä on konkreettinen valmiin määritelmä käyttäjätarinasta, koska ennen kuin käyttäjä voi toteuttaa tuotteessa kaikki hyväksymiskriteerit, käyttäjätarina ei ole valmis. Käyttäjätarinoiden luonti on yleensä tuoteomistajan vastuulla. (Stellman & Greene 2015, 143-145).

Kun sprintin käyttäjätarinat on määritelty, niille annetaan työmääräarvio. Työmääräarviota ei anneta henkilötyöpäivinä, vaan siihen käytetään tarinapisteitä eli story pointseja. Tarinapisteiden avulla saadaan selville työn nopeus jokaisen sprintin aikana ja niiden avulla on helpompi tuottaa ja kehittää tiimin työmääräarvioita. Tarinapisteiden laskemiseen käytetään lukuisia eri tapoja, mutta tärkeintä niiden käytössä on se, että koko Scrum-tiimi käyttää samaa laskemistapaa koko projektin ajan. Tarinapisteitä tarkastellaan myös sprintin aikana niin, että rakennetaan ”burndown chart” eli koordinaatisto, jossa x-akselilla on sprintin päivät ja y-akselilla on tarinapisteet. Tämän jälkeen koordinaatistoon lisätään aloituspäivän kohdalle ”pallo” ja siitä vedetään suora linja siihen päivään, kun sprintti on valmis. Tämän jälkeen jokaisen sprintin päivänä ”palloa” siirretään jäljellä olevien päivien ja jo valmistuneiden tarinapisteiden mukaan. Jos huomataan, että ollaan lineaarisen suoran linjan yläpuolella, työmäärää ja aikataulua pitää muuttaa ja se voidaan huomata jo päivätasolla. Jos taas huomataan olevan suoran viivan alapuolella, työmäärä on liian pieni ja sitä pitää lisätä. Tarinapisteitä suositellaan käytettävän siksi, että ne ovat yksinkertaisia, konkreettisia, tiimin kontrolloitavissa, keino avoimempaan keskusteluun työmääräarvioista, mieluisia ohjelmistokehittäjille, keino välttää käyttäjätarinoiden epäselvyyksien syntyminen ja väline auttaa tiimiä aidosti sitoutumaan työhönsä. (Stellman & Greene 2015, 146-151).

Sprintin suunnittelu kannattaa aloittaa arvokkaimpien käyttäjätarinoiden kautta niin, että ensin etsii pienimmän käyttäjätarinan ja asettaa sille suurin piirtein saman tarinapistearvon kuin edellisen sprintin samankokoiselle käyttäjätarinalle. Tuoteomistajan on samalla hyvä keskustella kehitystiimin kanssa arvioiden todenmukaisuudesta. Kun tätä jatketaan jokaisen käyttäjätarinan kohdalla, saadaan lopulta sprintin keskimääräinen työmääräarvio eli tarinapisteiden määrä. Tarinapisteiden määrä sprintissä ei koskaan ole vakio, vaan kehitystiimi pyrkii jatkuvasti parantamaan suoritustaan, jotta he saavat arvioitua aina tarkemman pistemäärän seuraavaa sprinttiä kohden. (Stellman & Greene 2015, 146-147).

Päiväpalaveri on sprintin jokaisena päivänä pidettävä palaveri, jossa maksimissaan 15-minuutin ajan keskustellaan siitä, mitä seuraavan vuorokauden aikana tehdään ja mitä edellisenä päivänä tehtiin. Palaveri pidetään joka päivä samaan aikaan samassa paikassa ja sen avulla tiimin suorituskyvyn ja yhteistyön on todettu parantuvan. Päiväpalaverin aikana kehitystiimi toimii itseohjautuvasti eli se päättää itse päiväpalaverin sisällön ja palaverin toteutuksen. (Schwaber & Sutherland 2017, 12-13).

Päiväpalaveri toteutetaan yleensä kysymyksien avulla tai yleiseen keskusteluun pohjautuen. Päiväpalaverissa pitäisi saada selville, mitä kehitystiimin jokainen jäsen sai aikaan eilisenä päivänä saavuttaakseen sprintin tavoitteen, mitä jäsen aikoo tehdä tänään tavoitteen saavuttamisen eteen ja onko jäsen havainnut tavoitteen saavuttamista vaikeuttavia esteitä. Päiväpalavereissa päivitetään työmäärää palaverin lopuksi, jotta sprintin tavoitteen saavuttamista on helpompi mitata. Scrummaster pitää huolta siitä, että kehitystiimi järjestää joka päivä päiväpalaverin ja että se ei veny 15-minuutin yli. (Schwaber & Sutherland 2017, 12-13).

Jokaisen sprintin lopussa pidetään sprintin katselmointi, jossa käydään läpi kehitetty inkrementti. Katselmointiin osallistuu Scrum-tiimi ja tuoteomistajan kutsumat tärkeimmät eri sidosryhmien edustajat. Katselmoinnissa keskustellaan seuraavasta kehitettävästä asiasta, jotta arvon tuottaminen saadaan optimoitua. Tuoteomistaja kertoo katselmoinnissa mikä osa työstä on ”valmista” ja mikä osa työstä on vielä tekemättä. Tuoteomistaja antaa katselmoinnissa arvion siitä, milloin tämänhetkisen edistymisten perusteella tuote on valmis. Kehitystiimin rooli katselmoinnissa on kertoa oma palaute siitä, missä he onnistuivat, mitä ongelmia he kohtasivat ja miten he ratkaisivat ne. Kehitystiimi esittelee myös inkrementin katselmoinnin osallistujille ja vastaa heidän kysymyksiinsä. (Schwaber & Sutherland 2017, 13-14).

Sprintin katselmoinnissa ei keskitytä työn etenemiseen, vaan siellä haetaan palautetta ja pyritään edistämään vuoropuhelua. Katselmoinnissa kaikki osallistujat pohtivat parhaita tapoja jatkaa työtä seuraavaa suunnittelupalaveria varten. Katselmoinneissa käydään läpi myös markkinatilanne, mahdolliset toiminnallisuudet, julkaisun aikataulu ja budjetti. Katselmointi kestää kuukauden sprintissä enintään neljä tuntia ja Scrummaster pitää huolen siitä, että katselmointi järjestetään ja että kaikki osapuolet ymmärtävät sen merkityksen. Katselmoinnin lopuksi tuotteen kehitysjonoa sopeutetaan palautteen mukaan ja siihen lisätään seuraavan sprintin tehtävät ja kehitysmahdollisuudet. (Schwaber & Sutherland 2017, 13-14).

Sprintin katselmoinnin jälkeen pidetään retrospektiivi, jossa Scrum-tiimi tarkastelee omaa toimintaansa ja tekee suunnitelman kehitysprosessin parannuksille seuraavaan sprinttiin. Retrospektiivissä tarkastellaan edellistä sprinttiä liittyen sen ihmisiin, prosesseihin, työkaluihin ja yhteistyöhön. Scrum-tiimi pyrkii tunnistamaan hyvät ja parannusta vaativat käytännöt ja luomaan suunnitelman työskentelytapojen parantamiseksi. Parannukset liittyvät usein viitekehyksen puitteissa oleviin kehitysprosessin muutoksiin. Retrospektiivin jälkeen aloitetaan seuraavan sprintin suunnittelu. Retrospektiivi kestää maksimissaan kolme tuntia ja Scrummaster pitää huolta siitä, että retrospektiivi järjestetään ja että kaikki ymmärtävät sen tarkoituksen. (Schwaber & Sutherland 2017, 14).

2.3.4 Tuotokset

Scrumin tuotokset kuvaavat työmäärää tai lisäarvoa lisäämällä läpinäkyvyyttä ja tilanteita niiden tarkastelulle ja sopeuttamiselle. Scrumin tuotosten idea on maksimoida läpinäkyvyys, jotta kaikilla on yhtenäinen käsitys tuotoksesta. Scrumin tuotoksia ovat tuotteen kehitysjono, sprintin kehitysjono ja inkrementti. (Schwaber & Sutherland 2017, 15).

Tuotteen kehitysjono on järjestetty lista kaikesta mitä tiedetään tuotteeseen tarvittavan ja ainoa dokumentti tuotteessa toteutettaville vaatimuksille ja muutoksille. Tuoteomistaja vastaa kehitysjonosta ja kaikki saman projektin Scrum-tiimit työskentelevät saman kehitysjonon kanssa. Tuotteen kehitysjonoa muokataan koko tuotteen kehityksen ajan ja sitä pyritään muokkaamaan koko ajan tarkoituksenmukaisemmaksi ja kilpailukykyisemmäksi. Tuotteen kehityksen alussa kehitysjonoon lisätään parhaiten tunnetut vaatimukset ja se kehittyy samalla kun tuote ja sen käyttöympäristö kehittyvät. (Schwaber & Sutherland 2017, 15).

Tuotteen kehitysjono pitää sisällään kaikki ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset, jotka tullaan toteuttamaan tuleviin inkrementteihin. Tämän lisäksi jokainen kehitysjonon kohta sisältää kuvauksen, järjestyksen, työmääräarvion ja arvon. Kohdat todetaan ”valmiiksi” siinä kohtaa, kun niille luotu testikuvaus on saatu toteutettua. (Schwaber & Sutherland 2017, 15-16).

Tuotteen kehitysjonoa myös jalostetaan eli siihen lisätään yksityiskohtia, työmääräarvioita ja muokataan sen kohtia. Jalostus tapahtuu tuoteomistajan ja kehitystiimin yhteistyöllä ja se on toistuva prosessi. Jalostamisessa arvioidaan kehitysjonon kohtia ja kehitystiimi käyttää yleensä maksimissaan 10% sen kapasiteetista jalostukseen. Kehitysjonon päivitystä tehdään myös muulloin kuin jalostuksen aikana tuoteomistajan toimesta. (Schwaber & Sutherland 2017, 15-16).

Tuotteen kehitysjono rakennetaan niin, että yläpäässä olevat kohdat ovat yksityiskohtaisempia ja selkeämpiä kuin kehitysjonon alemmat kohdat. Täten alemmat kohdat ovat usein helpompia tehdä kuin ylemmät kohdat. Kun kehitysjonosta valitaan seuraavaan sprinttiin kohdat, niiden täytyy olla jalostettu niin pieneen osaan, että ne pystytään toteuttamaan sprintin aikana. Työmääräarviot määräytyvät täysin kehitystiimin kautta, mutta tuoteomistaja voi olla mukana auttamassa tiimiä ymmärtämään kehitysjonon kohtien tarkoituksen. (Schwaber & Sutherland 2017, 15-16).

Sprintin kehitysjono pitää sisällään sprinttiin valittujen tuotteen kehitysjonon kohdat ja suunnitelman toimittaa inkrementti ja saavuttaa sprintin tavoite. Sprintin kehitysjonon avulla kehitystiimi tekee työnsä näkyväksi tarkastelua varten ja edistää jatkuvaa kehitystä niin, että kehitysjonoon lisätään edellisen retrospektiivin perusteella vähintään yksi suuren prioriteetin muutos. Sprintin kehitysjonon ei tarvitse olla täysin valmis sprintin alettua, vaan sen pitää olla tarpeeksi yksityiskohtainen, jotta sen edistyminen huomataan päiväpalavereissa. Jos kehitystiimi huomaa, että kaikkea tarvittavaa työtä sprintin tavoitteen saavuttamiseksi ei ole otettu sprintin kehitysjonoon mukaan, tarvittavat muutokset lisätään sprintin kehitysjonoon sprintin aikana. Vastaavasti jos kehitystiimi huomaa, että jokin sprintin kehitysjonon osa on tarpeeton, se otetaan pois kehitysjonosta. Kehitysjono on koko sprintin ajan läpinäkyvä kaikille ja sitä päivitetään jatkuvasti sprintin aikana. Ainoastaan kehitystiimi saa sprintin aikana muokata sprintin kehitysjonoa. (Schwaber & Sutherland 2017, 16-17).

Inkrementti pitää sisällään kaikki tuotteen kehitysjonon kohdat, jotka ovat valmistuneet nykyisen sprintin ja aiempien sprinttien aikana. Inkrementin tulee sprintin lopussa täyttää Scrum-tiimin yhteinen ”valmiin” määritelmä ja olla käyttökelpoisessa kunnossa. Jokainen inkrementti julkaistaan edellisten inkrementtien päälle ja se pitää olla läpikotaisin testattu, jotta voidaan olla varmoja inkrementin toimivuudesta. Inkrementtien tavoitteena on päästä lähemmäs lopullisen tuotteen visiota ja tavoitetta. (Schwaber & Sutherland 2017, 17).

Kaikkien osapuolten tulee ymmärtää, mitä ”valmis” tarkoittaa tuotteen kehitysjonon kohdan tai inkrementin yhteydessä. Valmis työ pitää olla aina kaikilla Scrum-tiimin jäsenillä tiedossa, vaikka eri tiimeillä olisikin erilainen määritelmä oman tiimin valmiille työlle. Kun jokaisen sprintin tavoitteena on toimittaa potentiaalisesti julkaistava inkrementti, valmiin määritelmä auttaa kehitystiimiä valitsemaan oikeat kohdat tuotteen kehitysjonosta. Inkrementin valmiiksi määrittelemiseen voi liittyä myös kehitysorganisaation vaatimia standar-

deja ja määräyksiä, joita kaikkien Scrum-tiimien täytyy vähintään noudattaa. Muussa tapauksessa kehitystiimi määrittelee inkrementin valmiuden. (Schwaber & Sutherland 2017, 17-18).

2.3.5 Scrum-viitekehyksen haasteet

Scrum-viitekehyksessä on myös suuria haasteita. Scrum-viitekehys toimii läpinäkyvänä asiakkaalle, uusiin rooleihin voi olla vaikea tottua ja viitekehyksen sisällyttäminen ei ole helppoa. (Wadhwa 26.3.2019).

Scrumiin kuuluu tärkeänä arvona läpinäkyvyys. Kun organisaatio siirtyy Scrumiin, sen pitäisi pystyä toimittamaan jokaisessa iteraatiossa julkaisukelpoinen inkrementti. Jos organisaatiolla on esimerkiksi teknistä velkaa tuotekehityksessään ja organisaation laatuprosessit eivät ole ajan tasalla, niin asiakas saa tämän selville heti ensimmäisen iteraation aikana. Scrum tarjoaa ikään kuin peilin organisaatiota vasten, jossa organisaatio saattaa syyttää viitekehystä ongelmistaan objektiivisesti todistetun palautteen sijaan. (Wadhwa 26.3.2019).

Scrumin rooleihin on myös vaikea sopeutua. Projektipäälliköt esimerkiksi eivät halua päästä irti vallastaan ja monet työntekijät joutuvat astumaan ulos mukavuusalueeltaan. Esimerkiksi jos työntekijä on tehnyt viimeiset 20 vuotta hyvää työtä omissa työtehtävissään, hänen voi olla vaikea ymmärtää uutta sitoutunutta kehitystiimimallia. (Wadhwa 26.3.2019).

Scrum on myös vaikea viitekehys käytännön kannalta. Viitekehyksen teoria ja käytännöt ovat helposti omaksuttavissa, mutta organisaatio toimii harvoin niin yksinkertaisesti, kun Scrum opettaa. Organisaatio ei myöskään aina huomioi viitekehyksen kokonaisuuden tärkeyttä, vaan omaksuu viitekehyksestä ainoastaan itselle tärkeimmät käytännöt. Esimerkiksi organisaatio saattaa tarvita asiantuntevan, päättäväisen ja sitoutuneen tuoteomistajan, mutta tällaisen henkilön löytäminen voi olla hankalaa. Kun sitten Scrum ei toimi epäpätevän tuoteomistajan takia, organisaatio olettaa viitekehyksen olevan hyödytön. (Wadhwa 26.3.2019).

Scrumin haasteita voivat olla myös dokumentointi, kommunikointi, asiakkaan osallistuminen, työympäristö ja Scrumin tapahtumat. Scrumissa dokumentointia harrastetaan paljon, mutta pääsääntöisesti se keskittyy aina iteraatioiden suunnitteluun. Vesiputousmallin perinteistä vaatimustenmäärittelyosiota ei ketterissä menetelmissä tunneta, vaan asiakas

päättää joka iteraatiossa itse tärkeimmät vaatimukset, jotka toteutetaan. Jos kehitystiimissä on kokeneita ohjelmistokehittäjiä, iteraatiokohtainen dokumentointi riittää heille hyvin kehitystyön toteuttamiseen. Kuitenkin jos uusi ja kokematon ohjelmistokehittäjä tulee työskentelemään saman ohjelmakoodin parissa kesken kehitystiimin iteraatiota, hänellä kestää aikaa ymmärtää järjestelmän rakenne ja hän joutuu kyselemään useasti kokeneemmilta ohjelmistokehittäjiltä käytännöistä, joka vie taas kokeneempien ohjelmistokehittäjien omaa työaikaa. Monet ohjelmistokehittäjät ovat myös tottuneet lukemaan määrittelydokumentteja, mutta ketterissä menetelmissä niitä harvemmin ylläpidetään tai käytetään ollenkaan. (Cho 2008).

Kommunikaatio toimii Scrumissa hyvin kehitystiimin sisällä, mutta suuret ongelmat ilmaantuvat silloin, kun saman projektin eri tiimit tai eri projektien tiimit eivät kommunikoi keskenään. Tästä voi aiheutua samojen asioiden kahdesti tekemistä. Tämä on merkittävä ongelma ketterässä sovelluskehityksessä ja sen korjaamiseksi on ehdotettu päiväpalavereiden pitoa eri tiimien välillä. Lisäksi asiakkaalta saatu palaute sprinttien lopuksi voi usein jäädä kovin minimaaliseksi, koska Scrum katselmoinnit voivat viedä paljon aikaa asiakkaan oikeista työtehtävistä, jolloin Scrumin idea jatkuvasta palautteesta kärsii. (Cho 2008).

Asiakkaan osallistuminen on muullakin tavoin hankalaa. Scrumissa olisi hyvä saada asiakkaalta joka viikko tai jopa joka päivä palautetta, mutta usein asiakkaan edustajat ovat niin kiireisiä, että heillä ei ole aikaa keskustella toteutuksesta kehitystiimin kanssa. Asiakkaan osallistumisella on myös se ongelma, että asiakas ajattelee tietävänsä mitä haluaa, mutta se ei kuitenkaan pidä paikkaansa. Esimerkiksi jos asiakas haluaa omaan järjestelmäänsä toiminnallisuuden, jolla voidaan seurata käyttäjien bonuspisteitä, se on heidän mielestään helppo ja yksinkertainen ominaisuus. Kuitenkin itse kehitystiimi tarvitsee tämän toteuttaakseen tiedot siitä, kuka bonuspisteitä käyttävä käyttäjä on, mitä bonuspisteet ovat, milloin ne vanhentuvat, kuinka kauan niitä seurataan ja mitä palkintoja käyttäjät saavat käyttämistään bonuspisteistä. Jos näihin kysymyksiin ei saada vastauksia, kehitystyö viivästyy ja kehitystiimi turhautuu. (Cho 2008).

Työympäristökin tuo Scrumissa joitakin ongelmia. Työympäristön olisi hyvä olla avoin tila tiimille, jotta he voivat aina tarpeen tullen heti kysyä apua tai keskustella toteutuksesta. Avoimen tilan haasteet ilmenevät kuitenkin silloin, jos joku ohjelmistokehittäjä yrittää tehdä työtään samalla kun kaksi muuta asiantuntijaa keskustelevat kovaäänisesti jostain muusta. Tällöin ohjelmistokehittäjä ei voi keskittyä omaan työhönsä ja hänen tehokkuuteensa heikkenee merkittävästi. (Cho 2008).

Scrumin tapahtumista taas päiväpalaverit ovat hankalia järjestää järkevästi. Päiväpalaverit voivat olla toisinaan todella yksinkertaisia ja monet ohjelmistokehittäjät voivat kokea turhaksi kuunnella jonkun toisen eilen tekemää työtä, jos se ei liity heidän työhönsä millään tavalla. Lisäksi päiväpalaverin aikatauluttaminen on kovin haastavaa. Päiväpalaveri olisi hyvä pitää aina samaan aikaan, mutta joustavien työtuntien takia se on lähes mahdotonta. Kehitystiimiin voi myös kuulua eri maista olevia työntekijöitä, jolloin fyysisen päiväpalaverin pitäminen on käytännössä mahdotonta. Jos palaveri pidetään esimerkiksi puoli kymmeneltä aamulla, puoli seitsemän aloittaneet asiantuntijat voivat kokea palaverin häiritseväksi, koska se rikkoo heidän keskittyneen, monta tuntia kestäneen työtilansa. Jos taas palaveri pidetään päivän lopussa, kaikki eivät pääse siihen tai se häiritsee myöhemmin tulleita asiantuntijoita. Kun joku lähtee puoli neljä ja toinen puoli kuusi, päivän loppumisella on liian suuri väli päiväpalaverin pitämiselle. (Cho 2008).

2.4 Projektien mittaaminen

2.4.1 Yleistä

Mittari on tarkasti määritelty menetelmä, jonka avulla kuvataan tietyn menestystekijän eli keskeisten liiketoiminnan menestymisen ja strategian toteutumisen kannalta olevien asioiden suorituskykyä. Mittareista muodostetaan mittauskohteen mukaan kokonaisuuksia, joita kutsutaan mittaristoksi. Mittaristoista taas löytyy useita erilaisia mittaristomalleja, joiden avulla organisaatiot muodostavat omia suorituskykymittaristojaan. Yksi tunnetuimmista mittaristomalleista on Robert Kaplanin ja David Nortonin kehittämä tasapainotettu mittaristo eli Balanced Scorecard. (Lönnqvist, Kujansivu & Antikainen 2006,13-14). Yleisimpiä mittareita organisaatioissa ovat Key Result Indicator eli keskeiset tulosmittarit, Result Indicator eli tulosmittarit, Performance Indicator eli suorituskykymittarit ja Key Performance Indicator eli keskeiset suorituskykymittarit. Näitä yleisiä mittareita selvennetään yleensä niin, että ne ovat kuin sipuli, jonka uloimmilla kerroksilla käydään läpi sipulin yleistä hyvinvointia eli keskeisiä tulosmittareita. Kun sipulia kuoritaan lisää, sen sisältä löytyvät suorituskyky- ja tulosmittarit ja sipulin ydin taas koostuu keskeisistä suorituskykymittareista. (Parmenter 2010, 1-2).

Keskeiset tulosmittarit ovat tuloksia monien tapahtumien seuraamuksesta. Ne antavat selkeän suunnan siitä, mihin suuntaan organisaatio on menossa. Ne eivät kuitenkaan kerro, mitä pitää muuttaa, tulosten parantamisen saavuttamiseksi. Keskeisiä tulosmittareita ovat esimerkiksi asiakastyytyväisyys, nettotuotto ennen veroja, asiakkaiden tuottavuus ja työntekijätyytyväisyys. Keskeiset tulosmittarit pitävät sisällään pitkän aikavälin tietoa kvartaalien tai kuukausien välein. (Parmenter 2010, 2-3).

Keskeisten tulosmittareiden ja keskeisten suorituskykymittareiden välissä ovat suorituskyky- ja tulosmittarit. Suorituskykymittarit auttavat tiimejä sijoittautumaan organisaation strategiaan. Suorituskykymittarit eivät ole taloudellisia, mutta ne tukevat keskeisiä suorituskykymittareita. Suorituskykymittareita ovat esimerkiksi prosentuaalinen nousu myynnissä tuottavimpien asiakkaiden keskuudessa, pääasiakkaiden valitukset ja myöhäiset toimitukset pääasiakkaille. Tulosmittarit taas ovat taloudellisia mittareita, kuten viikoittainen myyntianalyysi ja sairaaloiden sänkyjen viikoittainen käyttöaste. Tulosmittarit mittaavat toimintaa, mutta eivät kerro mitä pitäisi tehdä enemmän tai vähemmän. (Parmenter 2010, 3-4).

Keskeisille suorituskykymittareille on tyypillistä keskittyä organisaation suorituskyvyn osiin, jotka ovat kriittisiä nykyisen ja tulevaisuuden asioiden onnistumiselle. Jokainen organisaatio tuntee keskeiset suorituskykymittarit, mutta usein niitä ei mitata, koska niitä ei ole löydetty tai niitä ei ole käytetty moniin vuosiin. Keskeisille suorituskykymittareille on määritelty seitsemän ominaisuutta, joiden mukaan ne eivät ole taloudellisia mittareita, niitä mitataan usein, niitä käyttävät toimitusjohtaja ja johtotiimit, ne indikoivat selkeästi, mitä työntekijöiltä vaaditaan, ne ovat mittareita, joiden vastuu voidaan sitoa tiimille, niillä on merkittävä vaikutus ja ne rohkaisevat asianmukaiseen toimintaan. (Parmenter 2010, 4-7).

Suorituskykyä halutaan mitata myös kokonaisvaltaisesti, jonka yhteydessä käytetään usein tasapainotettua mittaristoa. Tasapainotetun mittariston keskiössä on organisaation visio ja strategia ja se jakautuu perspektiiveihin, joita ovat taloudellinen näkökulma, asiakasnäkökulma, sisäisten prosessien näkökulma ja oppimis- ja kasvunäkökulma. Tämän lisäksi jotkut käyttävät tasapainotetun mittariston kohdalla myös työntekijöiden tyytyväisyysnäkökulmaa ja ympäristö- ja yhteisönäkökulmaa. Taloudelliseen näkökulmaan kuuluu vahvuuksien käyttöaste, asiakkaan näkökulmaan kuuluu asiakkaan tyytyväisyyden lisääminen, sisäisten prosessien näkökulmaan kuuluu teknologian optimointi, oppimis- ja kasvunäkökulmaan kuuluu asiantuntijuuden kasvaminen ja parempi sopeutuminen, työntekijöiden tyytyväisyysnäkökulmaan kuuluu positiivinen yrityskulttuuri ja ympäristö -ja yhteisönäkökulmaan kuuluu paikallisten yritysten suosiminen ja tulevaisuuden työntekijöihin kohdistuva verkostoituminen. Tasapainotetun mittariston haasteena on, että keskeisiä suorituskykymittareita ei ole tarkasti määritelty ja että kriittiset onnistumistekijät eivät itsessään sovi hyvin tasapainotetun mittariston näkökulmien kanssa. (Parmenter 2010, 16-18).

Organisaatiolla kuuluu myös mittaamisen apuna olla kriittiset onnistumistekijät eli Critical success factors. Kriittiset onnistumistekijät ovat lista asioita ja näkökulmia organisaatiollisesta suorituskyvystä, jotka määrittelevät yrityksen nykyisen elinvoiman ja hyvinvoinnin.

Kriittisiä onnistumistekijöitä löytyy yleensä viidestä kahdeksaan jokaisesta organisaatiosta. (Parmenter 2010, 25).

Onnistumistekijät jaetaan taloudellisiin ja ei-taloudellisiin tekijöihin. Taloudelliset tekijät liittyvät esimerkiksi organisaation likviditeettiin eli maksuvalmiuteen, kannattavuuteen ja taloudelliseen kasvuun. Ei-taloudellisia tekijöitä ovat taas asiakastyytyväisyys, laatu ja toimitusaika. Onnistumistekijät muodostavat keskenään syy-seuraussuhteita eli ne jaetaan syytekijöiksi ja seuraustekijöiksi. Tämä ajatus perustuu siihen, että esimerkiksi henkilöstön osaaminen ja myyntimäärä voivat olla syynä asiakasuskollisuudelle. (Lönnqvist ym. 2006, 22.)

Mittarien luomisessa käytetään yleensä neljää eri vaihetta, jotka ovat mittariston suunnittelu, mittariston käyttöönotto, mittariston käyttö ja mittariston ylläpito. Suunnitteluvaiheessa valitaan yleensä mitä mitataan ja millaisia mittareita käytetään. Tämän jälkeen mittarit otetaan käyttöön kouluttamalla työntekijöitä niiden käytöstä. Käyttövaiheessa mittarit tarjoavat johtamiseen tukea ja apua organisaation kehittämiseen. Mittareiden ylläpito viimeiseinä on myös tärkeää, koska liiketoiminnan tavoitteet voivat muuttua ja tätä kautta mittarit voivat päivittämättöminä menettää merkityksensä. Päivityksessä mittareita voidaan myös lisätä tai poistaa tarpeen mukaan ja tämän jälkeen mittarien nelivaiheinen sykli alkaa alusta. (Lönnqvist ym. 2006, 12).

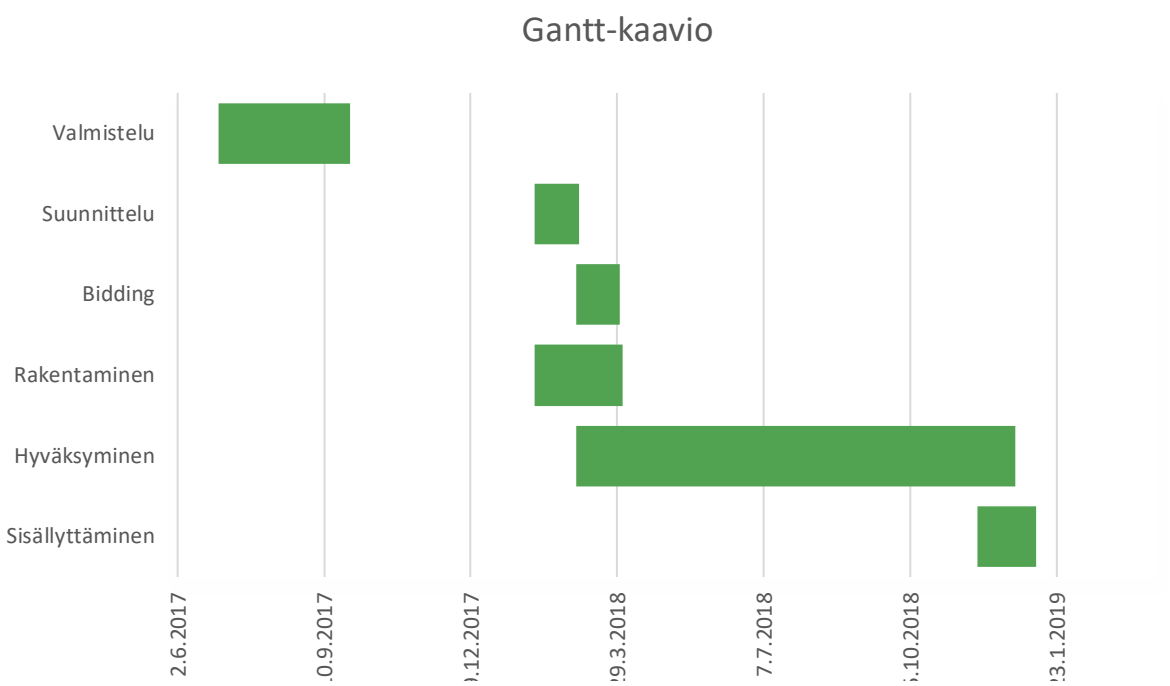
2.4.2 Ohjelmistokehityksen mittarit

Perinteisen ohjelmistokehityksen mittarit keskittyvät vesiputousmallista saatujen tietojen mittaamiseen. Näitä mittareita ovat esimerkiksi monimutkaiset tuotteen osat, ohjelmiston vaatimustenmäärittely, suunnittelukatselmoinnit ja sisäisissä virstanpylväissä pysyminen. (Leffingwell 2007, 312).

Vaatimustenmäärittelyn mittaamisessa voidaan tarkastella itse dokumentin laatua eli löytyykö dokumentista epäsuoria, valinnaisia tai ympäripyöreitä sanoja. Myös epämääräiset lauseet voivat muokata merkittävästi kehitystyötä, jos niitä ei ymmärretä asiakkaan ja toimittajan kanssa täysin samalla tavalla. Suunnittelukatselmoinnin mittarit toimivat usein katselmoinnista saadun palautteen kautta. Palaute voi olla yksinkertaisesti hyvien ja huonojen puolien järjestelmällistä kirjaamista tai esimerkiksi a/b-testausta, jossa kahta eri suunnittelua näytetään koehenkilöille ja kehitystyötä jatketaan enemmän ääniä saaneella suunnitelmalla. Ohjelmistokehityksen mittaamisen haasteena on tyypillisesti se, että se toteutetaan asiantuntijaorganisaatioissa, jolloin varsinaiset hyödynnettävät tulokset saadaan

vasta kehitystyön lopuksi. (Leffingwell 2007, 312). Tätä haastetta kuvaan tarkemmin luvun lopussa.

Ohjelmistokehityksessä yksi yleisimpiä aikataulun mittareita on Gantt-kaavio. Gantt-kaavio on visuaalinen esitystyökalu, joka esittää esimerkiksi projektin aktiviteetteja tai toimituksia tietyn aikataulun puitteissa. Gantt-kaavioita voi käyttää kaikenkokoisten projektien suunnitteluun ja ne antavat konkreettisen kuvan projektin aloituksesta ja lopetuksesta. Gantt-kaavio sisältää yleensä tärkeimmät toimitukset, tärkeimmät virstanpylväät, tehtävien aloitus -ja lopetuspäivät ja tehtävien jakamisen omiin ryhmiin tai tasoihin. Gantt-kaavioita löytyy räätälöitynä erilaisiin tarkoituksiin ja yleensä niitä käytetään infrastruktuurisuunnittelussa, tuotantosuunnittelussa ja tietotekniikassa. (Projectcubicle 2018). Gantt-kaavion esimerkki löytyy alta kuvasta kaksi.



Kuva 2. Gantt-kaavio (Ahonen 13.9.2015)

Gantt-kaavion etuja hyödyntävät eniten projektipäälliköt ja projektin sidosryhmät. Gantt-kaavio on helppo tehdä, lisää ymmärrystä kaikkien sidosryhmien kesken, tehostaa tiimin tuottavuutta, parantaa kommunikaatiota muiden sidosryhmien välillä, tukee hankintaprosessia näyttämällä toimituspäivät, helpottaa projektin tilanteen vertaamista alkuperäiseen aikatauluun ja helpottaa päätöksentekoa. Gantt-kaavion huonoja puolia on taas se, että monimutkaisissa projekteissa on hankalaa nähdä kaikki aktiviteetit, sen ylläpitäminen suurissa projekteissa vie paljon resursseja, on vaikeaa nähdä koko projektin tilanne yhdeltä

esitykseltä ja aktiviteettien muuttaminen toiseen aikatauluun voi olla hankalaa, jos se ei sovi alkuperäiseen suunnitelmaan. (Projectcubicle 2018)

Ohjelmistokehityksen mittarit ovat siis tyypillisesti asiantuntijaorganisaation suorituskyvyn mittareita. Yleisesti asiantuntijaorganisaatioiden mittaamishaasteet liittyvät työn tietointensiiviseen luonteeseen. Asiantuntijatyö on yleensä hyvin aineetonta ja tulosten todentaminen on vaikeaa. Tästä syystä työn suunnittelun laatua on vaikea määritellä ennen tehdyn työn valmistumista. Työprosessi on myös hankala mallintaa, koska se voi muuttua monen eri tekijän kautta aivan toisenlaiseksi. Lisäksi tuloksista voidaan usein tehdä vasta johtopäätöksiä niiden valmistumisen jälkeen, jolloin työn tuottavuudella ei ole enää suurta merkitystä. (Lönnqvist ym. 2006, 50-51).

Asiantuntijaorganisaation tärkeimpiä mittareita ovat tehokas tiedonkulku, henkilöstön osaamisen jatkuva kehittäminen ja osaamis- ja yhteistyöverkostojen luominen. Tehokkaan tiedonkulun takaamiseksi organisaatio voi mitata vuosittaisia rahallisia investointeja infrastruktuuriin ja tietokantoihin tallennettujen dokumenttien osuutta kaikista dokumenteista. Tiedonkulkua voidaan myös edistää panostamalla tutkimus- ja kehitystoimintaan ja mittaamalla eri tutkimuslaitosten välisten yhteistyöprojektien määrää oman yrityksen kanssa. Henkilöstön osaamista voi taas mitata vuosittaisilla koulutusmenoilla jokaista henkilöä kohden ja niin sanotulla rookie ratiolla eli alle kaksi vuotta organisaatiossa töissä olevien työntekijöiden määrällä. Osaamis- ja yhteistyöverkoston luomisessa tärkeitä mittareita ovat hyvän ilmapiirin mittaaminen kyselyiden avulla ja työntekijöiden ja organisaation tarpeiden yhteensopivuuden mittaaminen. Yhteensopivuuden mittaamisessa voidaan käyttää työtyytyväisyysindeksiä tai kehityskeskusteluista saatua palautetta. (Lönnqvist ym. 2006, 59-68).

2.4.3 Ketterän kehityksen mittarit

Ketterän kehityksen keskeisimmät mittarit ovat toimiva ohjelmisto ja sen toimivuus sille tarkoitetussa ympäristössä. Nämä kaksi asiaa tulisi aina määritellä empiirisesti eli havaintoihin pohjautuen jokaisen iteraation ja julkaisun yhteydessä. Kaikki muut mittarit ovat näistä johdannaisia eli niiden täytyy aina tukea näitä kahta vaatimusta. (Leffingwell 2007, 312).

Suorituskyvyn mittaamisessa voidaan ketterässä ohjelmistokehityksessä käyttää sekä perinteisiä että ketteryyteen erikoistuneita mittareita. Perinteiset mittarit voivat olla virheiden (defect) määrä ja yksikkötestatun ohjelmakoodin prosentuaalinen määrä ja ketteryyteen

erikoistuneita mittareita voivat olla valmistuneiden käyttäjätarinoiden määrä ja niiden esityskelpoisuus jokaisen iteraation lopuksi. Ketteryyteen erikoistuneita mittareita käytetään siis tiimin suorituskyvyn mittaamiseen projektitasolla. (Leffingwell 2007, 312-313).

Ketterät mittarit jaetaan yleensä kahteen kategoriaan, iteraatiomittareihin ja julkaisumittareihin. Iteraatiomittarit ovat jatkuvalla toistolla käytössä jokaisessa iteraatiossa ja ne tarjoavat nopeaa palautetta ja hienosäätöä toimintaan. Iteraatiomittareita ovat esimerkiksi toteutuneiden käyttäjätarinoiden määrä suhteessa kaikkiin suunniteltuihin käyttäjätarinoihin. Julkaisumittarit taas keskittyvät asiakkaalle tuotettuun arvoon hitaammalla aikavälillä. Julkaisumittarit voivat olla taas esimerkiksi asiakastyytyväisyyden seuraamista. (Leffingwell 2007, 312-313).

Vaikka nämä kaksi mittaria ovat yleisimpiä ketterälle tiimille, ne kuitenkin mittaavat ainoastaan tiimin suorituskykyä projektitasolla. Tämän takia sisäisten prosessin mittaamiselle tarvitaan suhteellisen ketteryyden mittaristo. Suhteellisen ketteryyden mittaristoon kuuluu kuusi mittaria, jotka on jaettu yhä alemman tason osioihin. Mittarit ovat tuotteen omistajuus/hallitsemiskapasiteetti, julkaisujen suunnittelun ja seurannan kyvykkyys, iteraatioiden suunnittelu ja seuranta, tiimin tuottavuus, testauskäytännöt ja kehitystyökäytännöt/infrastruktuuri. Näiden mittareiden alta löytyy väittämiä, jotka arvostellaan nollasta viiteen niin, että viisi vastaa kaikista ketterintä tulosta. Kaikki väittämät ovat selkeässä muodossa ja niitä voivat olla esimerkiksi kehitystiimin osallistuminen Scrumin eri tapahtumiin ja ylityötuntien yleisyys. Tämän jälkeen vastauksista voidaan tehdä ”radar”-kaavio, joka on moniulotteinen kaavio, josta nähdään eri mittareiden prosentuaalisen onnistumisen taso. Tämän mittariston avulla saadaan tehokkaasti selvitettyä tiimin ketteryys ja sen lisäksi ohjeet, mitä pitäisi parantaa työn ketteryyden lisäämiseksi. (Leffingwell 2007, 312-317)

Itsearviointi on ketterissä menetelmissä, varsinkin Scrumissa, elintärkeä mittaamistapa. Organisaation johto voi olla tästä mittaamistavasta skeptinen, koska jos kehitystiimi saa itse valvoa omia prosessejaan ja tuottamiaan laatuja, kehitystiimi pystyy huijaamaan helposti mittauksen tuloksia. Ketterissä menetelmissä on kuitenkin kaksi asiaa, jonka takia tätä ei yleensä tapahdu. Ensimmäinen syy on se, että kehitystiimi ei voi valehdella tuottaneensa loistavaa laatua, jos sidosryhmille ei ole tuotettu todellista arvoa. Koska ketteryyden yksi päämittareista on läpinäkyvästi toimiva ohjelma, kaikki sen tarkastelijat kyllä huomaavat esimerkiksi sprintin onnistumisen ja epäonnistumisen. Toinen syy on se, että ketterään ajattelutapaan kuuluu kehitystiimin vapauttaminen kaikista epäoleellisista rajoituksista. Kun tämä toteutetaan, kehitystiimin tuottavuus paranee ja kehitystiimin sisällä olevat työntekijät pitävät myös muita tiimin jäseniä vastuullisena toiminnastaan. Tätä kutsutaan

”taikapilleriksi”, joka vapauttaa kehitystiimin saavuttamaan parhaat tulokset. (Leffingwell 2007, 318).

Kun tiimitasoiset ja projektitasoiset mittarit on määritelty, tarvitaan organisaation ketteryydelle myös oma mittaristo. Tähän mittaamisen yksi parhaista työkaluista on tasapainotettu mittaristo. Kuvassa kolme voidaan nähdä esimerkkimittaristo liiketoiminnan suorituskyvyn mittaamiseen.

Tehokkuus Tutkimus- ja kehitystyöorganisaation mittaaminen tuottavuuden, nykyisten toimitusten ja kustannuskohteiden kautta Esimerkkimittareita: • Organisaatiollinen vakaus • Tiimin nopeus vs. kapasiteetti	Arvontuottaminen Asiakkaalle toimitetutun ohjelman arvontuoton mittaaminen mittausaikana (12 kuukautta) Esimerkkimittareita: • Versioiden määärä • Arkkitehtuurinen refaktoironti • Arvoa tuottavien toiminnallisuuspisteiden toimittaminen
Laatu Tuotteen laadun mittaaminen asiakkaan ympäristön mukaan Esimerkkimittareita: • Virheet (Defect) ja normalisoidut virheet • Tukipyynnöt ja normalisoidut tukipyynnöt • Tukityytyväisyys • tuotetyytyväisyys	Ketteryys Organisaation kyvyn mittaaminen parantaa ja saavuttaa tulevaisuuden suorituskykytavoitteet Ketteryysprosessin itsearviointi: • Tuotteen omistajuus • Julkaisujen suunnittelu ja seuranta • Iteraatioiden suunnittelu ja seuranta • Tiimityöskentely • Testauskäytännöt • Kehitystyön käytännöt

Kuva 3. Tasapainotettu mittaristo liiketoiminnan suorituskyvyn mittaamiseen (mukaillen Leffingwell. 2007, 320)

Kuvan kolme tasapainotettu mittaristo on jaettu neljään eri kohtaan, jotka ovat tehokkuus, laatu, arvontuottaminen ja ketteryys. Tehokkuuden tavoitteena on mitata tiimien ja organisaatioiden tehokkuutta saavuttaa omat tavoitteensa. Tehokkuusmittareita ovat esimerkiksi organisaatiollinen vakaus ja tiimin käyttöasteen hyödyntäminen arvon tuottoon omaan kapasiteettiinsa nähden. Tehokkuusmittarit voidaan usein suoraan sisällyttää projektitasoisiin mittareihin. (Leffingwell 2007, 319-320).

Laatumittarit keskittyvät asiakkaan ympäristöstä saatuun palautteeseen tuotteen laadusta. Laatumittareita käytettäessä käyttäjien määrä pitää normalisoida virheiden ja tukipyyntöjen yhteydessä, koska muuten ohjelma, jossa on paljon virheitä ja vähän käyttäjiä, vaikuttaa luotettavammalta kuin ohjelma, jossa on vähän virheitä ja paljon käyttäjiä. Arvontuottamismittarit on ketteriin menetelmiin erikoistunut mittari, joka keskittyy organisaation kykyyn toimittaa tuotettaan nopeammin ja enemmän. Arvontuottamismittareita käytetään yleensä vuositasolla. Toiminnallisuuspisteillä pyritään muuttamaan määräksi monimutkaisia, mutta tärkeitä mittareita. Niiden avulla pyritään selvittämään asiakkaan oikea arvoa tuottava toimitus tietyllä aikavälillä. (Leffingwell 2007, 320-321).

Ketteryysmittareilla keskitytään organisaation kykyyn tuottaa jatkuvasti parempaa suorituskykyä ja tarjota tulevaisuudessa yhä parempaa arvoa omalle organisaatiolle ja asiakkailleen. Ketterissä mittareissa tiimin itsearviointi kvartaaleittain on paras tapa saavuttaa raportointivaatimukset ja auttaa kehitystiimiä tähtäämään jatkuvaan itsensä kehittämiseen. (Leffingwell 2007, 321-322).

3 Haastattelujen läpikäynti

Opinnäytetyön tutkimusmenetelmänä käytetään laadullista eli kvalitatiivista tutkimusta. Tutkimuksen tavoitteena on ymmärtää osaston nykyinen sovelluskehitys ja sen ongelmat. Tutkimukseen osallistuu kuusi osaston asiantuntijaa, joita haastattelen kahdessa eri osiossa. Haastatteluissa keskitytään sovelluskehityksen nykytilaan ja ongelmiin, jotka johtuvat ketteryyden puutteesta tai sen puutteellisesta käyttöönotosta. Molemmat haastatteluosuudet toteutettiin kasvokkain kahdenkeskeisesti ja nauhoitettiin. Nauhoitus on tämän jälkeen litteroitu ja muokattu johdonmukaiseen muotoon karsien ketteryyteen liittymättömät keskustelut pois. Opinnäytetyön haastatteluosan lopputuloksena syntyy osaston sovelluskehityksen nykytila- ja ongelma-analyysi, joiden kautta ketteryyttä lisääviä keinoja voidaan lähteä havainnoimaan.

Ensimmäisessä osiossa eli nykytilahaastattelussa haastattelen kolmea eri asiantuntijaa osaston nykytilasta. Nykytilahaastattelun tavoitteena on avoimien kysymysten ja niistä seuraavien keskustelujen pohjalta ymmärtää sekä sovelluskehityksen nykytila että siihen liittyvät taustatekijät. Haastateltavat ovat kokeneita, yli 10 vuoden työkokemuksen omaavia asiantuntijoita, joilla on hyvä ymmärrys sovelluskehityksestä ja osaston nykytilasta. Haastatteluissa otetaan huomioon haastateltavien pitkä työura osastolla ja heidän oman työkuvansa subjektiivinen näkemys eri osa-alueisiin.

Toisessa osiossa eli ketteryysasiantuntijoiden haastattelussa haastattelen kolmea eri asiantuntijaa osaston ongelmista. Haastattelun tavoitteena on avoimien kysymysten ja niistä seuraavien keskustelujen pohjalta ymmärtää sovelluskehityksen haasteita. Haastateltavat ovat myös pitkän uran tehneitä asiantuntijoita, jotka ovat hankkineet Agile-sertifikaatin osaston koulutuksen kautta. Tämän perusteella he myös kertovat omia näkemyksiään osaston nykytilan muuttamisesta ketterämpään suuntaan muutaman kysymyksen yhteydessä. Haastatteluissa painotan kysymyksiäni nykytilahaastatteluun, mutta Agile-sertifikaatin takia haastattelusta voi löytyä uusia näkökulmia nykytilahaastattelun tuloksiin. Ketteryysasiantuntijoiden haastattelussa otan myös huomioon haastateltavien pitkän uran osastolla ja heidän oman työkuvansa subjektiivisen näkemyksen.

3.1 Nykytilahaastattelu

Nykytilahaastattelu koostui 13 kysymyksestä, joista kolme ensimmäistä kysymystä liittyivät pelkästään haastateltavien taustan selvittämiseen. Haastattelut toteutettiin yksilöllisesti ja haastateltavilla ei ole tietoa muiden haastateltavien vastauksista. Haastattelun pohja on

löydettävissä liitteestä 1. Haastattelun aluksi halusin ymmärtää, mikä oli jokaisen haastateltavan työtehtävä. Työtehtävän merkitys haastattelulle oli siksi tärkeä, että työtehtävien kautta sain selville osaston sovelluskehityksen roolijakoa ja pystyin siten arvioimaan jokaisen haastateltavan näkökulmia myös heidän omaan rooliinsa nähden.

Nykytilahaastatteluun osallistuvat asiantuntijat olivat työasemasovellusarkkitehti, testauspäällikkö ja projektipäällikkö. Arkkitehdit jaetaan osastolla yleensä työasemasovellusarkkitehteihin ja selainkäyttösovellusarkkitehteihin. Työasemasovellusarkkitehdin työtehtäviin kuuluu kokonaisuuden hallitseminen, kehittäjien ja testaajien tukeminen, vaatimustenmäärittely, use case eli käyttötapadokumenttien luominen, taskien eli työtehtävien tekeminen ja useiden dokumenttien tarkistaminen. Arkkitehdin mielestä häneltä oletetaan myös edellisten tehtävien lisäksi työasemasovellukseen kuulumattomien asioiden ymmärtäminen ja projektien aikataulujen selventäminen. Tämä ei työasemasovellusarkkitehdin mielestä ole arkkitehdin työtä ja hän ei oikein pidä sen siirtämisestä arkkitehdeille. Työasemasovellusarkkitehti myöntää kuitenkin, että hän yleensä tietää nämä asiat parhaiten muiden työtehtäviensä kautta.

Testauspäällikkö kertoo ohjaavansa testaamista, raportoi testauksen tilannetta eteenpäin ja keskustelee testauksesta asiakkaan kanssa. Testaukseen kuuluu järjestelmän komponenttien ja niiden integroinnin testaus ja bugien eli virheellisen ohjelmakoodin ja testitapausten hallinta. Projektipäällikön työtehtäviin kuuluu asiakkaan kanssa kommunikointi, muutoshallinta, kuukausiraportointi, laskutus, talouden seuranta, työmäärien toteutumisen seuranta, sisäisen työskentelyn ohjaus ja projektin etenemisen koordinointi. Projektipäällikön mielestä sisäisen työskentelyn ohjausta on jouduttu usein siirtämään arkkitehdeille ja testauspäälliköille, koska hänen aikansa ja osaamisensa eivät oikein riitä. Kaikki kolme haastateltavaa ovatkin sitä mieltä, että jatkuva kiire vaikeuttaa heidän työntekoaan.

Toisena kysymyksenä haastateltavat kertoivat omista työkaluistaan. Haastateltavien käyttämiä työkaluja olivat Microsoftin eri sovellukset kuten Outlook-sähköpostisovellus, Excel ja Team Foundation Server. Lisäksi Jira ja MPR-kaavio eli Exceliin tehtävä kaavio projektin talouden tilasta ovat testauspäälliköllä ja projektipäälliköllä merkittävässä käytössä. Microsoftin Team Foundation Server-ohjelmistoa käytetään sovelluskehityksessä erilaisten ohjelmavirheiden ylläpitämiseen, toteutustehtävien ylläpitämiseen ja yleiseen raportointiin esimerkiksi ratkaisukuvauksista ja järjestelmäkuvauksista. Jiraa eli tehtävienhallintaohjelmistoa kaikki kolme asiantuntijaa käyttävät asiakkaiden pyynnöstä myös samoihin käyttötarkoituksiin kuin Team Foundation Serveriä. Jira-ohjelmaan viedään yleensä ohjelmistovirheet ja muut tärkeät ilmoitusasiat, jotka kuuluvat asiakkaalle.

Kolmantena kysymyksenä halusin tietää, mitä haastateltaville merkitsee ketteryys. Teoriaosuudessa käsiteltiin Agile Alliencen lähdettä, jonka mukaan ketterät menetelmät nähdään ennemmin työkaluna kuin tapana ajatella. Halusin siis haastattelun perusteella selvittää, onko ketteryys haastateltavien mielestä enemmän työkalu vai ajattelutapa. Haastateltavien mielestä ketteryys on lyhyissä jaksoissa saatuja suunniteltuja toteutuksia, vesiputousmallin vastakohta ja suunnan nopeata vaihtamista. Yksi haastateltavista piti ketteryyttä muotiterminä, jota kukaan ei oikeasti osaa toteuttaa.

Neljäntenä halusin ymmärtää projektien sovelluskehityksen elinkaaren, joten pyysin haastateltavia kuvaamaan sen eri vaiheet lyhyesti. Haastateltavien mukaan projekti alkaa yleensä siitä, kun asiakas ilmaisee mielenkiinnon jostain asiasta. Tämän jälkeen myyntiosasto pyrkii keskustelemaan ratkaisuista ja eri asiantuntijat voivat tulla myynnin tueksi asiakasneuvotteluihin. Tämän jälkeen sopimus allekirjoitetaan, mikäli asiakas on hyväksynyt ratkaisun. Osasto ei rakenna alusta alkaen asiakkaalle tuotettaan, vaan osasto käyttää omaa tuotettaan, jota räätälöidään asiakkaan tarpeiden mukaan. Kun sopimus on allekirjoitettu, vaatimustenmäärittely on yleensä jo tehty tai sitä jatketaan pari kuukautta sopimuksen hyväksymisen jälkeen. Sopimukseen on myös kirjattu työmääräarviot valmiiksi.

Vaatimustenmäärittely tapahtuu oman tuotteen tietojen ja käyttötapadokumenttien kautta. Käyttötapadokumenttien avulla pyritään ilmaisemaan asiakkaalle, mitä tarkoittaa ”valmis” tuote. Käyttötapadokumentin kohdista luodaan asiakkaan vaatimusten mukaan työtehtävät, jotka siirretään ylläpidettäväksi Team Foundation Server-ohjelmaan. Oman tuotteen versio pitäisi olla aina jäädytettynä eli täysin testattuna ja toimivana, kun sen päälle aletaan rakentamaan asiakaskerrosta. Näin ei kuitenkaan haastattelijoiden mielestä käy, vaan omaa tuotetta ja asiakaskerrosta rakennetaan usein samanaikaisesti.

Kun varsinainen projekti alkaa, projekti toteutetaan iteratiivisesti. Kun pyysin iteratiivisuuteen tarkennusta, haastateltavat kertoivat, että iteratiivisessa projektissa päätetään iteraatioissa tehdystä toteutuksesta. Iteraatioissa tietyt tehtävät tehdään joko loppuun asti ja testataan kokonaan tai yhdessä iteraatiossa tehdään tietyt tehtävät ja ne testataan seuraavassa iteraatiossa. Iteraatiot voivat kestää kuukaudesta muutamaan kuukauteen. Käytännössä iteraatioita ei kuitenkaan suunnitella alkua pidemmälle, koska niitä ei hallita ollenkaan.

Kun toteutusvaiheessa ohjelmistokehittäjät ovat saaneet jonkun kokonaisuuden valmiiksi, se siirretään testausympäristöön testattavaksi. Jos testeissä ilmenee ohjelmavirheitä, niin testaaja selvittää ohjelmavirheen taustan ohjelmistokehittäjän tai arkkitehdin kanssa. Jos he ovat samaa mieltä ohjelmavirheiden olemassaolosta, se kirjataan Team Foundation

Server-ohjelmistoon, jossa ohjelmavirheitä ylläpidetään projektin loppuun asti. Tämän jälkeen projekti etenee osaston iteraatiomallin mukaan loppuun asti, mutta sovelluskehityksen välissä voi tulla katselmointeja. Kun toteutus on saatu valmiiksi, tuote siirretään asiakashyväksymistestaukseen, missä asiakas usein näkee ja pääsee testaamaan tuotetta ensimmäistä kertaa. Jos asiakas on tuotteeseen tyytyväinen, niin se siirtyy pilottiin ja sieltä lopulta tuotantoon käyttöön.

Projekteissa pidetään lopuksi myös niin sanottu opit-kokous, jossa käydään läpi onnistumisia ja parannettavia asioita projektipäällikön johdolla. Yrityksellä ei ole näihin kokouksiin mitään varsinaista ohjetta ja haastateltavien mukaan joskus näistä opitaan jotain, mutta usein kehitysideat jäävät ideatasolle. Osaston sovelluskehitys on siis teoriaosuuden (Resnick ym. 2011) mukaan vesiputousmalliin pohjautuva sen vakauden, suunnittelun ja dokumentoinnin perusteella. Sovelluskehitys etenee myös järjestyksessä vaatimustenmäärittelystä ylläpitoon teoriaosuuden kaavion (Royce 1970) mukaan.

Viidentenä halusin ymmärtää osaston työn organisoinnista. Haastateltavien mukaan tiimijohtajat hoitavat osaston työntekijöiden organisoinnin. Tiimijohtajat pyrkivät siihen, että jokaisella projektilla olisi työmääräarvioiden mukainen määrä työntekijöitä koko projektin ajan. Projektipäälliköt pyytävät Excel-arvioiden perusteella työntekijöitä ja työntekijöiden määrään ja työn kestoon vaikuttavat tilannetyötekijät eli kompetenssi. Kompetenssilla tarkoitetaan noin 20 erilaista kohtaa, joiden avulla voidaan yksittäiselle työntekijälle antaa joko plussia tai miinuksia riippuen hänen kokemuksestaan. Työmääräarviot luodaan ennen sopimuksen syntymistä arvioimalla työmäärä kokonaan keskivertotyöntekijän näkökulmasta, jonka jälkeen työmäärään lisätään vielä jonkin verran henkilötyöpäiviä projektin onnistumisen takaamiseksi. Haastateltavien mukaan kompetenssien perusteella työmäärä voi parhaimmillaan viisinkertaistua tai pudota kolmasosaan. Haastateltavat kertoivat myös, että jos projektin aikana huomataan, että projektiin on tulossa enemmän alhaisemman kompetenssin asiantuntijoita kuin aluksi oli suunniteltu, niitä ei muuteta tarpeeksi.

Projektin alussa annetut resurssit eivät pidä haastateltavien mukaan täysin paikkaansa, jolloin esimerkiksi arkkitehdit saattavat työskennellä täyspäiväisesti projektissa, johon heidän piti alun perin osallistua muutama päivä kuukaudessa. Työn organisointi kehitystiimin sisällä taas jää usein arkkitehtien ja testauspäälliköiden tehtäväksi, koska projektipäälliköllä ei usein ole tarvittavaa teknistä ymmärrystä tai aikaa työn ohjaukseen ja seurantaan. Haastateltavat kertoivat myös, että työn seurantaa ei käytännössä tapahdu kuin ainoastaan pitkissä viivästyksissä. Organisoinnin puute oli kaikkien haastateltavien mielestä ongelmallista ja suuri syy projektien viivästymiseen.

Kuudentena halusin ymmärtää osaston roolituksen. Roolituksen selvittämisessä halusin ymmärtää, löytyykö osastolta mitään ketterien menetelmien rooleja, kuten Scrummasteria tai tuoteomistajaa. Lisäksi halusin tietää, ovatko nykyiset roolit läpinäkyviä eli tietävätkö kaikki asiantuntijat oman ja muiden roolin työnjaon. Haastattelijat kertoivat, että roolitukset ja niille kuuluvat tehtävät on määritelty osaston omassa ylläpidettävässä dokumentissa.

Roolituksessa ohjelmistokehittäjät on jaettu työasemasovelluskehittäjiin ja selainkäyttösovelluskehittäjiin. Roolitus tapahtuu usein suurien asiakkuuksien mukaan, joissa monet työntekijät työskentelevät lähes pääsääntöisesti. Osa työntekijöistä kuitenkin liikkuu eri asiakkuuksien välillä tarpeen mukaan oman erikoisosaamisensa takia. Arkkitehdit jakautuvat myös työasemasovelluksen ja selainkäyttösovelluksen mukaan, mutta myös osa arkkitehdeistä tekee pelkästään samoja tehtäviä kuin ohjelmistokehittäjät.

Haastateltavat ilmaisivat, että projektipäällikön rooli on sekoittunut arkkitehtien ja testauspäälliköiden kesken johtuen suuresta hallinnollisesta määrästä ja teknisen osaamisen puutteesta. Arkkitehti ja testauspäällikkö ilmaisivat, että joissain projekteissa heillä menee suunnattoman paljon aikaa hallinnollisiin tehtäviin, joihin he eivät olleet varautuneet ennen projektin alkamista. Projektipäällikkö myös ilmaisi, että vaikka hän pystyy roolien sekoittumisen kautta keskittymään hallinnolliseen työhön enemmän, häneltä silti jää kokonaisuuden hallitseminen erittäin minimaaliseksi. Kun pyysin nähdä haastattelujen yhteydessä roolituksen dokumenttia, niin dokumentissa oli määritelty jokaisen roolin työtehtävät ja vaatimukset. Dokumenttia ei kuitenkaan ollut päivitetty kahteen vuoteen ja sieltä puuttui kokonaan ohjelmistokehittäjien roolikuvaus. Dokumenttien päivittämisen puute voikin olla yksi syy sille, miksi asiantuntijat kokevat oman roolituksensa menneen päällekkäin muiden asiantuntijoiden työtehtävien kanssa.

Seitsemäntenä kysymyksenä halusin ymmärtää lisää osaston iteraatioista, inkrementeistä ja mahdollisista sprinteistä. Teoriaosuuden mukaan iteraatiot ja inkrementit voivat kuulua sekä vesiputousmalliin että ketterän kehittämisen malliin. Sprintit taas vaativat teoriaosuuden mukaan täysin ketterää sovelluskehitystä. Haastateltavien mukaan iteraatioita ja inkrementtejä käytetään, mutta sprinttejä ei. Iteraatioita luodaan aluksi hyvin ja niille määritellään tehtävät, mutta kukaan ei lopulta seuraa niitä. Inkrementit taas toimivat haastateltavien mielestä hyvin. Kun aina tietty toteutustyö on saatu valmiiksi, se yhdistetään kehityskerrokseen, josta luodaan uusi versio testausympäristöön ja tarvittaessa asiakkaan ympäristöön. Tässä tapauksessa versiot eli inkrementit siis luodaan aina edellisen inkrementin päälle.

Kahdeksantena kysymyksenä halusin selvittää, miten osasto päättää projekteissa seuraavan kehitettävän asian ja miten osasto ylipäätään hallitsee kehitystyötään. Ketterien menetelmien, varsinkin Scrumin, mukaan yrityksillä on yleensä käytössä tuotteen kehitysjono, jossa hallitaan ja josta valitaan seuraavaksi kehitettävät osiot. Haastateltavat kertoivat, että osaston omalla tuotteella on tavallaan kehitysjono. Kehitysjonon kohtia muokataan usein asiakkaan tarpeiden mukaan ja omaa tuotetta harvoin ylläpidetään riittävällä tasolla sen kehitysjonon mukaan. Haastateltavilla ei tuntunut olevan muutenkaan hyvää ymmärrystä, miten oman tuotteen kehitystyö toimii ja mitä muutoksia siihen on lähiaikoina tehty.

Projekteissa seuraava kehitettävä asia määräytyy asiakkaan mukaan, kriittisen polun työtehtävien mukaan, seuraavan helpoimman työtehtävän mukaan ja jo lähes valmiiden osuuksien puuttuvien osien mukaan. Haastateltavien mukaan myös testausympäristöstä löytyvät ohjelmavirheet ovat ikään kuin tuotteen kehitysjonon kohtia, joilla todetaan yksi osuus valmiiksi. Tämän lisäksi jokaiseen iteraatioon sisällytetään kaikki käyttötapadokumentin kohdat ja ne viedään työtehtävinä sen jälkeen Team Foundation Server-ohjelmaan. Seuraavaksi kehitettävät asiat ovat siis jo luotu koko projektin ajalle valmiiksi iteraatioiksi ja niitä muokataan ja lisätään järjestelmän kehityksen kautta.

Yhdeksäntenä kysymyksenä halusin ymmärtää osaston yhteistyötä asiantuntijoiden välillä. Halusin selvittää, luotetaanko työntekijöiden ammattitaitoon ja onko eri työntekijöiden välillä vastakkainasettelua. Haastateltavat sanoivat, että henkilökemiat toimivat osastolla hyvin. Osaston työntekijöiden vaihtuvuus on todella pientä ja kaikki tulevat kaikkien kanssa toimeen. Testauspäällikkö kertoi, että hän kokee usein muiden asiantuntijoiden olettavan, että testaajat löytävän ohjelmavirheet järjestelmästä, jolloin kehittäjien ei tarvitse tehdä niin kattavaa yksikkötestausta. Kiire ja stressi aiheuttavat myös joskus eripuraa työntekijöiden välillä, mutta asiat on saatu aina lopulta selvitettyä.

Kymmenentenä kysymyksenä halusin ymmärtää mittareiden käytön osastolla. Testauspäällikkö sanoi käyttävänsä Team Foundation Serverin valmiita työkaluja, joita esimerkiksi olivat testiajojen määrä, ohjelmavirheiden määrä ja ohjelmavirheiden verifioinnit eli niiden korjausten hyväksymiset. Projektipäällikkö taas sanoi projektin etenemisen ja työtuntien olevan hänen päämittareitaan. Arkkitehdin mielestä mittareilla yleensä saadaan tieto ainoastaan siitä mitä mitataan ja että hän osallistuu harvoin mittarien hallitsemiseen. Osaston mittaaminen on siis hyvin pientä ja perustuu täysin tehdyn ja tekemättömän työn mittaamisen samalla tavalla kuten teoriaosuudessa (Lönnqvist ym. 2006) käyty perinteinen ohjelmistokehityksen mittaaminen.

Yhdentenätoista kysymyksenä halusin ymmärtää projektin ketteryydenmäärityksen. Haastateltavien mukaan asiakas päättää pitkälti projektin ketteryyden. Jos vaatimukset, hinta ja aikataulu on jo heti alussa lyöty lukkoon, on niitä vaikeaa toteuttaa täysin ketterästi. Haastateltavien mielestä ketteryyden määrittelee myös osaston johtajat, jotka hallitsevat projektisalkkua ja projektimenetelmiä. Projektipäällikkö voi joskus myös päättää projektin ketteryydestä.

Kahdentenatoista kysymyksenä halusin selvittää osaston palaverikäytännön. Haastateltavien mukaan palavereja järjestetään noin kerran viikossa koko tiimille, ydintiimi pitää viikkopalavereita kerran viikossa ja eri komponenttien osa-alueille pidetään omia seurantapalavereja kahden viikon välein. Myös isommille muutoshankkeille pidetään omia seurantapalavereja. Ydintiimiin kuuluu projektipäällikkö, testauspäällikkö, arkkitehdit ja versionrakentaja. Projektipäällikön mielestä projekteissa on aina haasteena löytää optimaalinen määrä palavereja ja kaikkien mielipiteitä palavereiden määrästä on vaikea tyydyttää. Arkkitehdin mielestä palavereita tarvitaan yleensä silloin, kun hommat eivät etene.

Viimeiseksi halusin ymmärtää osaston kommunikoinnista. Haastateltavat kertoivat, että kommunikointia hoidetaan pääsääntöisesti palaverien, erilaisten työtilojen, sähköpostien ja käytäväkeskustelujen kautta. Kommunikointia vaikeuttaa jonkin verran se, että osaston projekteihin osallistuu koko Suomen halki ihmisiä ja osasto käyttää paljon ulkoista osaamiskeskusta ulkomailla.

Tiivistettynä haastateltavat olivat suurimmasta osasta asioita samaa mieltä. Haastateltavien mielestä ketteryys on nopeata suunnan vaihtamista ja vesiputousmallin vastakohtaa, mutta samalla jotain, mikä ei käytännön tasolla toimi. Haastateltavien antamien tietojen perusteella osaston sovelluskehitys on vesiputousmaista, jossa on myös iteraatiot ja inkrementit käytössä. Projektit alkavat yleensä vaatimustenmäärittelyllä, jonka jälkeen siirretään vaatimukset iteraatioihin ja aletaan rakentaa asiakaskerrosta. Tämän jälkeen kehitystyö aloitetaan ja se testataan testausympäristössä jokaisessa versiossa. Kun toteutus on valmis, tuote viedään asiakashyväksymistesteihin, pilotointivaiheeseen ja lopulta tuotantoon. Projektin työ organisoidaan työmääräarvioilla, joihin vaikuttaa mm. tilannetyötekijät. Projektin kehitystiimin työn organisointi tapahtuu usein arkkitehtien ja testauspäällikön kautta. Roolitus jakautuu asiantuntijoiden kanssa muuten selvästi, mutta arkkitehdin, testauspäällikön ja projektipäällikön rooleissa on välillä päällekkäisyyksiä. Tuotteen kehitysjono löytyy osaston omasta tuotteesta ja Team Foundation Server-ohjelmassa ylläpidettävät tehtävät ovat projekteissa kehitysjonon kohtia. Näitä ei kuitenkaan ylläpidetä ja niille tarvittaisiin joku vastuuhenkilö. Osaston henkilökemia toimii hyvin ja mittareina käytetään

työn etenemistä ja ohjelmien antamia valmiita mittareita. Asiakas, osaston johto ja projektipäällikkö määrittelevät projektien ketteryyden ja palavereja järjestetään projekteissa erikseen kaikille työntekijöille, ydintiimille, osa-alueille ja muutoshankkeille. Kommunikointi toimii myös pääsääntöisesti hyvin, mutta työntekijöiden hajanainen sijainti vaikeuttaa tiedon kulkua.

3.2 Ketteryysasiantuntijoiden haastattelu

Ketteryysasiantuntijoiden haastattelu koostui 15 kysymyksestä, joista kolme ensimmäistä kysymystä liittyivät pelkästään haastateltavien taustan selvittämiseen. Haastattelut toteutettiin yksilöllisesti, mutta haastateltavilla on yleinen ymmärrys muiden haastateltavien vastauksista, koska käytän nykytilahaastattelun vastauksia pohjana ketteryysasiantuntijoiden haastattelukysymyksiin. Haastattelun pohja on löydettävissä liitteestä 2.

Ketteryysasiantuntijahaastatteluun osallistuivat kaksi arkkitehtia ja yksi ohjelmistokehittäjä. Yksi arkkitehti oli selainkäyttösovellusarkkitehti ja toinen arkkitehti osallistui enemmän työasemasovelluksen kehittämiseen. Ohjelmistokehittäjä osallistuu selainkäyttösovelluksen kehittämiseen. Selainkäyttösovellusarkkitehdin työtehtävät eivät eronneet oikein työasemasovellusarkkitehdin työstä, mutta toisella työasemasovellusarkkitehdilla ja kehittäjällä olivat hyvin samanlaiset työtehtävät. Ohjelmistokehittäjä ja ohjelmointia tekevä työasemasovellusarkkitehti kertoivat, että heidän työtehtäviinsä kuuluu asiakkuuksiin liittyvä vaatimustenmäärittely, toteutus eli varsinainen ohjelmointiosuus, asiakastapaamiset, testauksen ja muiden kehittäjien tukeminen, käyttöönoton tukeminen ja tuotanto-ongelmien ja muutoshankkeiden ratkaiseminen.

Kun haastateltavat olivat kertoneet työtehtävistään, halusin ymmärtää heidän käyttämiään työkaluja. Sekä ohjelmistokehittäjä että työasemasovellusarkkitehti käyttävät Microsoftin Visual Studiota, jossa he ylläpitävät ohjelmakoodia. Lisäksi SQL-Server, sähköposti ja Team Foundation Server ovat tärkeitä päivittäisiä työkaluja.

Kolmas ja viimeinen taustakysymys liittyi ketteryyden määrittelemiseen. Haastateltavien mielestä ketteryys on tapa ajatella asioita ja olla avoin muutokselle, koska muutoksia tapahtuu joka tapauksessa. Ketteryys oli myös haastateltavien mielestä sitä, että pystytään keskittymään tuottavaan työhön hallinnollisen työn sijasta.

Neljäs kysymys liittyi projektin elinkaaren ongelmiin. Tämän kysymyksen aikana kävin läpi eri osa-alueita sovelluskehityksestä nykytilahaastattelun perusteella. Selainkäyttösovellusarkkitehdin mielestä osasto yleisesti suunnittelee projektin alussa liian tarkasti ja liian

pitkälle aikavälille. Kun asiakas esittää myyntineuvotteluissa ”pykälälistan” vaatimuksia, niin osasto tarjoaa tähän suoraan ratkaisun eikä mieti varsinaisesti asiakkaan oikeaa tarvetta.

Elinkaaren suurin ongelma on kaikkien haastateltavien mielestä muutoshallinta. Kun projektia toteutetaan vesiputousmallisesti, se ei ole kovin avoin muutoksille. Osaston alkukartoitus projekteissa toimii haastateltavien mielestä erittäin hyvin, mutta muutoksien sisällyttäminen tuottaa ongelmia. Muutoksissa ei huomata niiden aiheuttamia sivuvaikutuksia, kuten lisätyön määrää ja haasteellista teknistä toteutusta, jolloin aikataulua pitää kiristää tai viivästyttää projektin loppua.

Haastateltavien mukaan asiakkaalla on myös usein hankaluuksia ymmärtää ratkaisukuvausta ja asiakas ei lue kuin pintapuolisesti kaikki dokumentit, missä osasto kertoo mitä he tarjoavat ja sisällyttävät projektiin. Haastateltavien mukaan käyttötapadokumenttien ongelmana on usein se, että niistä puuttuu kokonaan järjestelmien välinen vuorovaikutus. Haastateltavien mielestä myös osaston oma formaatti on liian monimutkainen asiakkaalle. Tästä syystä ”valmiin” tuotteen määritelmä projekteissa on todella hankalaa.

Osastolla oman tuotteen kehitys ja asiakaskerroksen samanaikainen kehitys on myös suuri ongelma. Kun omaa tuotetta ei olla vielä toteutettu ja testattu täysin, ei voida olla varmoja sen yhteensopivuudesta asiakaskerroksen kanssa. Usein oman tuotteen uusinta versiota odotetaan melkein projektin loppuun asti ja joskus asiakaskerroksen kehitystyö keskeytyy siihen asti, kunnes oma tuote saadaan käyttöön. Lisäksi asiakaskerroksen joitakin uusia ominaisuuksia halutaan lisätä omaan tuotteeseen heti, jolloin oman tuotteen julkaistun version aikataulu automaattisesti viivästyy. Tämä taas lisää projektin aikataulua ja hankaloittaa muiden projektien resursointia entisestään, kun joudutaan tekemään kompromisseja. Projektien lopussa olevien opit-keskustelujen ongelmana taas on se, että niissä käydään hyvää keskustelua parannettavista ideoista, mutta niitä otetaan harvoin käyttöön.

Viides kysymys liittyi työn organisoinnin ongelmiin. Organisointiin liittyvät ongelmat keskittyvät haastateltavien mukaan työmääräarvioihin, tilannetyötekijöihin ja hajanaiseen resurssointiin. Työmääräarviot rakennetaan niin, että niistä pyritään tekemään kaikenkattavia vaatimustenmäärittelyn jälkeen. Työmääräarvioiden rakentamiseen menee tämän takia paljon aikaa ja usein kokonaisvaltaisen suunnittelun haittana on se, että suurta osaa erilaisista skenaarioista ei kuitenkaan koskaan pääse tapahtumaan. Työmääräarvioihin vaikuttaa myös tilannetyötekijät projektien alussa, mutta niitä käytetään haastateltavien mukaan todella vähän. Tämä on haastateltavien mielestä huono juttu, koska nimenomaan tietotekniikassa kahdella asiantuntijalla voi olla työn tehokkuudessa jopa 100-kertainen

ero. Haastateltavat kertoivat, että esimerkiksi tietyn osan toteutuksessa asiantuntija voi jäädä jumiin yhteen pulmaan, jossa pahimmillaan kestää monta päivää ratkaista. Toinen asiantuntija taas voi tietää tämän pulman ratkaisun ja rakentaa osion hetkessä. Tätä ei näy haastateltavien mukaan todellisessa työmäärässä eikä työmääräarvioissa. Työmääräarvioita ei myöskään muuteta projektin aikana, mikä aiheuttaa sivuilmiöitä, kuten ohjelmistokehittäjien paineita pysyä kireässä aikataulussa ja huonosti ylläpidettävää ja virherikasta ohjelmakoodia.

Resurssoinnissa haastateltavien mielestä ongelmana on se, että työntekijät resursoidaan hajanaisesti ja pieninä palasina paikasta toiseen. Haastateltava ohjelmistokehittäjä kertoi esimerkin, jossa hän voi olla kuukaudessa pahimmillaan muutaman päivän yhdessä projektissa, sitten vaihtaa kahdeksi viikoksi toiseen projektiin ja jatkaa tämän jälkeen muutaman päivän edellistä projektia. Tämän kahden viikon aikana yksi tai jopa kaksi muuta ohjelmistokehittäjää ovat jatkaneet hänen toteutustaan ja hänellä on mennyt muuten ylimääräistä aikaa heidän perehdyttämiseensä projektin toteutuksen osasta. Sitten kahden viikon päästä haastateltava joutuu edellisen ohjelmistokehittäjän perehdytettäväksi ja aikaa kuluu taas turhaan lisää. Sen lisäksi, että yhtä asiaa teki kolme eri ohjelmistokehittäjää, lopullinen toteutus ei täsmännytkään kaikkien erityisvaatimusten kanssa, jolloin toteutusta piti yhä korjata kiireellä aikataulupaineiden takia. Kun toteutus oli saatu valmiiksi, se toimi, mutta siinä oli yhä muihin osiin vaikuttavia pieniä ohjelmavirheitä. Ohjelmakoodia ei kuitenkaan haluttu tämän jälkeen refaktoroida eli optimoida helposti ylläpidettävän muotoon, koska pelättiin sen menevän korjauksen jälkeen rikki. Tämä oli haastateltavan ohjelmistokehittäjän mukaan esimerkki muutoshankkeesta, jossa alkuperäinen 10 henkilötyöpäivän työ venyi lähes kaksinkertaiseksi resursoinnin takia.

Kuudes kysymys liittyi roolituksen ongelmiin. Roolituksessa suurin ongelma löytyy haastateltavien mukaan siitä, että vaikka yrityksellä on roolitukseen käytäntö niin kukaan ei käytä sitä. Jokaisessa projektissa roolit ovat siis yksilöllisiä ja muuttuvia eli ad hoc-tyylisiä. Monella työntekijällä ei myöskään ole tietoa kaikista omista työtehtävistään, ellei niitä korosteta heille erikseen.

Muita roolitukseen liittyviä ongelmia ovat haastateltavien mielestä osaston ylimmän johdon priorisoinnin puute projekteissa ja asiakkuuksissa ja arkkitehdin ja kehittäjän roolitukset. Haastateltavien mielestä ylimmän johdon pitäisi käydä enemmän strategiaansa läpi ja osallistua päätöksillään eri projektien ja asiakkuuksien priorisointiin johtuen siitä, että työtunnit eivät meinaa riittää kaiken tekemiseen. Arkkitehdin rooli ei varsinaisesti ole epäselvä, mutta se menee haastateltavien mukaan välillä paljon päällekkäin projektipäällikön kanssa. Selainkäyttöarkkitehdin mielestä projektipäällikön roolin jakautuminen on neutraali

asia ja osaston johdon päätös. Nykytilahaastattelun työasemasovellusarkkitehti taas piti roolituksen jakoa ikävämpänä asiana ja toivoi projektipäällikön ottavan enemmän hallinnollista vastuuta nykytilaan verrattuna. Selainkäyttöarkkitehti kuitenkin ilmaisi, että ei hänkään tästä tilanteesta aina pidä ja se vie hänen muiden työtehtäviensä aikaa jonkin verran. Selainkäyttöarkkitehti totesi lyhyen keskustelun jälkeen, että jos projektipäällikkö pystyy hoitamaan hallinnolliset tehtävät, jakamaan työtehtävät ja kommunikoimaan projektista asiakkaan kanssa arkkitehtien pienellä avustuksella, niin se on ihan riittävää.

Haastateltavan ohjelmistokehittäjän mukaan myös ohjelmistokehittäjän ja arkkitehdin roolit menevät usein sekaisin. Hän on esimerkiksi joskus määritellyt arkkitehdille työtehtävät, vaikka perinteisesti roolitus menee toisinpäin. Ohjelmistokehittäjän mielestä nykyinen kehittäjien jako työasemasovellukseen ja selainkäyttösovellukseen on huono, koska komponenttien integrointi keskenään on erittäin kriittinen toimenpide. Hänen mielestään olisi parempi, jos kaikki ohjelmistokehittäjät olisivat esimerkiksi erikoistuneet yhteen komponenttiin, mutta osaisivat silti toteuttaa pieniä osia muiden komponenttien välillä sekä integroida niitä keskenään helpommin. Tämäkään ongelma ei ole yksiselitteinen, koska esimerkiksi nykytilahaastattelun työasemasovellusarkkitehti ilmaisi, että koska eri komponentit ovat monimutkaisia, niiden samanaikainen opettelu ei tuota lisäarvoa osaston kehitystyöhön. Lisäksi työasemasovellusarkkitehti selitti, että työasemasovellus sopeutuu aina selainkäyttösovelluksen arkkitehtuuriin, jolloin integrointi ei hänen mukaansa ole niin monimutkaista. Nykyisessä kehittäjäjaossa on myös se ongelma, että uuden työntekijän sisäänajo on pitkä prosessi ja usein projekteihin valitaan mieluummin vanhoja kokeneita tekijöitä kuin uusia kokemattomia työntekijöitä. Tämä pitkittää heidän oppimisprosessiaan entisestään.

Seitsemäntenä halusin ymmärtää projektien kehitysjonoon liittyviä ongelmia. Haastateltavien mielestä seuraavaksi kehitettävän asian ongelmana on se, että yleensä kehitystyötä jatketaan seuraavaksi helpoimman kehitystyön kautta. Tällä tavoin saa nopeita tuloksia, mutta kaikki vaikeammat ja tuntemattomimmat työt jäävät projektin loppuun, jolloin niihin liittyviin ongelmiin on vaikea reagoida optimaalisella tavalla.

Haastateltavien mielestä myös työtehtävien hallinta Team Foundation Server-ohjelmistolla on ongelmallista. Projektin alussa osasto tekee tehtävät erittäin onnistuneesti, hallitsee niitä hyvin ja tekee testisuunnitelmat niiden pohjalta, joilla ne todetaan valmiiksi. Tehtävien hallinta muuttuu siinä kohtaa haastavaksi, kun muutoshankkeita ja muita mahdollisia muutoksia alkaa syntyä. Projekteissa myös usein tehdään tehtävät erikseen selainkäyttösovellukselle ja työasemasovellukselle, mutta niiden yhdistämisen toteutusta ei usein tutkita ennakkoon. Ohjelmistokehittäjä kertoi, että hänen yhdessä osallistumassaan projektissa on

yli 100 erilaista muutoshanketta ja pitkälti niiden takia projekti on viivästynyt alkuperäisestä suunnitelmasta kaksi vuotta.

Kahdeksantena halusin ymmärtää osaston yhteistyöhön liittyviä ongelmia. Haastateltavien mukaan kaikki työntekijät ovat pääsääntöisesti mukavia ja luotettavia. Yhteistyön ongelmana on enemmän eri komponenttien integrointi. Usein työntekijät haluavat kiinnittää huomiota ja keskustella ainoastaan omasta komponentistaan, jolloin myöhemmin komponenttien integrointi voi osoittautua haastavaksi. Lisäksi Suomen ulkopuolisissa projekteissa ilmenee helposti väärinymmärryksiä, koska kriittisiin kysymyksiin saadaan heikosti ja monen välikäden kautta vastauksia. Yhteistyötä vaikeuttaa myös pitkät maantieteelliset etäisyydet eri toimistojen välillä ja vieraalla kielellä kommunikointi.

Yhdeksäntenä kysymyksenä halusin ymmärtää osaston iteraatioihin ja inkrementteihin liittyviä ongelmia sekä mahdollista sprinttien käyttöönottoa. Haastateltavien mukaan osastolla puhutaan kyllä inkrementeistä ja iteraatioista osana kehitystyötä, mutta oikeasti osasto toimii aikataulutetun vesiputousmallin mukaan. Haastateltavien mukaan iteraatiot suunnitellaan ja toteutetaan aluksi hyvin, mutta heti kun projektin aikana tulee projektipaineita tai aikarajoja, aletaan edetä projektissa yksi ongelma kerrallaan vastoin alkuperäistä iteraatiosuunnitelmaa. Haastateltavien mielestä myös inkrementtejä osastolla ei ole käytössä, vaikka nykytilahaastattelujen asiantuntijoiden mielestä niitä käytetään. Ketteryyshaastateltavien mielestä osasto kyllä tekee aina vanhan version päälle uuden version, mutta niitä ei voida sitouttaa iteraatioihin, jolloin inkrementin idea ei toteudu. Inkrementit perustuvat myös usein pelkkiin vaatimuksiin eikä toimivaan ohjelmistoon, mikä sotii vahvasti inkrementin määrityksen kanssa. Kuitenkin kun toteutus on valmis, niin kehitystyötä tehdään ohjelmavirheitä kierrättämällä eli raportoidaan ohjelmavirhe, jonka jälkeen se korjataan. Sitten vanhaan versioon lisätään uudet ohjelmavirhekorjaukset, korjaus testataan ja mahdolliset uudet ohjelmavirheet raportoidaan.

Sprinttien käyttöönotosta haastateltavat olivat sitä mieltä, että ne pystyttäisiin ottamaan käyttöön tietyillä reunaehdoilla. Haastateltavien mukaan ketterälle projektille pitäisi ensin antaa edellytykset eli projektia ei voida rakentaa enää samalla tavalla kuin nykyinen vesiputousmalli. Projektiohjausta ja seurantaryhmän ymmärrystä pitää haastateltavien mukaan muuttaa sprinttien tapaiseen ajatusmalliin ja asiakas pitäisi saada sitoutettua tähän malliin. Näiden reunaehtojen toteutuessa haastateltavien mielestä sprinttien onnistuminen olisi todennäköistä. Haastateltavat lisäsivät myös sen, että ketterien menetelmien koulutusta olisi myös hyvä lisätä mahdollisten sprinttien onnistumisen takaamiseksi.

Kymmenentenä kysymyksenä halusin ymmärtää mittareiden käytöstä. Kaikkien haastateltavien mielestä mittareita käytetään osastolla liian vähän. Mittareiden käyttö on omalla tavallaan hankalaa, mutta työmääräarvioiden seuraamisen sijaan esimerkiksi tarinapisteet olisivat parempi työn mittayksikkö. Nyt työmääräarvioita ei haluta näyttää asiakkaalle, koska asiakas haluaa projektit yleensä kiinteähintaisina ja asiakas ei ole tietoinen osaston asettamista riskirajoista. Jos taas tarinapisteitä käytettäisiin, työ arvioitaisiin yksilöllisesti jokaiselle iteraatiolle ja pisteet tarkentuisivat projektin edetessä. Lisäksi haastateltavin mukaan ohjelmakoodikatselmointeja pitäisi olla enemmän ainakin kriittisille osille ja niiden tehokkuuteen tarvittaisiin jonkinlainen mittari.

Yhdentenätoista kysymyksenä halusin ymmärtää haastateltavien näkökulmia projektin ketteryyden toteuttamiseen. Nykytilahaastattelussa ilmeni, että asiakas, osaston johto ja projektipäällikkö olivat yleisesti niitä, jotka päättävät projektin ja osaston toimintatavoista. Ketteryysasiantuntijoiden mielestä projektipäälliköllä tulisi olla jatkossa näkemys ketterästä projektista ja ilman Scrum-viitekehyksen tyyppistä Scrummasteria ketterien menetelmien sisäänajoa on vaikea toteuttaa. Asiakas ja osaston johto pitäisi saada sitoutumaan uuteen ketterään malliin ja projekteihin pitää saada mukaan sitoutunut tiimi. Ohjelmistokehittäjän mielestä nykytilan ketterät parannusehdotukset eivät tule parantamaan toimintaa ollenkaan, ellei projekteihin saada sitoutunutta tiimiä, jonka jäsenet eivät vaihda jatkuvasti varausten perusteella projektia ja työtehtäviä.

Kahdentenatoista kysymyksenä halusin selvittää asiakkuuskeskeisyyden ongelmia. Kun nykytilahaastattelussa kerrottiin työntekijöiden jaosta useisiin asiakkuuksiin, niin useita eri näkökulmia tuli kaikilta haastateltavilta. Yhden haastateltavan mielestä asiakkuusjako on ihan hyvä, mutta tiimien jako komponentteihin on huono asia, josta pitäisi siirtyä siihen, että koko tiimi osaisi tehdä kaiken ilman suurta ulkopuolista apua. Toinen haastateltava kokee taas, että asiakkuuskeskeisyydestä pitäisi päästä pois ja tiimit tulisi jakaa nimenomaan omiin komponentteihinsa. Kolmas haastateltava ei pidä asiakkuuskeskeisyyttä ongelmana niin kauan, kun tiedon jakaminen toimii.

Kolmantenatoista kysymyksenä halusin ymmärtää palavereihin liittyvistä ongelmista. Palaverien ongelmana on usein se, että niihin kutsutaan varmuuden vuoksi paljon henkilöitä. Palavereihin olisi myös hyvä lisätä agenda heti kalenterikutsuun, jotta osaisi valmistua niihin paremmin. Isot palaverit voivat myös olla ongelmallisia, mikäli niissä käydään keskustelua osallistujien kanssa. Isot palaverit toimivat haastateltavien mielestä ainoastaan tiedotustilanteissa.

Neljäntenätoista kysymyksenä halusin ymmärtää tiedon jakamisessa esiintyviä ongelmia. Tiedon jakamisen ongelmana on haastateltavien mukaan se, että oikeata tietoa on välillä vaikea löytää. Tärkeitä tietoja ylläpidetään monia vuosia vanhoissa, sähköposteista tehdyissä kansioissa tai osaston omilla cabin-sivustoilla, joista tietoa on vaikeasti saatavilla. Tiedonkululla on myös usein projektirajat ja omasta tuotteesta tiedetään hyvin vähän. Projektirajoilla haastateltavat tarkoittavat sitä, että tietoa ei jaeta herkästi oman projektin yli, jolloin sama työ voidaan tehdä kahdesti ja yhteensopivuus oman tuotteen kanssa kärsii. Oman tuotteen kehityksestä haastateltavat tiesivät kovin vähän ja toivoivatkin tähän pikaista muutosta.

Tiivistettynä ketteryysasiantuntijat olivat pitkälti samaa mieltä osaston ongelmista. Ketteryysasiantuntijat näkivät ketteryyden tapana ajatella asioita ja olla avoimena muutokselle. Ketteryysasiantuntijat näkivät siis ketterät menetelmät enemmän ajatusmallina kuin nykytilahaastattelun asiantuntijat. Projektin elinkaaressa asiantuntijoiden mielestä suurin ongelma on muutoshallinta. Muita elinkaariongelmia ovat projektin alun suunnittelu liian tarkasti ja liian pitkälle aikavälille, asiakkaan vaikeus ymmärtää ratkaisukuvausta, järjestelmien välisen vuorovaikutuksen dokumentoinnin puuttuminen, oman tuotteen ja asiakaskerroksen samanaikainen kehitys ja opit-keskustelun parannusten jääminen ideatasolle. Organisointiin liittyvissä ongelmissa asiantuntijoiden mielestä työmääräarviot, tilannetyötekijät ja hajanainen resurssointi olivat suurimpia ongelmia. Organisointiongelmissa ilmeni myös työntekijöiden hajanainen ja pieninä palasina paikasta toiseen tapahtuva resurssointi. Roolitusongelmia olivat asiantuntijoiden mukaan roolituksen epäselvyys, osaston ylimmän johdon priorisoinnin puute projekteissa ja asiakkuuksissa, arkkitehdin, projektipäällikön ja kehittäjän roolitusten päällekkäisyys ja laaja sisäänajo uusien työntekijöiden kanssa. Seuraavana kehitettävien osien ongelmana oli haastateltavien mielestä kehitystyön tekeminen seuraavaksi helpoimman kehitystyön kautta, tehtävien hallinta Team Foundation Server-ohjelmistolla ja selainkäyttösovelluksen ja työasemasovelluksen yhtenäisen toteutuksen pohtiminen projektin alussa. Yhteistyöhön liittyviä ongelmia ovat eri komponenttien integroinnista keskustelemisen puuttuminen, kansainväliset projektit, maantieteelliset rajat ja vieraalla kielellä kommunikointi. Iteraatioiden ja inkrementtien ongelmana on se, että niitä ei haastateltavien mielestä kunnolla ole sovelluskehityksen aikataulutetun vesiputousmallin takia. Sprinttien käyttöönotto voisi tapahtua heidän mielestään osaston työntekijöiden koulutuksen ja tiettyjen reunaehtojen kautta. Mittaamisongelmina haastateltavien mielestä oli se, että toimintaa ei mitata tarpeeksi ja siinä ei olla kovin läpinäkyviä muille sidosryhmille. Ketteryysasiantuntijoiden mielestä ketteryys tulisi jatkossa toteuttaa projektipäällikön ja Scrummasterin yhteistyöllä ja saada johto ja asiakas ketterään kehittämiseen mukaan. Lisäksi sitoutunut tiimi on ehdoton vaatimus haastateltavien

mielestä ketterien menetelmien sisällyttämiselle. Siiloutumisesta eli asiakkuuskeskeisyydestä asiantuntijat eivät olleet yksimielisiä, mutta vahvan tiedonsiirron yhteydessä kaikkien mielestä siiloutuminen toimii. Palaverien ongelmana on liian suuri osallistumismäärä ja kokouksen kutsun epäselvyys ja tiedonjakamisessa asiantuntijat kokivat välillä vaikeaksi löytää oikeata tietoa, saada tietoa projektirajojen yli ja ymmärtää nykytilaa oman tuotteen kehityksessä.

4 Kehitystoimenpiteet

4.1 Ketteryyttä lisäävät keinot

Teoriaosuuden, nykytilahaastattelun ja ketteryyshaastattelun perusteella yksinkertaisin ketteryyttä lisäävä keino on Scrum-viitekehyksen eri käytäntöjen käyttöönotto. Scrumin säännöt ja periaatteet ovat yksinkertaisia ja helppoja kommunikoida eteenpäin, jolloin ne sopivat yrityksen osaston sovelluskehitykseen hyvin.

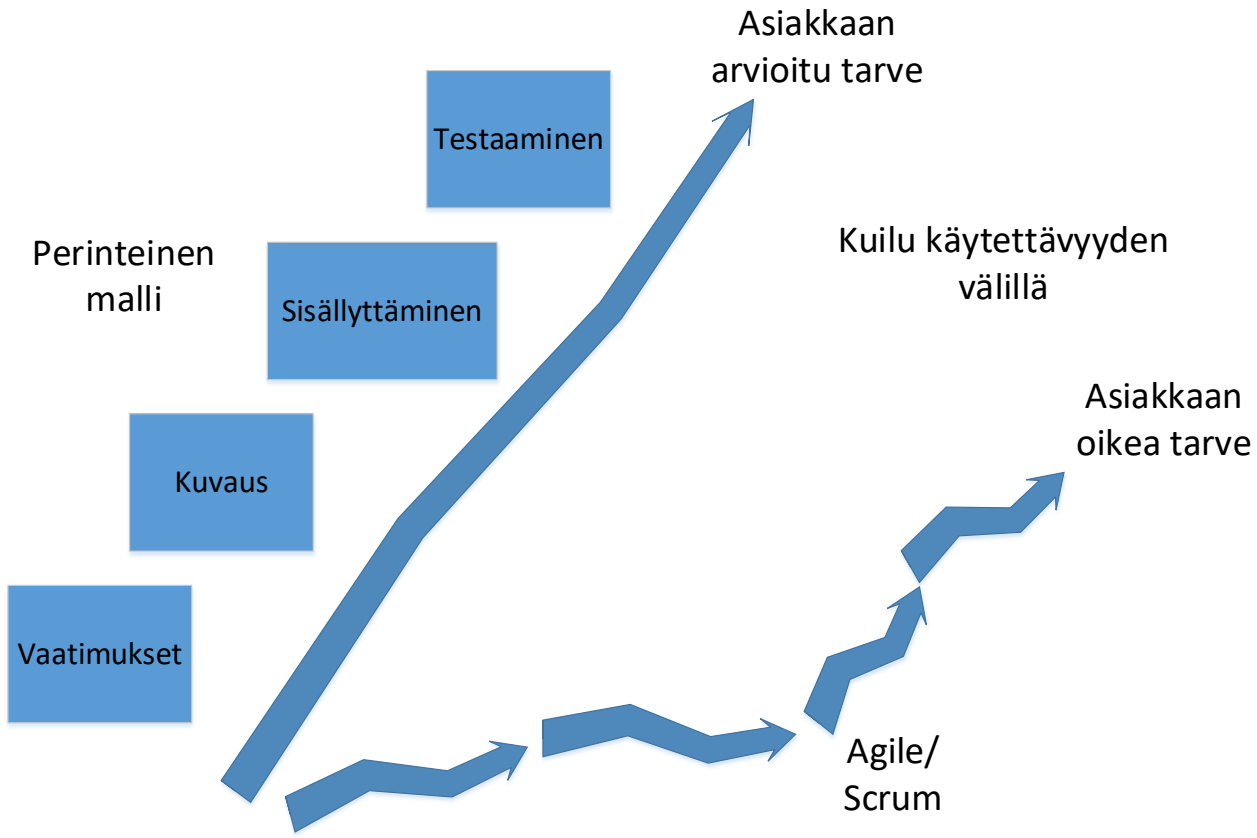
Scrumin ensimmäinen ketteryyttä lisäävä keino on iteraatiomainen lähestymistapa projektityöhön. Työn toimeksiantajan ketteryyshaastattelussa ilmeni projekteissa ongelmaksi se, että projektit suunnitellaan aluksi liian tarkasti ja liian pitkälle. Projektin alussa tehdään kaiken kattava vaatimustenmäärittely, keskivaiheessa projektipaineet ja tarkka aikataulutus alkavat aiheuttaa muutoksia projektin alkuperäiseen aikatauluun ja kaikista vaikeimpien työtehtävien toteutus jätetään projektin loppuun.

Tärkeimpien vaatimusten kartoittaminen ja tuotteen kehitysjonon rakentaminen voi olla aluksi vaikeaa, koska usein asiakas tietää vain osittain haluamansa ominaisuudet ja asiakkaalla voi olla monta edustajaa neuvotteluissa, joilla jokaisella on omat prioriteettinsa vaatimuksista. Tästä syystä tuotteen vision ja tuotteen kehitysjonon rakentaminen heti projektin alussa asiakkaan läsnä on tärkeä ketterän projektin mahdollistaja. Kun asiakkaan kanssa on saatu sopimus aikaiseksi, olisi hyvä pitää osaston kehitystiimin ja asiakkaan kanssa työpaja. Työpajassa voitaisiin keskustella asiakkaan tärkeimmistä ominaisuuksista lopullisen tuotteen kannalta ja luoda täten heti yhteinen ymmärrys toteutuksesta ja inkrementtimallista. Tämän jälkeen kehitystiimi alkaa asiakkaan läsnä aikaisemman kokemuksen kautta pisteyttämään näitä tärkeimpiä ominaisuuksia ja auttaa asiakasta ymmärtämään, miten tuote on järkevintä toteuttaa suhteutettuna asiakkaan omaan ympäristöön ja markkinatilanteeseen. Kun asiakas on koko ajan suunnittelussa mukana, asiakkaan edustajat pääsevät tutustumaan osaston kehitystiimiin ja he pääsevät heti esittämään askarruttavia kysymyksiä suoraan tuotteen toteuttajilta. Tätä kautta ketteryydelle tärkeä luottamus ja sitoutuminen kasvaa molemminpuolisesti, koska asiakas saa heti projektin alussa lisäarvoa ja kehitystiimi voi aloittaa heti merkityksellisen työn toteuttamisen. (Overeem 15.7.2016).

Sprinttien käyttöönottaminen tulee lisäämään merkittävästi osaston ketteryyttä. Kun osasto aloittaa oman tuotteen räätälöinnin asiakkaan tarpeisiin, sen ei tarvitse Scrumin

mukaan suunnitella etukäteen kuin yksi sprintti kerrallaan ja projektin alussa tuotteen kehitysjono. Tuotteen kehitysjonon rakentamisesta puhutaan seuraavassa kappaleessa. Kun ratkaisusta aletaan keskustella asiakkaan kanssa, siinä pitäisi ensin selvittää asiakkaan kriittisimmät vaatimukset. Kun kriittiset vaatimukset on onnistuttu kartoittamaan, voidaan ne lisätä tuotteen kehitysjonoon ja aloittaa ensimmäisen sprintin suunnittelu. Tällöin koko projektin aikataulu ei ole vielä lukkoon lyöty ja osaston kehitystiimin on helppo keskittyä toteuttamaan yhden sprintin tehtävät. Samalla työtä ei jatketa helpoimman ominaisuuden kautta, vaan asiakkaalle eniten arvoa tuottavien ominaisuuksien kautta. Asiakas voi siis itse päättää, mitä osasto lähtee toteuttamaan seuraavaksi. (Schwaber & Sutherland 2017, 10-11).

Scrumin sprintit auttavat myös katselmointien ja läpinäkyvämmän asiakasyhteistyön kanssa osaston asiakkaan kanssa ilmeneviin ongelmiin. Osastolla on ollut vaikeuksia ymmärtää aluksi asiakkaan oikeaa tarvetta, koska asiakas esittää ”pykälälistana” vaatimuksia ja osasto tarjoaa niille ratkaisun. Asiakkaan esittämä pykälälista on usein poikkeava lopullisista asiakkaan vaatimuksista, jolloin osasto joutuu myöhemmin pitkittämään projektin kestoa erilaisten muutoshankkeiden vuoksi. Asiakkaan tarpeen määritteleminen on osastolle myös projektin alussa vaikeaa, koska asiakas harvoin tietää täysin mitä haluaa ja asiakkaan organisaatiossa ja taloudellisissa intresseissä voi tapahtua äkillisiä muutoksia. Ketteryyshaastattelussa kerrottiin, että asiakas koostuu yleensä monesta edustajasta, jotka haluavat priorisoida omat ideansa lopulliseen tuotteeseen. Asiakas pääsee myös harvoin näkemään tuotetta ennen kuin se on lähes valmis, jonka takia asiakas ymmärtää vasta projektin lopussa oikeasti oman tarpeensa. Kuva neljä esittää haastetta perinteisen mallin asiakastarpeen määrittelemisessä, kun asiakkaan tarpeen oletetaan pysyvän samana koko projektin ajan. Näin voi joskus käydä, mutta nykytilahaastattelun perusteella projekteissa lähes aina asiakas haluaa muuttaa alkuperäisen suunnitelman mukaisia vaatimuksia.



Kuva 4. Asiakkaan tarpeen määritteleminen perinteisessä mallissa ja ketterässä mallissa (mukaillen Leffingwell 2007, 294)

Kuvan neljä ketterä malli auttaa löytämään paremmin asiakkaan oikean tarpeen. Sprinteissä käytettävä sprintin katselmointi auttaa osastoa kehittämään tuotettaan asiakkaalle sopivammaksi ja minimoimaan turha työ asiakasprojekteissa. Kun jokaisen iteraation lopussa pidetään katselmointi, sinne tulisi saapua kaikki asiakasorganisaation tärkeimmät edustajat, jotta katselmoinnissa saadaan mahdollisimman laaja ymmärrys kehitettävän tuotteen suunnasta. Kun tuoteomistaja kertoo tuotteen tilanteesta ja valmistumisajankohdasta, edustajille syntyy hyvä ymmärrys työn edistymisestä ja he voivat miettiä samalla työn edistymisen vaikutusta oman organisaationsa toimintaan. Katselmoinnissa käydään läpi myös asiakkaan markkinatilanne ja budjetti. Jos asiakkaalla on esimerkiksi taloudellisia ongelmia ja he ovat ennen viimeistä arvioitua iteraatiota tyytyväisiä tuotteen toiminnallisuuksiin, projekti voidaan lopettaa ja asiakas ei joudu maksamaan arvoa tuottamattomista ominaisuuksista mitään. (Schwaber & Sutherland 2017, 13-14).

Katselmoinnista on myös yhteistä hyötyä sekä osaston kehitystiimille että asiakkaalle. Kun katselmoinnissa keskitytään palautteen antamiseen ja työn suunnitteluun, kehitystiimi oppii omista virheistään ja löytää paremman tavan suunnitella työtään jatkon kannalta. Ketteryyshaastattelussa ilmeni myös, että kun tuoteomistaja kertoo katselmoinnin alussa työn

etenemisen tilan, asiakas voi keskittyä paremmin oman tarpeensa kommunikointiin tuotteen esittämisen aikana. Asiakas näkee jokaisessa katselmoinnissa tehokkaampia tuloksia ja muutoshankkeet voidaan lisätä ja priorisoida tuotteen kehitysjonoon ilman ylimääräistä hallinnollista työtä.

Työn jakaminen sprintteihin auttaa myös työpaineiden minimoimisessa. Sprintin suunnittelupalaveri koostuu kahdesta eri osasta, jossa ensimmäisessä osassa keskustellaan toteutettavista ominaisuuksista ja käyttäjätarinoista ja toisessa osassa suunnitellaan niiden toteutus. Kehitystiimi voi ensimmäisessä palaverissa ilmaista oman työmääräarvionsa, jolloin kukaan tiimin ulkopuolinen päättäjä ei voi vaikuttaa työn kestoon. Lisäksi toisessa palaverissa kehitystiimi suunnittelee yksin sprintin toteutustavan ja saa tätä kautta vapautta suhteuttaa työtä tasapuolisesti ja realistisesti jokaiselle jäsenelle saman verran. Kehitystiimin työn jako menee varmasti harvoin tasan, mutta tämä auttaa esimerkiksi uusien työtehtävien oppimisessa. (Schwaber & Sutherland 2017, 10-12). Esimerkiksi jos työasemasovellukseen ei ole tulossa seuraavan sprintin aikana muuta kuin pieniä muutoksia, työasemasovelluksen kehittämiseen erikoistunut työntekijä voi opetella selainkäyttösovelluksen kehittämistä selainkäyttösovelluksen kehittämiseen erikoistuneen työntekijän kanssa. Tämä lisää osaston työhön jatkuvan kehittymisen periaatteen, joka on elintärkeätä ketterän sovelluskehityksen onnistumisen takaamisessa.

Kun työmäärä suunnitellaan viikkotasolla suunnittelupalaverissa ja sitä seurataan päivätasolla päiväpalavereissa, työmääräpoikkeamat huomataan nopeasti ja niihin voidaan reagoida tehokkaasti. Työ ei myöskään ole ketterässä kehityksessä pirstaloitunutta ja varauksiin pohjautuvaa, vaan esimerkiksi yhden ominaisuuden toteuttaminen ja testaaminen tapahtuu kerralla loppuun yhden sprintin aikana. (Schwaber & Sutherland 2017, 11). Ketteryyshaastattelun selainkäyttöarkkitehdin mukaan sprintin aikana voidaan myös esimerkiksi heti selvittää ohjelmavirheen korjaamistarpeet ja muokata sprintin kehitysjonon kohtia niin, että korjaaminen saadaan toteutettua. Jos joku käyttäjätarina jää korjaamisprosessin takia tekemättä ja se ei vaikuta sprintin tavoitteeseen, käyttäjätarina voidaan palauttaa kehitysjonoon ja toteuttaa esimerkiksi seuraavassa sprintissä. Tämä vähentää paljon ohjelmistokehittäjien päittäin vaihtamisesta johtuvaa ylimääräistä perehdyttämistyötä ja lisää todennäköisyyttä toimivan ohjelmiston rakentamiseen jokaisen iteraation lopuksi.

Myöhemmissä suunnittelupalavereissa keskustellaan myös kehitystiimin edellisten sprinttien nopeudesta ja työn määrästä. Ensimmäinen iteraatio voi olla vielä suhteellisen epätarkka antamaan oikeaa työmääräarviota, koska kehitystiimi ei vielä siinä vaiheessa tiedä kaikista työhön vaikuttavista tekijöistä. Kun kehitystiimi etenee aina seuraavaan iteraati-

oon, se tarkastelee työtään ja osaa tarkentaa koko ajan yhä paremmin omia työmääräarvioitaan. Nämä vaikuttavat jokaisessa sprintissä työn aikatauluttamiseen, jolloin kehitystiimille tulee ainoastaan yhden sprintin aikataulupaineet kerralla. Lyhyellä aikavälillä on myös helppo puuttua selkeisiin ongelmakohtiin ja siten lisätä kehitystiimin tehokkuutta vielä projektin alkuun verrattuna. Työmäärän ja nopeuden jatkuvan säätämisen avulla projektipaineet ja mahdolliset muutokset saadaan helposti otettua kehitystyössä huomioon. (Schwaber & Sutherland 2017, 14).

Päiväpalaverit ovat hyvä ketteryyttä lisäävä keino. Koska päiväpalaverit ovat aikarajattuja ja koskevat yleensä ainoastaan yhtä kehitystiimiä, tuoteomistajaa ja Scrummasteria, palaveri ei veny ja sinne ei kutsuta liikaa ihmisiä. Päiväpalavereissa pystytään puuttumaan kehitystyön ongelmiin helposti, kun kaikki kehitystyöhön osallistuvat ovat samassa tilassa. Kehitystiimin jäsenillä on myös erilaisia vahvuuksia ja heikkouksia ja he saattavat katsoa asioita vain yhdestä näkökulmasta. Tämän takia jokin monimutkainen ongelma voidaan ratkaista heti päiväpalaverissa, jos esimerkiksi tiimin toinen ohjelmistokehittäjä huomaa huolimattomuusvirheen toisen ohjelmistokehittäjän toteutuksessa. (Schwaber & Sutherland 2017, 12-13).

Retrospektiivien käyttöönotto jokaisen sprintin päätteeksi auttaa myös osaston parannusten toimeenpanossa. Kun kehitystiimi saa itse arvioida omaa toimintaansa jokaisen sprintin jälkeen, saadaan tarkempia kehitysideoita kuin esimerkiksi vuositasolla tai projektikohtaisesti mitattuna. Kun kehitystiimi pääsee itsenäisesti pohtimaan parempia toimintatapoja, sen on helpompi miettiä niihin suoria ratkaisuja. Yleensä projektipäälliköt vetävät projektien lopussa opit-keskustelua, mutta niissä kehityksestä vastaavat työntekijät osallistuvat ainoastaan välillisesti keskustelemalla projektin hyvistä ja huonoista asioista. Retrospektiiveissä siis kehitystiimi keskittyy haasteiden havainnointiin ja korjaamiseen. Kun kehitystiimi huomaa kehitysprosessissa haasteita, niistä yksi voidaan ottaa käyttöön seuraavaan sprinttiin. Näin päästään lyhyellä aikavälillä kokeilemaan parannusta käytännössä ja saamaan nopeita tuloksia. Parannuksia on hyvä myös olla aina vain yksi kerrallaan, koska muuten niiden seuranta sprintin aikana voi muuttua haastavaksi. Uusien parannuksien läpikäyminen voidaan aina toteuttaa seuraavan sprintin retrospektiivissä ja analysoida sen tarpeellisuus jatkossa. (Schwaber & Sutherland 2017, 14).

Asiakkaan edustajilla on myös usein hankaluuksia ymmärtää osaston käyttötapadokumentteja. Ketteryyshaastatteluissa ilmeni, että asiakas ei aina ymmärrä dokumenttien sisältöä ja ei aina muutenkaan lue läpikotaisin kaikkia toimitettavia dokumentteja. Asiakkaalla on myös haastateltavien mukaan suuri luottamus siitä, että osaston asiantuntijat

ovat lisänneet dokumentteihin kaiken tarpeellisen räätälöinnin onnistumiseksi. Tämän takia asiakkaan ja osaston välillä on epäselvä ymmärrys valmiista tuotteesta ja toteutukseen lisätään paljon muutoshankkeita, jotka pitkittävät projektin aikataulua entisestään.

Tästä johtuen Scrumin toinen ketteryyttä lisäävä keino on käyttötapadokumenttien vaihtaminen käyttäjätarinoihin. Kun tärkeimmät ominaisuudet on ketterän mallin mukaan kartoitettu, niitä voidaan alkaa kirjata käyttötapadokumenttien sijaan käyttäjätarinoilla. Käyttäjätarinoiden etuna on se, että kuka tahansa ymmärtää niiden sisällön ja jokaiselle käyttäjätarinalla on omat hyväksymiskriteerit. Kun käyttäjätarinoita toteutetaan, kehitystiimin pitää toteuttaa käyttäjätarina niin hyvin, että se täyttää hyväksymiskriteerit. Tällöin sekä asiakas että osasto voi todeta yksimielisesti käyttäjätarinan valmiiksi. Käyttäjätarinat auttavat myös kehitystiimiä huomioimaan järjestelmien väliset riippuvuudet, koska hyväksymiskriteeriksi voidaan kirjoittaa yhteensopivuuden testaus ennen sprintin aikana toteutetun inkrementin julkaisemista. Käyttäjätarinoiden kirjoittaminen on myös hyvä jakaa tasan koko Scrum-tiimille, jotta kaikki osaavat tehdä niitä tarvittaessa ja jokainen ymmärtää niiden tarkoituksen. (Stellman & Greene 2015, 143-145).

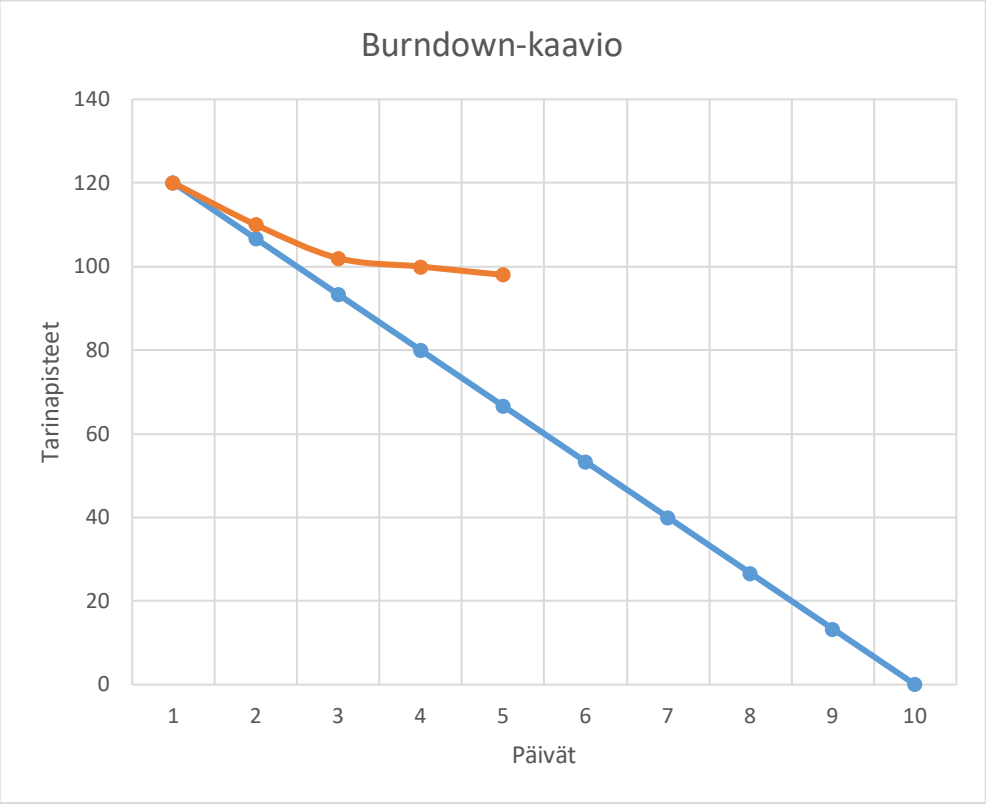
Kun osasto alkaa käyttämään käyttäjätarinoita, niin niiden työmääräarviota ei voi toteuttaa enää samalla tavalla kuin nyt. Käyttäjätarinoille suositellaan käyttävän tarinapisteitä, koska ne kuvaavat paremmin työmäärää. Tarinapisteet lasketaan jokaiselle sprintille erikseen eli niitä ei tehdä projektin alussa loppuun ja tarinapisteet tarkentuvat projektin edetessä eli ne ovat dynaamisia projektin alusta loppuun. (Stellman & Greene 2015, 146-147).

Tarinapisteiden kautta saadaan huomattavasti realistisempi arvio työmäärästä, koska niiden toteutumista seurataan päivätasolla ja iteraatioittain. Osaston haasteena oli työmäärien arvioiminen, jolloin tarinapisteet mahdollistaisivat työmääräarvioiden muuttamisen vielä projektin loppupuolellakin. (Stellman & Greene 2015, 147).

Tarinapisteet saavat myös koko kehitystiimin keskittymään työn arviointiin ja luomaan jatkuvasti parempia työmääräarvioita. Nykytilahaastattelun mukaan osaston tiimijohtajat hoitavat työntekijöiden organisoinnin ja kehitystyön sisäisen organisoinnin hoitavat projektipäälliköt, arkkitehdit ja testauspäälliköt. Suurin osa osaston työntekijöistä ei siis pääse vaikuttamaan työmäärien tekoon ja osaston pitää olettaa, että näiden muutamien työntekijöiden kokemus riittää koko projektin mittaisen työmäärän arviointiin. Tästä seuraa ketteryyshaastattelun mukaan paljon työpaineita ja heikkoja toteutusratkaisuja. Jos osasto

vaihtaisi työn resurssoinnin ainakin tiimien sisällä itseohjautuvaksi, työmäärät olisivat realistisempia. Työmäärä on kuitenkin huomattavasti helpompi arvioida tiimin sisällä kaksi viikkoa eteenpäin kuin kaksi kuukautta eteenpäin.

Tarinapisteiden hyvä puoli on myös se, että niistä saa tehtyä helposti burndown-kaavion. Burndown-kaavio auttaa sprintin päivittäisen työn seurannassa. Kaavion avulla tuoteomistaja ja kehitystiimi voi helposti huomata työmäärän poikkeamat ja kommunikoida niistä asiakkaan kanssa. (Stellman & Greene 2015, 148). Kuva viisi esittää sprintin aikana tapahtuvaa burndown-kaaviota. Kun alkuperäinen työmääräarvio on tavoite, sitä voidaan helposti esittää suoralla viivalla sprintin ensimmäisestä päivästä viimeiseen päivään. Kun varsinainen sprintti aloitetaan, oikeasta työmäärästä tehdään toinen viiva, joka muokataan joka päivä. Kuvasta viisi voidaan esimerkiksi huomata, että viidennen päivän kohdalla on yhä noin 100 tarinapistettä jäljellä, vaikka alkuperäisen arvion mukaan tarinapisteitä pitäisi olla enää noin 70 jäljellä. Tästä voidaan päätellä, että sprintin toteutuksessa on tapahtunut jokin suurempi ongelma, tai kehitystiimi on arvioinut oman tehokkuutensa väärin. Jos poikkeamana on kehitystiimin toteutuksessa ilmennyt ongelma, siihen voidaan heti reagoida ja kertoa asiakkaalle ongelman tausta. Jos poikkeama on kehitystiimin oman tehokkuuden yliarviointi, vähiten arvoa tuottavia käyttäjätarinoita voidaan siirtää seuraavaan sprinttiin niin paljon, että työmäärä palaa takaisin arvioidun työmäärän kohdalle. Burndown-kaaviot ovat myös erinomainen esimerkki ketterästä iteraatiomittarista.



Kuva 5. Sprintin burndown-kaavio (mukaillen Parkhe 15.7.2015)

Jotta sprintit voidaan ottaa käyttöön, osastolle tarvitaan myös uusia rooleja. Scrumin kolmas ketteryyttä lisäävä keino on siis kehitystiimin, Scrummasterin ja tuoteomistajan käyttöönotto. Nykytilahaastattelun mukaan osastolla on tällä hetkellä hyvin sekalaiset roolit arkkitehdeillä, ohjelmistokehittäjillä ja projektipäälliköillä. Projektipäällikkö on delegoinut työtehtäviään arkkitehdeille ja testauspäälliköille ja usein osaston työntekijät eivät tiedä täysin omia työtehtäviään projekteissa. Osastolla on myös jaettu ohjelmistokehittäjät ja arkkitehdit selainkäyttösovelluskomponenttiin ja työasemasovelluskomponenttiin. Selkeä Scrumin roolitus auttaisi siis epäselvän roolituksen ratkaisemisessa.

Osaston ketteryyttä lisäävistä tekijöistä yksi merkittävin on kehitystiimin käyttöönotto. Kun osasto muuttaa nykyiset suuret määrät työntekijöitä kehitystiimeiksi, on työn hallinta huomattavasti helpompaa. Kun kehitystiimi toimii itseohjautuvasti eli suunnittelee oman työnsä jokaisen sprintin aikana, työn laatu paranee. Työtä ei tarvitse tehdä jatkuvalla kiireellä ja kaikilla kehitystiimin jäsenillä on selkeä päämäärä lyhyelle aikavälille.

Osaston kehitystiimi ei voi koostua kymmenistä työntekijästä, vaan kehitystiimin tulee koostua teoriaosuuden mukaan 3-9 henkilöstä. (Schwaber & Sutherland 2017, 6-7). Kun kehitystiimi on pieni, sen täytyy olla moniosaava. Kehitystiimin sisällä pitää olla sekä selainkäyttösovelluksen ja työasemasovelluksen osaajia, koska vain silloin kehitystiimeistä on hyötyä. Kun kehitystiimeihin lisätään kaikkia eri osaajia osastolta, kehitystyön kommunikaatio toimii paremmin. Eri komponentit keskustelevat helposti yhdessä, koska niitä kehitetään samaan aikaan ja kehitystiimi pystyy heti testaamaan inkrementin toimivuuden. Kehitystiimi voi esimerkiksi heti korjata kriittisen ohjelmavirheen, koska se huomataan parhaimmillaan heti kehitystyön jälkeen. Tällä hetkellä osastolla testaamisessa voi olla pahimmillaan monen kuukauden viive, jolloin kriittisetkin ohjelmavirheet huomataan kehitystyön näkökulmasta liian myöhään. (Schwaber & Sutherland 2017, 6-7).

Kehitystiimin sisällä ei myöskään ole titteleitä tai hierarkiaa, jolloin kehitystiimi päättää keskenään toteutuksen työn jakautumisen. Osastolla siis periaatteessa kaikkien työntekijöiden olisi hyvä osasta ainakin hieman jokaista työosa-aluetta. Tämä tarkoittaa sitä, että esimerkiksi testaajien pitäisi opetella ohjelmointia ja palvelimienhallintaa ja myös arkkitehdit osallistuisivat testien ajamiseen. Tämä ei kuitenkaan tarkoita sitä, että työntekijöiden erikoisosaamisia hylättäisiin kokonaan, koska aina tulee olemaan teknisesti vaikeita toteutuksia, johon tarvitaan kokeneen asiantuntijan taitoja. Yleisestä osaamisesta on kuitenkin paljon hyötyä, koska silloin parhaimmillaan yksi ihminen pystyy tekemään seitsemän ihmisen työt. (Schwaber & Sutherland 2017, 6-7).

Kehitystiimi auttaa myös työntekijöiden organisoinnissa. Kun sama kehitystiimi pysyy koko projektin ajan samana, osaston ei tarvitse kärsiä osittaisen varaamisen haittapuolista. Kun kehitystiimi tekee kaiken alusta loppuun yhtenäisesti, työtä ei tarvitse jättää kesken ja turhaa perehdyttämistyötä ei tapahdu. Kehitystiimi auttaa myös uusien työntekijöiden kouluttamisessa, koska he voivat mennä samaan tiimiin kokeneen asiantuntijan kanssa. (Schwaber & Sutherland 2017, 6-7). Asiantuntija voi samalla opettaa tarpeen mukaan uutta työntekijää ja seuraavassa projektissa uusi työntekijä ymmärtää jo hyvin osaston käytännöt ja osaston tuotteen taustan.

Tuoteomistajan rooli on myös tärkeä. Tuoteomistaja hallinnoi tuotteen kehitysjonoa ja vastaa kehitystiimin tavoitteiden asettamisesta. Tuoteomistaja auttaa myös asiakkaan kanssa kommunikointiin, koska kehitystiimi voi vähintään viikon välein keskustella sprintin työtehtävistä ja tällöin minimoida turha työ. (Schwaber & Sutherland 2017, 6).

Tuoteomistaja on usein asiakkaan edustajan rooli, mutta ketteryyshaastattelun asiantuntijat ilmaisivat, että tuoteomistaja voisi aluksi olla osaston sisäpuolelta. Kun osasto saa kokemusta ketteristä projekteista ja asiakas ymmärtää tuoteomistajan roolin, voidaan tuoteomistaja vaihtaa asiakkaan edustajaksi. Osaston sisällä tuoteomistajan roolia voitaisiin kokeilla tuotantopäälliköillä aluksi, koska heiltä löytyy sekä teknistä että liiketoiminnallista ymmärrystä osaston sovelluskehityksestä.

Scrummaster on myös tärkeä osa Scrumin roolitusta. Scrummasteria tarvitaan hallinnoimaan viitekehyksen toteutumista. Scrummaster ymmärtää Scrumin ja mahdollisesti muiden viitekehyksien teorian erinomaisesti, jolloin hän pyrkii varmistamaan, että osaston toiminta on ketterää. Scrummaster vastaa myös siitä, että kaikki Scrumin tapahtumat järjestetään. (Schwaber & Sutherland 2017, 7-9).

Scrummaster pyrkii myös auttamaan tuoteomistajaa ja kehitystiimiä omissa työtehtävissään. Scrummaster auttaa tuoteomistajaa kehitysjonon hallinnassa. Tuoteomistajan pitää muokata kehitysjonon järjestys sellaiseen muotoon, että se tuottaa mahdollisimman paljon arvoa. Scrummaster auttaa tämän mahdollistamisessa ja tarpeen mukaan voi ottaa hetkellisesti tuoteomistajan roolin omakseen. Kehitystiimin kannalta Scrummaster valmentaa heitä kohti itseohjautuvaa ja moniosaavaa tiimiä. Scrummaster pyrkii myös poistamaan haitallisia esteitä tiimin työn edistymisen takaamiseksi ja tuottamaan tällä tavoin tiimille mahdollisimman suurta arvoa. Scrummaster voi esimerkiksi päiväpalaverissa huomata kehitystiimin tarvitsevan nopean asiakaspalaverin sprintin aikana toteutettavaan ominaisuuden takia, jolloin hän voi hoitaa sen järjestämisen. (Schwaber & Sutherland 2017, 6-7).

Scrummaster pyrkii myös parantamaan Scrumin ja muiden viitekehysten tehokkuutta ja valmentaa kaikkia sidosryhmiä ymmärtämään ketteryyden merkityksen. Scrummaster pyrkii jatkuvasti muiden Scrummasterien kanssa kehittämään toimintaa paremmaksi, koska minkään tiimin osaaminen ja käytännöt eivät ikinä ole täydellisiä. Lisäksi Scrummaster pyrkii selventämään esimerkiksi muulle organisaatiolle ja asiakkaalle Scrumin käytännöistä, jotta kaikki projektien sidosryhmät ovat selvillä kaikista tapahtumista. (Schwaber & Sutherland 2017, 9).

Scrumin lisäksi ketteryyttä voidaan lisätä oman tuotteen hallinnoimisessa. Oman tuotteen samanaikainen kehitys asiakaskerroksen kanssa on osastolla suuri haaste, jota voisi auttaa vakaan oman tuotteen rakentaminen. Ketteryyshaastattelussa ohjelmistokehittäjä kertoi, että hänen mielestään lähes kaikkia asiakkaan järjestelmän vaatimuksia tulisi kohdella aluksi omana kerroksenaan. Kun asiakkaan järjestelmää on käytetty esimerkiksi puoli vuotta tuotannossa, niin vasta sitten hyödylliset toiminnot voidaan viedä omaan tuotteeseen. Tämä auttaisi projektien viivästymisiin, koska silloin osaston kehitystiimin ei tarvitsisi odottaa täysin testattua omaa tuotetta, vaan voisi suoraan rakentaa jokaisen senhetkisen asiakasprojektin oman tuotteen päälle. Myös muut ketteryyshaastattelun asiantuntijat olivat tästä samaa mieltä.

Tiedon jakaminen on nykytilahaastattelun mukaan myös haastavaa osaston sovelluskehityksessä. Tieto ei liiku hyvin projektien välillä ja päivittäiset työntekijät ovat sijoittautuneet ympäri Suomea ja ulkomaita. Tähän osasto pystyy helposti lisäämään ketteryyttään, kun se hyödyntää paremmin nykyisiä järjestelmiään.

Osasto käyttää Microsoftin Team Foundation Server-ohjelmaa ohjelmavirheiden ja työtehtävien hallintaan. Kun osasto ottaa käyttöönsä sprintit, kehitystiimit ja käyttäjätarinat, sen pitää löytää uusia työkaluja näiden hallitsemiseen. Osaston ei kuitenkaan tarvitse hankkia näitä varten uusia ohjelmia, vaan he voivat hyödyntää jatkossa laajemmin Team Foundation Server-ohjelmaa. Osasto voi esimerkiksi luoda työryhmiä kehitystiimejä varten, kirjoittaa käyttäjätarinoita sähköisesti ja hallinnoida sprinttien aikataulua ja kehitysjonoa suoraan Team Foundation Server-ohjelmalla. Osaston jokainen työntekijä osaa käyttää ainakin pääpiirteittäin ohjelmaa, jolloin uusien toiminnallisuuksien käyttöönotto olisi kohtuullisen yksinkertainen prosessi. (Microsoft Visual Studio 25.7.2015).

Kommunikointia voisi myös edistää Slack-sovelluksen käyttö. Slack on pikaviestisovellus, jonne voi luoda useita eri keskusteluhaaroja. Ketteryyshaastattelun asiantuntija kertoi, että hänen tietotekniikkayrityksissä työskentelevät kaverinsa käyttävät päivittäin Slack-sovellusta, joka on helpottanut kommunikointia työntekijöiden välillä. Slackin avulla voitaisiin

tehdä esimerkiksi ryhmiä kehitystiimeille, joissa puhuttaisiin erikseen yhden sprintin töistä. Tätä kautta oleellinen tieto olisi aina helposti löydettävissä ja viestit pystyttäisiin lukemaan kaikilla päätelaitteilla missä tahansa. (Gupta 1.7.2018).

Kiireellisissä tilanteissa Slackin kautta voitaisiin pitää myös päiväpalaveri. Osaston työtekijät työskentelevät liukuvasti ja eri aikavyöhykkeillä, joten päiväpalaverin pitäminen on haastavaa pitää aina samaan aikaan. Slackin avulla jokainen kehitystiimin jäsen kertoo suurin piirtein samaan aikaan päiväpalaverin osuutensa, jolloin muut jäsenet pysyvät ajan tasalla muiden työntekijöiden työstä ja heidän ei tarvitse olla fyysisesti paikalla. Kehitystiimin jäsenet voivat myös huomata tarvitsevansa toistensa tukea tehtävissä, jolloin Scrummaster voi tagien avulla ilmaista kahdenkeskisen työn tarpeen. (Gupta 1.7.2018).

Osaston johdon priorisointi ja päätösten läpinäkyvyys on myös haastattelujen perusteella haasteellista. Yksi tapa vastata tähän ongelmaan on omaksua johdon keskuudessa ketteriä käytäntöjä. Ketterät käytännöt voisivat lisätä osaston johdon tietämystä ketteristä menetelmistä ja osaston työntekijät voisivat luottaa osaston toimintamalliin enemmän, kun koko osasto hyödyntäisi ketteryyttä. Osaston johto voisi alkaa pitämään omia päiväpalavereja. Osaston johto voisi joka päivä pitää alle viidentoista minuutin palaverin, jossa jokainen kävisi yhdessä tai kahdessa minuutissa läpi mitä teki eilen, mitä aikoo tehdä tänään ja mitä esteitä työhöni saattaa tulla. Työn esteet voivat varsinkin luoda syvempää keskustelua, jolloin niistä voidaan siirtää työtaululle muistilappu, josta keskustellaan tarkemmin myöhemmin. Tämä vahvistaa johdon keskinäistä kommunikaatiota ja mahdollistaa nopeamman ongelmanratkaisukäytännön. (Leffingwell 2007, 302).

Osaston johto voisi kokeilla muuttaa kaikki aloitteet, ketteryyttä haittaavat esteet ja muut työtehtävät omaksi kehitysjonoksi, jota hallinnoitaisiin ja priorisoitaisiin samalla tavalla kuin kehitettävän tuotteen kehitysjonoa. Jokaiselle kehitysjonon kohdalle asetetaan prioriteetti ja vastuuhenkilö. Tätä kautta osaston johto voisi läpinäkyvämmin kertoa tekemisestään varsinaisille kehitystiimeille ja luottamus johdon päätöksiin voisi kasvaa. Osaston johdon olisi myös tärkeä priorisoinnin lisäksi hakea heti ratkaisuja kehitysjonon tärkeimpiin kohtiin, koska ilman niiden ratkaisua osaston muun toiminnan tehokkuus voi kärsiä. Esimerkiksi jos osasto on muuttanut toimintatapaansa Scrumin iteraatiomalliin, mutta asiakkaille tapahtuva myynti toimii kiinteähintaisena ja kiinteillä resursseilla, osaston ketteryydellä ei ole merkitystä. (Leffingwell 2007, 302).

Osaston johdon voi olla vaikea toteuttaa järkevästi kehitysjonon kohtia epämääräisellä aikataululla, joten näitä kohtia voitaisiin työstää myös iteraatiomallin mukaan. Jos osasto työskentelee normaalisti esimerkiksi kahden viikon iteraatioissa, osaston johto voisi ottaa

tavoitteekseen saada noin kaksi tai kolme kehitysjonon kohtaa valmiiksi jokaisessa kahden viikon iteraatiossa. Iteraation päätteeksi johdon edistymisen arvioidaan ja hyväksytään ja johto aloittaa uusien kohtien työstämisen seuraavassa iteraatiossa. Iteraatiomallin päätavoite on saada osaston johto työstämään ja saavuttamaan tavoitteitaan helpommin ja näyttämään esimerkkiä muulle osastolle ketterän organisaation toimintatavoista. (Leffingwell 2007, 303).

Osaston mittaaminen keskittyy pääsääntöisesti työn etenemiseen ja muiden sovellusten antamiin valmiisiin mittareihin. Osaston pitää jatkossa keskittyä mittaamisessa toimivaan ohjelmistoon ja sen toimivuuteen sille tarkoitetussa ympäristössä. Osaston pitää ottaa myös ottaa iteraatiomittarit ja julkaisumittarit käyttöön projektien sisällä. (Leffingwell 2007, 312).

Iteraatiomittareissa osasto mittaa sprinttitasolla tapahtuvia mittauskohteita. Iteraatiomittarit ovat esimerkiksi käyttäjätarinoiden tehtyä määrää ja kehitystiimin tarinapisteiden toteutumismäärää. Julkaisumittarit taas keskittyvät enemmän asiakkaalle tuotettuun arvoon hitaammalla aikavälillä. Julkaisumittareita voi esimerkiksi olla asiakastyytyväisyys ja epäonnistuneiden versioiden määrä projektin aikana. (Leffingwell 2007, 312-313).

Kun osasto ottaa ketteryyttä lisääviä keinoja käyttöönsä, sen kannattaa siirtyä kuvan kolme tyyppiseen ketterään tasapainotettuun mittaristoon. Osaston kannattaa mitata työn tehokkuutta, työn laatua, arvontuottoa ja ketteryyttä omina kokonaisuuksinaan. Tehokkuutta osasto voi mitata esimerkiksi tarkastelemalla kehitystiimin nopeutta tuottaa arvoa asiakkaalle ottaen kehitystiimin kapasiteetin huomioon. Osasto voi siis mittarin avulla arvioida paremmin työmäärää riippuen siitä, onko kehitystiimissä kokeneita työntekijöitä vai paljon aloittelevia työntekijöitä. (Leffingwell 2007, 320).

Työn laatua osasto pyrkii mittamaan asiakkaan palautteen kautta. Työn laatumittareita voivat olla tuotetyytyväisyys ja tukipyyntöjen määrä. (Leffingwell 2007, 320-321).

Arvontuottoa osasto ei haastattelun mukaan mittaa, koska sitä käytetään yleensä ainoastaan ketterässä sovelluskehityksessä. Arvontuottomittareita ovat esimerkiksi julkaistujen versioiden määrä ja arkkitehtuurin optimointi helposti ylläpidettävän muotoon, joiden avulla osasto pystyy seuraamaan tuotteen toimitusten määriä paremmin. (Leffingwell 2007, 321-322).

Ketterät mittarit keskittyvät taas kehitystiimin itsearviointiin. Ketterien mittarien tarkoitus on mitata osaston kykyä vastata tulevaisuuden suorituskykytavoitteisiin. Taulukko yksi on esimerkki kehitystiimin itsearvioinnin mittaamisesta. Itsearviointi tapahtuu niin, että jokaisessa kvartaalissa kehitystiimi vastaa jokaisesta ketteryysalueesta muutamaan väittämään, johon annetaan arvosanaksi jotain nollan ja viiden välillä. Tämän jälkeen ketteryysalueen yhteispisteet jaetaan maksimipisteillä, jolloin saadaan jokaisen kehitystiimin suhteellinen ketteryys jokaisesta ketteryysalueesta. Kun kaikki ketteryysalueet on laskettu, niin niiden summa voidaan jakaa maksimipisteillä, josta saadaan kehitystiimin ketteryyden prosenttiosuus. Tätä kautta osasto pystyy löytämään ketteryyden kipupisteet ja kehitystiimi pääsee helposti pohtimaan syitä ja jatkotoimenpiteitä jokaiselle ketteryysalueelle. (Leffingwell 2007, 312-317).

Ketteryysalue	%
Tuotteen omistajuus/hallitsemiskapasiteetti	**55 %**
Julkaisujen suunnittelun ja seurannan kyvykkyys	**72 %**
Iteraatioiden suunnittelu ja seuranta	**80 %**
Tiimin tuottavuus	**58 %**
Testauskäytännöt	**40 %**
Kehitystyökäytännöt/infrastruktuuri	**67 %**
Tiimin pistemäärä:	**120**
Ketteryyden prosenttiosuus	**65 %**

Taulukko 1. Ketterän prosessiohjautuvuuden mittaaminen (mukaillen Leffingwell 2007, 313-316)

4.2 Edellytykset ketterien keinojen onnistuneeseen sisällyttämiseen

Kun osastolle aletaan sisällyttää ketteryyttä lisääviä keinoja, on tärkeää huomioida keinojen tuovan ison muutoksen osaston sovelluskehitykseen. Osasto on toiminut pitkään vesiputousmallin mukaan, jolloin ketterän toimintamallin omaksuminen ei ole helppoa. Osastolta vaaditaan siis hyvää muutoksenhallintastrategiaa, jotta ketterän toimintamallin sisällyttäminen onnistuu.

Ketteryyden sisällyttämisessä pitää ottaa huomioon ketterän ajatusmallin luominen. Jotkut yritykset pystyvät ottamaan helposti ketterän sovelluskehitysmallin käyttöönsä, koska heiltä löytyy jo valmiiksi ketterä ajatusmalli. Osasto ei kuitenkaan omaa ketterää ajatusmallia, joka johtuu pitkälti osaston yrityskulttuurista. Osaston pitääkin tämän takia panostaa jokaisen työntekijän koulutukseen, jotta ketterät menetelmät eivät tunnu liian vaikeilta ottaa käyttöön. (Stellman & Greene 2015, 369).

Osastolla työskentelevien työntekijöiden keski-ikä on noin 50-vuotta, jolloin uuden toimintatavan sisällyttäminen on haastavampaa kuin keskimäärin nuoremman yrityksen toimintatapojen muuttaminen. Osaston monet työntekijät ovat työskennelleet suurimman osan, ellei jopa koko työuransa osastolla, jolloin heille on voinut pinttyä ainoastaan yksi tapa kehittää ohjelmistoja. Osastolla on myös paljon ketteryydestä tietäviä ja osaavia asiantuntijoita, mutta on myös paljon työntekijöitä, jotka eivät ole kuulleetkaan ketterästä sovelluskehityksestä. Tästä syystä osasto tarvitsee onnistuneeseen ketterien keinojen sisällyttämiseen Agile Coach eli ketteryysvalmentajan. (Stellman & Greene 2015, 370).

Ketteryysvalmentajan suositellaan tulevan osaston ulkopuolelta ja hän auttaa tiimiä sopeutumaan ketteryyteen. Ketteryysvalmentaja keskustelee jokaisen osaston työntekijän kanssa ja auttaa heitä pääsemään teknisen ja henkisen muurin yli, koska roolituksessa tapahtuvat muutokset ja tiimin itseohjautuvuus voivat olla raskaita asioita käsitellä osaston työntekijöiden keskuudessa. Osaston yrityskulttuurin takia muutosvastarinta uutta toimintatapaa vastaan voi olla suurta ilman huolellista suunnittelua. (Stellman & Greene 2015, 370).

Yksi tärkeimmistä tavoista selvittää osaston ketteryys ja samalla hallita osaston oppimista on kamppailulajeista tuttu oppimiskaava, jonka nimi on shuhari. Shuharin idea on se, että oppiminen tapahtuu kolmessa eri vaiheessa. Ensimmäinen vaihe on shu eli totteleminen ja tarkasteleminen, toinen vaihe on ha eli irtautuminen ja kolmas vaihe on ri eli erkaantunut. (Stellman & Greene 2015, 377).

Ensimmäisessä vaiheessa kuuluu noudattaa annettua viitekehystä täsmällisesti. Osasto voi siis ensimmäisen vaiheen mukaan ottaa Scrum ensin rautalankamaisesti käyttöön eli kaikkia tapahtumia ja rooleja noudatetaan juuri viitekehyksen teorian mukaan. Tämän avulla osaston jokainen työntekijä alkaa ymmärtää ketterän sovelluskehityksen periaatteet ja käytännöt ja heidän on helppo totutella uuteen malliin. Aikuisille ihmisille oppiminen on yleensä paljon työläämpää kuin esimerkiksi lapsille, jolloin yksinkertainen malli antaa hyvän pohjan oppimisen aloittamiselle. (Stellman & Greene 2015, 377).

Toisessa vaiheessa oppijat ymmärtävät jo viitekehyksen käytännöt ja niitä on toistettu käytännössä. Osaston työntekijät ovat siis tässä kohtaa jo tottuneet ketterän viitekehyksen toteuttamiseen ja ovat alkaneet huomata sen teorian täsmällisen käytön haittoja. Tässä kohtaa onkin hyvä aloittaa ketterän viitekehyksen räätälöiminen niin, että se vastaa osaston tarpeita. Kun osaston työntekijät alkavat ymmärtää taustan ketterien menetelmien käyttöön, he alkavat huomata parannuksia uuden mallin toteuttamisessa. Toisen vaiheen oppijat ovat siis alkaneet omaksua ketterän ajatusmallin. (Stellman & Greene 2015, 378).

Kolmannessa vaiheessa kaikki ovat sujuvia viitekehyksen käytännöissä, ideoissa ja arvoissa. Osasto ei tässä kohtaa enää välttämättä tarvitse viitekehystä, vaan voi suhteuttaa jokaiselle tilanteelle luontevan ketterän ratkaisun. Kun osasto pääsee kolmannelle tasolle oppimisessa, se on omaksunut täydellisen ketterän ajatusmallin. (Stellman & Greene 2015, 378).

Kun osasto ottaa ketteryysvalmentajan käyttöön ja aloittaa ketterän sovelluskehityksen sisällyttämisen, osasto voi kokea vahvaa muutosvastarintaa. Osastolla voi ilmetä esimerkiksi vastaväitteitä siitä, että sovelluskehitys toimii jo hyvin, joten miksi muuttaa jotain, joka ei ole rikki. Tämä voi johtua siitä, että monien vuosien kokemus lukuisista projekteista saa jotkut työntekijät puolustuskannalle omasta työstään. Jotkut työntekijät eivät myöskään ole huomanneet suuremassa mittakaavassa osaston haasteita, jolloin on elintärkeää pyrkiä ensin osoittamaan käytännössä nykyisen sovelluskehitysmallin haasteet ja sen jälkeen selittämään, miten Scrum tai muu ketterä viitekehys vastaa siihen. (Stellman & Greene 2015, 371-372).

Toinen yleinen vastaväite ketterälle sovelluskehitykselle on se, että se on liian riskialtis. Vesiputousmallissa on yleensä niin hyvät riskivalmiudet ja häilyvät virstanpylväät toimituksissa, että toteutustyön voi tehdä suhteellisen rauhallisesti. Ketterä toimintatapa on taas läpinäkyvä ja toimii lyhyemmissä sykleissä, jolloin pienikin virhe näkyy heti kaikille projektin osapuolille. Jotta tältä ajatusmaailmalta vältytään, osaston työntekijöille pitää painottaa, että virheiden tekeminen on hyväksyttävää, kunhan niistä oppii. (Stellman & Greene 2015, 372-373).

Kokeneiden asiantuntijoiden suusta voi kuulla myös vastaväitteen siitä, että tämä käytäntö ei tule toimimaan minun kanssani. Esimerkiksi ohjelmistokehittäjä, joka on tottunut ajattelemaan ohjelmavirheet hyväksymiskelvottomana työnä, voi kokea olonsa epämiellyttäväksi pariohjelmoinnin yhteydessä. Osaston on siis tärkeää aloittaa tällaisten työntekijöiden kanssa tunnetuimpien ja heitä eniten hyödyttävien käytäntöjen kanssa. Kun tällaiset työntekijät alkavat ymmärtää käytäntöjen hyödyt, alkavat he omaksua paremmin myös muita käytäntöjä. (Stellman & Greene 2015, 373).

Yleisimpiä vastaväitteitä on myös se, että ketteryys ei sovi meidän liiketoimintaamme. Tämä väite on yleistä isoissa yrityksissä, koska yrityksien työntekijät ovat tottuneet rakentamaan yksityiskohtaisia suunnitelmia ennen työn aloittamista ja he eivät voi kuvitellakaan työskentelevänsä millään muulla tavalla. Työntekijät saattavat perustella väitettään niin,

että ketteryys ei sovi heidän monimutkaiseen liiketoimintaansa, koska se vaatii aluksi hyvää suunnittelua. Yleensä kuitenkin kaikkien yritysten liiketoiminta on monimutkaista ja suurien kokonaisuuksien pilkkominen pieniin osiin mahdollistaa monimutkaisenkin tuotteen rakentamisen ketterästi. (Stellman & Greene 2015, 374-375).

Viimeinen yleinen vastaväite on se, että meidän sovelluskehityksemme toimii jo näin, mutta ainoastaan toisella nimellä. Väitteellä tarkoitetaan sitä, että yrityksillä voi olla näennäisesti ketteriä käytäntöjä käytössään, mutta oikeasti ne ovat naamioituja vesiputousmallin mukaisia käytäntöjä. (Stellman & Greene 2015, 375). Osastolla on esimerkiksi käytössä Scrum-viikkopalaveri, joka on kuitenkin vesiputousmallille tyypillinen katsaus projektin tilasta. Kun ketterää mallia lähdetään viemään eteenpäin, osasto voi pitää juuri näiden palaverien takia ketteryyttä turhana, koska he ovat kuvitelleet Scrum-palaverien olevan ketteriä palavereja. Tähän osaston pitää panostaa niin, että työntekijät ymmärtävät oikeiden ketterien käytäntöjen hyödyn. (Stellman & Greene 2015, 375-376).

Keinojen sisällyttämisen haasteena on myös kehitystiimien jako. Nykytilahaastattelun mukaan osastolla on useita asiakasprojekteja samanaikaisesti, jolloin osasto tarvitsee jokaiseen asiakasprojektiin vähintään yhden kehitystiimin. Lisäksi osastolla on ollut projekteja, joissa työntekijöiden määrä lähenee 80 työntekijää. Ketteryyshaastattelun asiantuntijoiden mukaan osasto ei siis pysty myöskään pitämään yhtä tiimiä yhdessä asiakasprojektissa, vaan sen on luotava useita eri kehitystiimejä.

Ketteryyshaastattelun asiantuntijat olivat sitä mieltä, että kehitystiimit pitäisi jakaa niin, että jokaisessa tiimissä on kokeneita osaajia ja uusia tulokkaita. Tiimit eivät saisi olla komponenttisidonnaisia eli jokaisen tiimin pitäisi yhteistoimin pystyä tuottamaan toimintoja vähintään selainkäyttösovellukselle ja työasemasovellukselle. Kehitystiimien pitää olla myös jatkossa vähintään projektiin sitoutuneita eli kehitystiimin kokoonpano ei vaihtuisi ainakaan kesken iteraatiota tai edes projektia. Asiantuntijoiden mielestä osaston pitää myös antaa kehitystiimille edellytykset toteuttaa projektit ketterästi. Tällä asiantuntijat tarkoittavat sitä, että osasto ei voi esimerkiksi jatkaa hajanaista työntekijöiden resurssointia kehitystiimin yhteydessä, vaan kun kehitystiimi otetaan käyttöön, se on täysin sitoutunut projektiin koko projektin ajalle.

Jotta kehitystiimit eivät tekisi päällekkäistä työtä, heidän olisi hyvä ottaa käyttöön Scrumin Scrum. Scrumin Scrumilla tarkoitetaan päiväpalaveria, johon osallistuu jokaisesta kehitystiimistä yksi lähettiläs. Lähettiläs on joku kehitystiimin työstä vastaava Scrummaster, joka valitaan yhteisesti päiväpalaverissa Scrum of Scrumin palaveriin. Palaverissa lähettiläät

keskustelevat suoritetusta työstään, tulevista töistä ja mahdollisista esteistä heidän työskentelyssään. Nämä kaikki kirjataan yhteiseen kehitysjonoon, jota päivitetään jokaisen palaverin yhteydessä. (Leffingwell 2007, 150-151).

Scrumin Scrumin lisäksi olisi hyvä perustaa ohjausryhmä, joka tapaa kerran viikossa. Ohjausryhmään kuuluu kaikki Scrummasterit ja usein myös ohjelmistoarkkitehdit, osastopäälliköt, kokeneet laatujohtajat ja muut yrityksen johdon keskeiset työntekijät. Ohjausryhmän tapaamisessa keskustellaan nykyisen inkrementin tilasta, kehitystiimien tavoitteiden tilasta ja iteraatioaikataulusta. Ohjausryhmä kommunikoi organisaation laajuisesti mittarien etenemistä, mutta jättää palautteen antamisen Scrummasterien tehtäväksi. Scrumin Scrumin avulla ohjausryhmä voi helpottaa kehitystiimin työtä ratkomalla ulkoisten sidosryhmien ongelmia, mutta ohjausryhmä ei edelläänkään saa puuttua kehitystiimin tapaan toteuttaa työnsä. (Leffingwell 2007, 152-153).

Osaston suuri koko tekee myös käyttötapadokumenttien vaihtamisen käyttäjätarinoihin hankalaksi. Kun käyttäjätarinoita aletaan käyttää, voi olla hyvä tapa jakaa ne myös toiminnallisuuksiin ja suurempiin kokonaisuuksiin, joita kutsutaan nimellä epic. Osaston komponentit ovat niin suuria kokonaisuuksia, että niitä on vaikea kuvata täysin pelkkien käyttäjätarinoiden ja toiminnallisuuksien avulla. Taulukko kaksi esittää esimerkkijaon, miten osastolla kaikki työtehtävät voitaisiin jakaa. (Microsoft Visual Studio 25.7.2015).

Työtehtävän tyyppi	Otsikko
EPIC	Myyntiohjelmisto
Ominaisuus	Puhelin
Käyttäjätarina	lisää tietolomake
työtehtävä	automaattinen tietojentäydennys tietolomakkeeseen
työtehtävä	tietolomakkeen automaattitallennus

Taulukko 2. Työtehtävien jako suurempien järjestelmien yhteydessä (mukaillen Microsoft Visual Studio 25.7.2015)

Epic koostuvat siis yleensä isoista kokonaisuuksista suurissa järjestelmissä ja osasto voisi itse määritellä esimerkiksi työasemasovelluksensa ja selainkäyttösovelluksensa tällaiseksi johtuen niiden suuresta koosta. Näiden alla taas voisi olla ominaisuuksia, jotka jaetaan käyttäjätarinoihin. Käyttäjätarinat voidaan sitten jakaa sprinteissä edelleen työtehtäviksi. (Microsoft Visual Studio 25.7.2015).

Osaston johdon olisi hyvä jatkossa keskittyä esteiden havaitsemiseen ja korjaamiseen. Osaston on mahdotonta huomata heti kaikkia esteitä, koska monet esteet ovat juurtuneet osaston toimintaan erittäin syvälle ja ne tuntuvat liian tutuilta. Scrumin käyttöönotto tuo vasta oikeasti näkyviin syvälle juurtuneet esteet, koska silloin kaikkea toimintaa tarkastellaan kriittisemmin. Esteitä esiintyy tyypillisesti neljässä eri osa-alueessa, jotka ovat:

1. Itse Scrum prosessissa esiintyvät esteet.
2. Ihmisten käytännöt, jotka haittaavat tuotteen kehitystä, jakelua ja tukea.
3. Tuotteen rakentamiskäytännöt, jotka vaikuttavat osaston mission ja pääoman tuottoasteen optimointiin.
4. Organisaation systemaattiset haasteet, jotka estävät tuotteen toimituksen maksimoinnin.

(Lönnqvist 2006, 298-299).

Osaston johdon pitää myös ottaa vastuun lisäksi muutoksen johtaminen omille harteilleen. Osaston johdon pitää painottaa kaikille osaston työntekijöille, että sovelluskehitykseen on tullut uusi, ketteriin käytöntöihin pohjautuva suunta. Osaston johdolla on useita tapoja miettiä muutoksen johtamista, mutta yleisimpiä tapoja muutosjohtamiseen ovat:

1. Osaston johto tai osa johdosta ottaa henkilökohtaisen roolin ketterän polun opiskelemisessa ja valmentamisessa koko johtotiimille. Johdon kanssa aletaan heti työstää rohkeaa suunnitelmaa, joka toteutetaan kokonaisuudessaan.
2. Hanki vierasluennoitsijoita tai konsultteja muutokseen valmistumista varten, jotka ovat ammattilaisia ketteryydessä ja organisaatiomuutoksissa.
3. Valtuuta tiimijohtajia ja muita osaavia ja kokeneita asiantuntijoita auttamaan johtoa sisällyttämään ketterää toimintamallia ja luomaan optimaalisia kehitystiimejä.

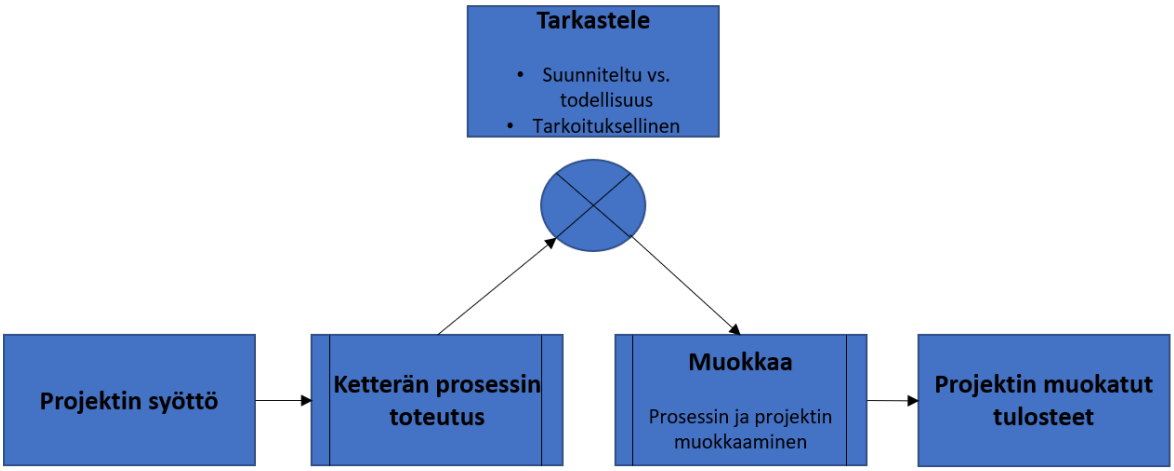
(Lönnqvist 2006, 301).

Asiakkaan saaminen ketterään malliin tulee olemaan haastavaa. Asiakkaalle on yleensä tärkeää ymmärtää projektin budjetti, jotta he voivat suhteuttaa projektin hinnan omaan liiketoimintaansa. Ketterässä mallissa ei kuitenkaan suositella esimerkiksi kiinteää projektihintaa, jolloin asiakkaalle joutuu perustelemaan uuden toimintatavan eri tavalla. Osaston kannattaa jatkossa keskittyä asiakastapaamisissa muutoksen esittelemiseen, iteratiiviseen sovelluskehitykseen, jatkuviin palautesykleihin, parempaan arvon tuottamiseen ja ohjelmistotuotannon käytäntöjen yksinkertaistamiseen. (Verwijs 29.4.2012).

Muutoksenhallinta ja iteratiivinen kehitys ovat tärkeitä asiakkaalle painotettavia asioita. Kun kehitystyö voidaan aloittaa lähes heti, asiakas saa paljon nopeammalla aikataululla valmiin tuotteen. Asiakkaan vaatimukset ovat tyypillisesti aina toisenlaisia projektin lopussa, jolloin ketterän mallin muutoksenhallinta tarjoaa asiakkaalle heidän oikeisiin tarpeisiinsa pohjautuvan tuotteen. (Verwijs 29.4.2012).

Kun asiakas pääsee antamaan jatkuvasti palautetta tuotteen väliversioista, heillä on parempi mahdollisuus vaikuttaa tuotteen ominaisuuksiin. Jos asiakas näkee tuotteen vasta projektin lopussa, ei palautteen näkökulmia voida enää ottaa samalla tavalla huomioon. Asiakas pääsee myös jatkuvasti seuraamaan tuotteen toteutuksen etenemistä, jolloin he pystyvät näyttämään omalle johdolleen konkreettisia, arvoa tuottavia tuloksia. (Verwijs 29.4.2012).

Ohjelmistokehityksen käytäntöjen yksinkertaistaminen edesauttaa myös asiakkaan ja osaston välistä kommunikaatiota ja tiedonvaihtoa. Kun asiakas osallistuu kehitystyöhön kehitystiimin kanssa, asiakas ymmärtää paremmin oikean tarpeensa ja sovelluskehityksen tuomat haasteet. Asiakas ei ole aina tietotekniikan ammattilainen, joten läpinäkyvä kommunikaatio on elintärkeää tuotteen onnistumisen takaamiseksi. (Verwijs 29.4.2012).



Kuva 6. Ketterän mallin empiirinen prosessinhallinta (mukaillen Leffingwell 2007, 292)

Osasto ei voi myöskään jättää ketterien keinojen sisällyttämistä Scrum-viitekehyksen käytäntöihin. Sovelluskehitys on alanaan jatkuvasti muuttuva ja monimutkainen. Sovelluskehityksessä ei luoda pyörää uudelleen ja uudelleen, vaan jatkuvasti kehitetään uusia asioita uusille toimialoille. Kehitettävä sovellus ei välttämättä ole koskaan ollut kenenkään muun kehitteillä, jolloin yksinkertainen suunnitelma ei riitä parhaiden käytäntöjen saavuttamiseksi. Siksi kuva kuusi esittää empiiristä eli havaintoihin perustuvaa prosessinhallintaa, jossa prosessit ovat jatkuvassa kehityksessä. Ensin projekti ja sen prosessit toteutetaan ja tarkastellaan, jonka jälkeen havaittuja muutoksia muokataan ja viedään seuraavissa projekteissa käytäntöön. Osaston on siis ehdottoman tärkeää ymmärtää, että vaikka he saavuttaisivatkin erinomaisia tuloksia Scrumin sisällyttämisessä, uusien ketterien keinojen sisällyttämistä pitää jatkaa aina. (Leffingwell 2007, 291-292).

5 Pohdinta

Ketterä sovelluskehitys on samaan aikaan hyvin yksinkertaista ja kovin monimutkaista. Sprinttien ymmärtäminen on todella helppoa, koska ne noudattavat aina samoja käytäntöjä suunnittelusta retrospektiiviin asti. Muutokseen vastaaminen on myös hyvin ymmärrettävä prosessi, koska yritykset ovat hyvin monimuotoisia ja yritysten tavoitteet muuttuvat jatkuvasti erilaisten intressien kautta. Ketteryydestä tulee kuitenkin heti monimutkaista, kun sitä lähdetään käyttämään yritysten liiketoiminnassa. Mitä isompi yritys on, sitä monimutkaisempi ketterä malli tarvitaan sen tarpeiden tyydyttämiseen.

Yrityksen osastolle löydetyt ketterät keinot vastasivat hyvin teoriaosuuden lähteitä. Osastolla on selkeästi vielä skeptinen näkökulma ketterään sovelluskehitykseen, joka on täysin ymmärrettävää. Monet osaston työntekijät ovat olleet osaston palveluksessa kymmeniä vuosia ja he eivät ole välttämättä koskaan kuulleet muusta tavasta rakentaa ohjelmistoja. Monella työntekijällä on yhä ajatus siitä, että ketterä malli sopii ainoastaan pienten yritysten toimintaan ja yksinkertaisten ohjelmistojen rakentamiseen.

Opinnäytetyössä esitetyt ketterät keinot ovat varmasti hyvä aloituspohja osaston ketteryyden kehittämiseen. Yrityksen osasto voi hyvin aloittaa käyttämään Scrum-viitekehystä, mutta sen lisäksi osaston kannattaa ottaa periaatteita myös muista ketteristä viitekehyksistä. Osaston kannattaa miettiä tarkkaan avainhenkilöt ketterän mallin sisällyttämiseen, koska muutos uuteen malliin ei tule olemaan helppoa. Jatkon kannalta osasto ei voi pitkällä aikavälillä jäädä käyttämään Scrumia, vaan sen on omaksuttava edistyneempi ketterä malli, joka vastaa parhaalla tavalla sen liiketoiminnan haasteisiin.

Yrityksen osaston ketterä malli voisi tulevaisuudessa olla Scaled Agile Framework eli SAFe. SAFe on toimiva viitekehys osaston kaltaiselle isolle yritykselle, joka auttaa tuottavuuden ja laadun kehittämisessä ja työntekijöiden sitouttamisessa. SAFe pyrkii jatkuvaan ja ennustettavaan tuotteen toimittamiseen ja se perustuu Lean-viitekehyksen pääperiaatteisiin. (Scaled Agile 2019).

Opinnäytetyö toimii hyvänä ohjeena suurille ja keskisuurille yrityksille, jotka hakevat keinoja muuttaa toimintaansa ketterämpään suuntaan. Nykytilahaastattelun perusteella osasto toimii juuri samalla tavalla kuin vesiputousmallin teoria osoittaa. Vaikka opinnäytetyö keskittyy pitkälti osaston haasteiden tunnistamiseen, on osasto myös onnistunut monessa asiassa. Osastolla työntekijät tulevat hyvin toimeen keskenään ja varsinkin osaston testaustiimiä kehuttiin hyvästä työstä. Osastolla on myös todella kokeneita asiantuntijoita,

joilla on laaja ammattitaito ja kyky omaksua uutta tietoa. Osaston siis kannattaa jatkossakin keskittyä haasteiden ratkaisemisen lisäksi ylläpitämään näitä pitkällä aikavälillä hankittuja vahvuuksiaan.

Opinnäytetyö opetti minulle paljon uutta ketterästä sovelluskehityksestä. Itselleni tuli yllätyksenä, kuinka paljon eri käytäntöjä Scrumiin kuuluu, koska Scrum on viitekehyksenä muuten todella yksinkertainen. Scrum on myös muuttunut paljon, koska moni Scrumille tyypillinen käytäntö ei kuulukaan varsinaisen viitekehyksen pariin. Esimerkiksi aikaisemmin kuvittelin päiväpalavereissa seisomisen kuuluvan Scrumin viitekehykseen, mutta kirjaa lukiessani huomasin, että seisominen ei kuulu perinteiseen Scrumiin ollenkaan.

Opinnäytetyössä oli myös paljon haasteita. Työni rajaaminen oli paljon vaikeampaa kuin kuvittelin, vaikka rajasinkin työni koskemaan yhtä ketterää viitekehystä. Lähdemateriaalia on ketterille menetelmille niin paljon, että oleellisen tiedon erottaminen toissijaisesta tiedosta oli välillä erittäin haastavaa. Esimerkiksi Scrumin kritiikin löytäminen oli välillä turhauttavaa, koska moneen kritiikkiin Scrumin asiantuntijat olivat perustelleet kritiikin olevan sivuvaikutus huonosta Scrumin sisällyttämisestä.

Haastattelut olivat myös haasteellisia. Haastattelujen määrä oli suhteellisen pieni, mutta sain mielestäni kuitenkin tarvittavasti tietoa kokonaisuuden hahmottamiseen. Kun olin suorittanut osastolla kaikki haastattelut niin osastolle oli juuri tullut uusia Agile-sertifikaatin omaavia asiantuntijoita. Jos olisin haastatellut myös heitä, niin olisin varmasti saanut vielä enemmän keinoja osaston ketteryyden lisäämiseen. Tiukan aikataulun takia en kuitenkaan ehtinyt enää haastatella heitä, jolloin jouduin pitäytymään kuudessa haastateltavassa. Toimeksiantajan kanssa käyty katselmointi ei myöskään onnistunut projektisuunnitelman mukaisena aikana, koska sekä itselläni että toimeksiantajalla oli paljon kiireitä. Saimme kuitenkin noin kolmen viikon välein katselmoitua opinnäytetyötä, jolloin sain mielestäni riittävästi palautetta.

Kaikista eniten opinnäytetyön teossa opin yleisestä ketteryydestä ja mittaamisesta. Ketteryyden ei tarvitse olla jonkun ketterän viitekehyksen noudattamista, vaan se on yksinkertaisesti muutokseen vastaamista tilanteeseen optimaalisella tavalla. Vaikka ketteryyttä on hyvä toteuttaa tiettyjen käytäntöjen kautta, niin ilman ketterää ajatusmallia käytännöillä ei ole väliä. Ketterä mittaaminen osoittaa myös sen, että parhaat mittaamistulokset saadaan nimenmaan panostamalla arvon tuottamiseen ja toimiviin ratkaisuihin. Jos asiakas ei ole tyytyväinen lopputulokseen, niin silloin tilanteesta pitää oppia ja kehittyä sellaiseen suuntaan, että jatkossa samoja virheitä ei toisteta uudelleen. Kun asiakkaan tarve tulee muuttumaan, niin toimittajan pitää pystyä myös sopeutumaan.

Lähteet

Agile Allience 2019. Agile 101. Luettavissa: https://www.agilealliance.org/agile101/. Luettu: 12.3.2019.

Agilemanifesto 2001a. Ketterän ohjelmistokehityksen julistus. Luettavissa: https://agilemanifesto.org/iso/fi/manifesto.html. Luettu: 20.3.2019.

Agilemanifesto 2001b. Julistuksen takana olevat periaatteet. Luettavissa: https://agilemanifesto.org/iso/fi/principles.html. Luettu: 20.3.2019.

Ahonen, M. 13.9.2015. Mikael Ahonen – Gantt-kaavio Excelissä. Luettavissa: https://mikaelahonen.com/fi/blog/gantt-kaavio-excelissa/. Luettu: 10.4.2019.

Chi, C. 31.10.2018. Everything You Need to Know About Using the Waterfall Methodology. Luettavissa: https://blog.hubspot.com/marketing/waterfall-methodology. Luettu: 11.4.2019.

Cho, J. 2008. Issues and challenges of agile software development with scrum. Luettavissa: http://iacis.org/iis/2008/S2008_950.pdf. Luettu: 14.4.2019.

Gupta, S. 1.7.2018. What if we do daily scrum/stand up over Slack. Luettavissa: https://hackernoon.com/what-if-we-do-daily-scrum-stand-up-over-slack-59e61d72dc8b. Luettu: 8.5.2019.

Fowler, M. 9.7.2006. Writing The Agile Manifesto. Luettavissa: https://martinfowler.com/articles/agileStory.html. Luettu: 12.3.2019.

Leffingwell, D. 2007. Scaling software agility: best practises for large enterprises. Pearson Education Inc. Boston.

Lönnqvist, A, Kujansivu, P & Antikainen, R. 2006. Suorituskyvyn mittaaminen – tunnusluvut asiantuntijaorganisaation johtamisvälineenä. Edita Publishing Oy. Helsinki.

Microsoft Visual Studio. 25.7.2015. Agile Project Management in Team Foundation Server 2015. YouTube video. Katsottavissa: https://www.youtube.com/watch?v=I6OMeXu3CZc. Katsottu: 29.4.2019.

Overeem, B. 15.7.2016. 8 Best Practices to Start a Scrum Project. Luettavissa: https://www.scrum.org/resources/blog/8-best-practices-start-scrum-project. Luettu: 27.4.2019.

Parkhe, R. 15.7.2015. A handy burn-down chart excel template. Luettavissa: https://softwarenature.com/blog/agile/2015/07/15/a-handy-burndown-chart-excel-template/. Luettu: 7.5.2019.

Parmenter, D. 2010. Key performance indicators: developing, implementing and using winning KPIs. John Wiley & sons, Inc. New Jersey.

Paul, D. & Girvan, L. 2017. Agile and Business Analysis. BCS Learning & Development Ltd. Swinson.

Projectcubicle 2018. Advantages and Disadvantages of Gantt Charts. Luettavissa: https://www.projectcubicle.com/advantages-disadvantages-gantt-charts/. Luettu: 10.4.2019.

Resnick, S, Bjork, A, De la Maza, M. 2011. Professional Scrum with Team Foundation Server 2010. Wiley Publishing, Inc. Indiana.

Royce, W. 1970. Managing the development of large software systems. Luettavissa: https://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf. Luettu: 20.3.2019.

Scaled Agile. 2019. What is SAFe. Luettavissa: https://www.scaledagile.com/enterprise-solutions/what-is-safe/. Luettu: 9.5.2019.

Schwaber, K & Sutherland, J. 2017. Scrum-opas. Luettavissa: https://www.scrum-guides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Finnish.pdf. Luettu: 12.3.2019.

Stellman, A & Greene, J. 2015. Learning Agile. O'Reilly Media. Kalifornia.

Verwijs, C. 29.4.2012. How to sell your Agile approach to customers: Emphasize change. Luettavissa: https://medium.com/the-liberators/how-to-sell-your-agile-approach-to-customers-emphasize-change-729b05ba3bca. Luettu: 9.5.2019.

Wadhwa, D. 26.3.2019. TOP 3 REASONS WHY COMPANIES STRUGGLE WITH AGILE AND SCRUM. Luettavissa: https://luis-goncalves.com/why-companies-struggle-with-agile-scrum/. Luettu: 10.5.2019.

Liitteet

Liite 1. Nykytilahaastattelu

1. Kerro omista työtehtävistäsi?
2. Mitä työkaluja käytät työssäsi?
3. Mitä ketteryys merkitsee sinulle?
4. Kerro projektin elinkaaresta lyhyesti?
5. Miten työn organisointi tapahtuu?
6. Roolitus osastolla?
7. Löytyykö kehitystyöstä iteraatioita, inkrementtejä tai sprinttejä?
8. Product backlog eli tuotteen kehitysjono – miten määritellään aina seuraava kehitettävä asia?
9. Yhteistyön merkitys – luotetaanko tiimeihin ja heidän asiantuntijuuteensa, jaetaanko tietoa ahkerasti kaikille vai onko se hierarkkista? Koetaanko tiimien sisällä vastakkainasettelua?
10. Minkälaisia mittareita käytätte työssänne?
11. Kuka/ketkä työntekijät määrittelevät projektien ketteryyden?
12. Mikä on projektien palaverikäytäntö?
13. Kommunikointi – Miten tietoa jaetaan eri projektitiimin työntekijöille?

Liite 2. ketteryysasiantuntijoiden haastattelu

1. Kerro omista työtehtävistäsi?
2. Mitä työkaluja käytät työssäsi?
3. Mitä ketteryys merkitsee sinulle?
4. Mitä ongelmia näet projektin elinkaaressa yleisesti?
5. Onko työn organisoinnissa mielestäsi ongelmia?
6. Onko roolituksessa mielestäsi ongelmia?
7. Seuraavan kehitettävissä olevan vaatimuksen/use casen/taskin määrittäminen – onko tässä mielestäsi ongelmia?
8. Yhteistyön merkitys – luotetaanko tiimeihin ja heidän asiantuntijuuteensa, jaetaanko tietoa ahkerasti kaikille vai onko se hierarkkista? Koetaanko tiimien sisällä vastakkainasettelua?
9. Iteraatiot, inkrementit – mitä ongelmia osastolla löytyy niistä? pitäisikö osaston ottaa käyttöön myös sprintit?
10. Mittareiden käyttö – pitäisikö niitä mielestäsi muuttaa tai lisätä?
11. Miten projektin ketteryys tulisi mielestäsi jatkossa toteuttaa?
12. Siiloutuminen eli toimiminen erittäin asiakkuuskeskeisesti – näetkö tässä ongelmia?
13. Pitäisikö nykyisiä palaverikäytäntöjä muuttaa?
14. Pitäisikö tiedon jakamista eri työntekijöille muuttaa?