Mittaus ketterien ohjelmistotiimien suorituskyvyn ja ohjelmistokehityksen tukena

Joonas Halkilahti

Pro gradu -tutkielma
Helsinki 2.5.2020

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen osasto

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto – Fakultet/Sektion –Faculty/Section	Koulutusohjelma – Studieprogram – Study Programme
Matemaattis-luonnontieteellinen tiedekunta	Tietojenkäsittelytieteen maisteriohjelma

Tekijä – Författare – Author

Joonas Halkilahti

Työn nimi – Arbetets titel – Title

Mittaus ketterien ohjelmistotiimien suorituskyvyn ja ohjelmistokehityksen tukena

Ohjaajat – Handledare – Supervisors

Petri Kettunen

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro Gradu -tutkielma	2.5.2020	51 sivua + 3 liitesivua

Tiivistelmä – Referat – Abstract

Ohjelmistoalalle on muodostunut lukuisia metriikoita, joiden avulla ohjelmistotiimin suorituskykyä on mahdollista seurata eri sidosryhmien näkökulmasta. Ketterässä ohjelmistotuotannossa tiimien tutkimus on saanut yhä enemmän huomiota, kun muiden alojen tutkimuksen ansiosta on huomattu, että ryhmä ihmisiä pystyy yhdessä tiimityöskentelyn avulla saamaan aikaan enemmän kuin jos henkilöt työskentelisivät erillään ja työn tulokset yhdistettäisiin. Tiimityöskentelyllä on todettu olevan vaikutuksia myös ohjelmistotiimin suorituskykyyn. Suorituskykyyn vaikuttavien tekijöiden tunnistaminen on tärkeää, kun organisaatiot pyrkivät muodostamaan tiimejä, jotka suoriutuvat tehtävistään hyvin, mahdollisesti jopa kilpailijoitaan paremmin.

Tässä tutkielmassa on tutkittu sekä kirjallisuuden että tapaustutkimuksen avulla miten erilaiset metriikat voivat tukea ketteriä ohjelmistotiimejä sekä tiimien suorituskykyä. Haastattelemalla kahden eri ohjelmistokehitystiimin henkilöitä muodostui havainto, että monet jo aiemmin tunnistetut yleisesti käytetyt suorituskyvyn mittarit ovat yhä käytössä. Mittaustuloksia voidaan seurata iteraatiosta toiseen, jolloin tiimin suorituskyvyn muutosta valittujen metriikoiden valossa voidaan seurata. Tiimien käyttämien metriikoiden välillä havaittiin myös eroja, joka osittain selittyi sillä, minkälaisiin tavoitteisiin tiimin tulee pystyä vastamaan. Empiirisen tapaustutkimuksen tulosten pohjalta saatiin myös vahvistusta tutkimustiedolle siitä, että osa metriikoista palvelee vain rajattua joukkoa ohjelmistoprojektien sidosryhmistä.

Ohjelmistotiimeillä on valittavinaan useita metriikoita, joiden avulla voidaan kuvata tiimin suorituskykyä eri sidosryhmille. Tiimin suorituskykyä henkilö- sekä tiimintyöskentelytasolla kuvaavia metriikoita ei havaittu tapaustutkimuksen tuloksissa. Juuri näiden pehmeiden metriikoiden tunnistaminen ja tutkiminen kentällä olisi mielenkiintoista jatkossa, kun yritetään ymmärtää ohjelmistotiimin suorituskykyä ja siihen vaikuttavia tekijöitä kokonaisuutena.

ACM Computing Classification System (CCS):
Software and its engineering → Software creation and management → Software development process management → Software development methods
Software and its engineering → Software creation and management → Collaboration in software development → Programming teams
General and reference → Cross-computing tools and techniques

Avainsanat – Nyckelord – Keywords

Suorituskyky, ketterä ohjelmistotuotanto, metriikka, tiimi, mittaus

Säilytyspaikka – Förvaringställe – Where deposited

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Mittaus on yksi insinööritieteiden kulmakivistä ja empiirisen datan tärkeistä keräysmenetelmistä. Mittaus on kuitenkin myös hyvin arkista ja osana päivittäistä toimintaa esimerkiksi ruokakaupalla hedelmiä punnittaessa tai ajoneuvon renkaiden ilmanpainetta tarkistaessa. Mittauksen ja erilaisen tiedonkeräyksen määrä kasvaa etenkin erilaisissa digitaalisissa ympäristöissä, jossa tiedosta on tullut arvokasta valuuttaa. Erilainen tiedonkeräys on jatkuvaa ja vaikkapa verkkoselailu työmatkalla kerryttää dataa useiden toimijoiden tietovarastoihin ennennäkemättömiä määriä [GMP14, s. 24-27]. Käsitteet kuten tiedolla johtaminen levittäytyvät eri aloille pohjautuen yhä enemmän automatisoituun tiedonkeräykseen.

Myös ohjelmistoalalle on alan kehittyessä muodostunut erilaisia metriikoita, mittareita sekä mittaustapoja. Metriikalla (engl. metric) tarkoitetaan jotain mitattavaa arvoa, esimerkiksi koodin rivimäärää. Mittaaminen (engl. measurement) puolestaan on jokin toimenpide, jonka seurauksena halutulle metriikalle saadaan arvo. Edellä mainitun koodin rivimäärän tapauksessa, mittaus olisi koodin rivimäärän laskeminen. Mittari (engl. measure) esittää mitattua arvoa, metriikkaa, esimerkiksi jossakin visuaalisessa muodossa.

Ketterät ohjelmistokehitysmenetelmät kuten XP, Scrum sekä Kanban, jotka nykyään ovat vakiinnuttaneet paikkansa ohjelmistokehityksessä, tarjoavat kehitystiimeille useita keinoja ohjelmistoprojektien etenemisen seurantaan, ohjelmiston laadun ja teknisten ominaisuuksien arviointiin sekä antamaan tietoa, milloin ohjelmistokehityksen eri vaiheissa tiedossa olevat vaatimukset ja toiminnallisuudet voidaan toimittaa asiakkaalle [KMI15]. Ammattimaisen ohjelmistotuotannon yksi keskeisistä tavoitteista on tuottaa tietylle käyttäjäryhmälle vaatimukset täyttävä ja tarkoitusta palveleva ohjelmistotuote. Käyttöönotettava tuote on omalta osaltaan kuvaus, kuinka hyvin tiimin ja eri sidosryhmien välinen yhteistyö on toiminut kehitysprojektin aikana. Ketterille ohjelmistoprojekteille on tyypillistä, että vaatimukset muuttuvat projektien aikana ja tällöin myös mittareiden ja työkalujen, joita käytetään projektin edistymisen arviointiin, tulisi sopeutua nopeasti muuttuneisiin vaatimuksiin.

Tässä tutkielmassa selvitetään, miten ketterissä ohjelmistotiimeissä käytännössä seurataan suorituskykyä ohjelmistotuotannon eri vaiheissa. Suorituskyvyn tutkimus on ajankohtaista, koska ohjelmistotiimeihin kohdistuu yhä enemmän mielenkiintoa, sillä

organisaatiot ovat alkaneet ymmärtää tiimin olevan muutakin kuin vain jäsentensä summa. Erityisesti tiimityö muilta aloilta on myös ohjelmistoalaa kiinnostava ilmiö, sillä tiimityöskentelyn positiiviset vaikutukset koko tiimin suoriutumiseen ovat saaneet paljon todisteita tutkimuksen kautta.

1.1 Tutkielman tavoitteet sekä tutkimusongelma

Ketterälle ohjelmistotuotannolle on kirjallisuuden puitteissa esitetty useita metriikoita, joita ohjelmistotiimien on mahdollista seurata ja hyödyntää omassa ohjelmistokehityksessään. Tutkielman keskeinen tavoite on selvittää, käytetäänkö metriikoita ja mittausta ohjelmistokehityksen tukena haastatelluissa yrityksissä. Myös se mihin tarkoitukseen metriikoita käytetään, on tutkimuksen kohteena. Empiirisen tapaustutkimuksen avulla muodostetaan vastaus seuraaviin tutkimuskysymyksiin:

Tutkimuskysymys 1: Mitä metriikoita ketterillä ohjelmistotiimeillä ja organisaatioilla on käytössä suorituskykynsä seurantaan?

Tutkimuskysymys 2: Ketkä ohjelmistotiimissä sekä ohjelmistotiimin organisaatiossa seuraavat metriikoita ja käyttävät mittaustuloksia?

Tutkimuskysymys 3: Tukevatko käytetyt metriikat ohjelmistokehitystä ja voidaanko mittauksen avulla kehittää ohjelmistotiimin suorituskykyä tai havaita ongelmia?

1.2 Tutkielman rakenne

Seuraavassa pääluvussa 2 käydään kirjallisuuskatsauksen pohjalta läpi ketterää ohjelmistotuotantoa sekä metriikoita ja perehdytään tiimien suorituskykyyn. Luvun lopussa kootaan lyhyeen tiivistelmään kirjallisuuskatsauksen keskeiset havainnot ennen empiiristä tutkimusosiota.

Kolmannessa luvussa esitellään empiirisessä tutkimuksessa käytetty tutkimusmenetelmä sekä tutkimusaineiston keräys ja analysointi. Neljännessä luvussa käsitellään tapaustutkimuksen tuloksia tutkimusaineiston analyysin pohjalta.

Luvussa 5 tutkimustulosten ja kirjallisuuden pohjalta kootaan vastaukset esitettyihin tutkimuskysymyksiin sekä arvioidaan tutkielman tavoitteiden täyttymistä sekä tutkimustyön laatua ja luotettavuutta. Viimeiseksi luvussa 6 on tutkielman yhteenveto.

2 Mittaus ohjelmistotuotannossa

Se mitä kaikkea voidaan mitata, muuttuu jatkuvasti teknologian sekä mittausvälineiden ja tekniikoiden kehittyessä. Kokeellinen mittaus eri aloilla ei välttämättä tarkoita täysin samaa asiaa, mutta mittauksen tavoite on silti sama, antaa luotettavaa tietoa mittauksen kohteista ja ilmiöistä [Him09]. Mittauksen avulla pystytään ilmaisemaan erilaisten järjestelmien ominaisuuksia numeroiden avuin. Lordi Kelvin on kuvannut asiaa: ”*Kun voit mitata ja ilmaista numeroin sen mistä puhut, tiedä asiasta jotain. Mutta mikäli et voi mitata sitä numeroin, on tietosi aiheesta niukkaa ja epätyydyttävää*”. Numeeristen arvojen vertailu on selkeää ja mahdollistaa tulosten esittämisen tarkoilla arvoilla. Toki se mitä luvut ja mittaustulokset tarkoittavat vaatii tuntemusta aihealueesta.

Tämä luku on jaettu viiteen alalukuun, joissa käsitellään tutkielman taustan muodostavat käsitteet ja olemassa olevaa tutkimustietoa. Alaluvussa 2.1 käydään läpi ohjelmistotuotantoa ja metriikoita katsauksella siitä, miten metriikat ovat alkaneet muodostua osana ohjelmistotuotantoa. Alaluvussa myös esitellään metriikoiden käyttöä ja valintaa. Alaluvussa 2.2 käsitellään ketterää ohjelmistotuotantoa, Lean-ohjelmistotuotantoa sekä muutamia yleisesti käytettyjä ketteriä ohjelmistokehitysmenetelmiä. Alaluku 2.3 jatkaa kahden edellisen alaluvun jälkeen käsittelemällä metriikoita ja mittausta erityisesti ketterässä ohjelmistotuotannossa. Alaluvun 2.1 ja 2.3 välillä yksi selkeä ero tulee siitä, kuinka ketterässä ohjelmistotuotannossa itse ohjelmistokehitysprosessia mitataan ja seurataan monelta eri näkökulmalta. Alaluku 2.4 avaa tiimin, tiimityöskentelyn ja suorituskyvyn käsitteitä. Erilaiset metriikat ovat keino, jolla tiimin suorituskykyä voidaan pyrkiä mittaamaan ja mittaustulosten kautta tarjoamaan tiimeille työkaluja oman toimintansa kehittämiseen. Viimeiseksi alaluvussa 2.5 on lyhyt yhteenveto tämän tutkielman kirjallisuuskatsauksesta.

2.1 Ohjelmistotuotanto ja metriikat

Ohjelmistotuotanto (engl. software engineering) on tieteen silmissä kohtuullisen nuori tieteenala. Englanninkielinen nimi kuitenkin kuvaa hyvin, että kyse on insinööritieteistä, joissa mittaus on yksi tärkeä tapa erilaisten ilmiöiden tulkitsemiseen. Tieteenala on muodostunut 1960-luvun loppupuolella, kun silloisen ohjelmistotuotannon katsottiin olevan kriisissä [Wir08]. Tietokoneet olivat yleistyneet 1950-luvun lopulta ja tarve uusille ohjelmille sekä ohjelmien toiminnallisuuksille kasvoi tietokoneiden kehittyessä. Pian todettiin,

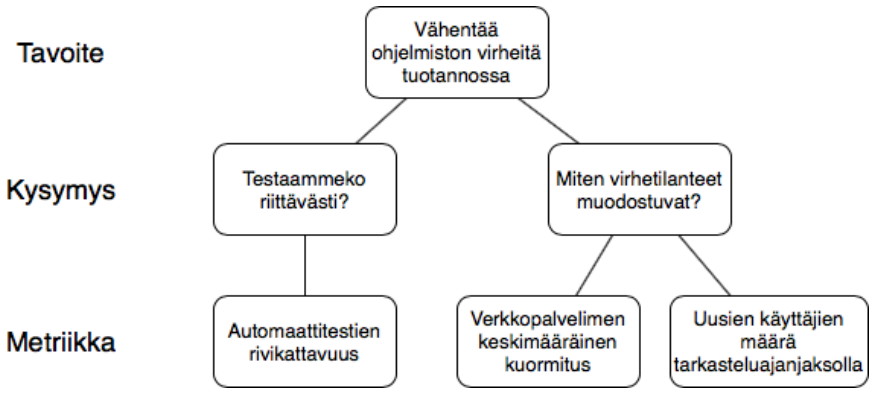
että ohjelmointi sekä ohjelmistojen tuottaminen oli haastava tehtävä ja apukeinoja haasteisiin yritettiin löytää parempien työkalujen ja uusien ohjelmointikielien avulla. Tästä huolimatta järjestelmät myöhästyivät aikatauluista ja suuret alalla toimivat yritykset eivät löytäneet ratkaisua ongelmiin. Kesti vuoteen 1968 asti ennen kuin ongelmat nostettiin pöydälle ja ohjelmistotuotannon haasteista ja ongelmista keskusteltiin avoimesti konferenssissa Saksassa [NaR69]. Konferenssissa käsiteltiin mm. ohjelmistojen testausta sekä järjestelmien suorituskyvyn valvontaa ja ohjelmistojen suunnittelua. Teemat ovat tänä päivänä edelleen ajankohtaisia mutta tieteenalan kypsyessä ohjelmistojen sekä ohjelmistokehitysprosessien arviointi erilaisten metriikoiden avulla on yhä tärkeämmässä roolissa, kun pyritään ymmärtämään ohjelmistotuotannon haasteita ja ohjaamaan ohjelmistokehitystä [KPF95]. Erityisesti se, minkälaiset metriikat auttavat ennakoimaan ohjelmistotuotteen mahdollisia ongelmia ja vikoja ohjelmiston elinkaaren aikana on ollut yksi keskeisimpiä tutkimuksen kohteita ohjelmistoalalla [FeN99].

Mittaaminen ohjelmistotuotannossa on laaja käsite ja metriikoiden kehityttyä mittaamalla pystytään kuvaamaan niin ikään ohjelmistotuotteen kuin ohjelmistokehitysprosessin ominaisuuksia. Mittaaminen on tehokas tapa seurata edistymistä ohjelmistoprojektin tavoitteisiin nähden [Wes05]. Ohjelmistokoodia ja ohjelmistoja kuvaavista metriikoista, jotka tuottavat lukuja, on käytetty nimitystä klassiset metriikat [FeN00]. Näiden avulla ohjelmistokehityksen eri vaiheissa voidaan seurata selkeitä koodin ominaisuuksia kuten rivimäärä ja testikattavuutta. Puhuttaessa ohjelmistotuotannon metriikoista voidaan käsitellä myös ohjelmiston laatuun vaikuttavia tekijöitä sekä työresurssien ja työmäärän arviointia. Edellä kuvatun jaon ohjelmistotuotteen sekä ohjelmistokehityksen mittauksen lisäksi metriikoista on tunnistettu kaksi selkeää osatekijää, joilla molemmilla on omat keskeiset ominaisuutensa. Ensimmäinen metriikan osatekijöistä määrittää sen mitä käytännössä mitataan ja toinen osatekijä puolestaan kuvaa miten mittausdataa kerätään, sekä miten dataa hallinnoidaan ja käytetään [FeN00]. Erilaisia metriikoita ohjelmistoalalla on 1960-luvulta alkaen esitetty satoja, jopa yli tuhanteen asti. Osa metriikoista tuottaa tietoa ohjelmiston kehityksen ympärillä pyörivistä prosesseista esimerkiksi kuvaamalla jäljellä olevan työnmäärää suhteessa tavoiteaikatauluun. Toisaalta jotkut metriikat keskittyvät arvioimaan työn lopputulosta riippumatta siitä, miten siihen on päädytty esimerkiksi käyttäjien tyytyväisyyden perusteella. Huolimatta siitä missä kohtaa ohjelmiston kehitystä mittaus antaa tietoa, on kaikilla metriikoilla yhteiset keskeiset tavoitteet: arvioida ja ennakoida kehitystyön työmäärää sekä kustannuksia ja ohjelmiston laatua. Tästä on seurannut, että

metriikoiden, joiden avulla pyritään seuraamaan ohjelmistokehityksen eri vaiheita, käyttö on yleistynyt niin pienten kuin suurtenkin ohjelmistoyritysten ohjelmistotuotannossa [FeN00]. Valitettavasti metriikoita käytetään joskus väärin ja akateemisen tutkimuksen esittämät metriikat eivät välttämättä sovellu käyttötarkoituksensa ja sisältönsä puitteissa ohjelmistoteollisuuden käyttöön [FeN99].

Mittaamisen yhtenä haasteena on usein oikean metriikan sekä mittaustavan löytäminen. Koodin rivimäärä on ohjelmistoalan metriikkana yksi ensimmäisistä ja esimerkiksi jo 1960-luvun lopun ohjelmistokriisin aikoihin oli se yksi tyypillinen mittari seuraamaan kehittäjän tuottavuutta (mm. koodin rivimäärä per kuukausi) [FeN99]. Sittemmin on kuitenkin tunnistettu, että vaikka koodin rivimäärää on helppo käyttää ja ymmärtää mitä metriikka tarkoittaa, ei se kuitenkaan palvele tarvetta esimerkiksi kehittäjän tuottavuuden arvioinnissa sillä koodin rivimäärä on etenkin ohjelmointikieliriippuvaista mutta myös kannustaa tuottamaan koodia, joka on monimutkaista, vaikealukuista sekä jatkossa hankalasti ylläpidettävää. Yllä oleva esimerkki kuvaa myös hyvin metriikan kahta osatekijää. Rivimäärä on selkeästi mitattava ja numeerisesti esitettävä arvo metriikasta, eli osatekijä, joka vastaa konkreettisesti mittaustuloksesta. Toinen metriikan osatekijä tässä esimerkissä siitä miten tulosta käytetään, on mittaustuloksen soveltaminen kehittäjän tuottavuuden arviointiin.

Jotta organisaatio voi mitata suoriutumistaan tulee sen määrittää tavoitteet itselleen ja projekteilleen sekä muodostaa tapa kerätä tietoa, jota analysoimalla voidaan selvittää, onko tavoitteet saavutettu. Tavoite/Kysymys/Metriikka (engl. Goal/Question/Metric, GQM) on yksi keino määrittää ja arvioida operatiivisia tavoitteita käyttäen mittausta apuna [Bas92]. GQM on tapa yhdistää ja muokata tavoitteita ohjelmistotuotantoprosessien, ohjelmistotuotteiden ja laadun kanssa. Tavoitteet muokataan järjestelmällisesti ja jäljitettävällä tavalla, eli myöhemmin voidaan näyttää, miten tavoitteet on muokattu. Tavoitteiden muovaamiseen käytetään joukkoa kysymyksiä, joiden avulla arvioitavasta projektin tai ohjelmistotuotteen osasta saadaan tietoa. Kysymykset puolestaan auttavat määrittämään joukon metriikoita, joiden avulla tietoa kerätään ja tulkitaan. Eli kääntäen, tietyt metriikat vastaavat tunnistettuihin kysymyksiin, joihin vastaamalla voidaan arvioida, onko tavoitteet saavutettu vai ei. Tyypillisesti yhteen tavoitteeseen liittyy useita kysymyksiä ja kysymykseen vastaus voidaan saada yhden tai useamman metriikan avulla. Yksinkertainen esimerkki GQM-prosessia käyttäen muodostetusta tavoitteesta on esitetty kuvassa 1.



Kuva 1: Tavoite/Kysymys/Metriikka esimerkki

Yksi keino, jolla organisaatiot ja tiimit valitsevat metriikat, joita seurataan, on muodostaa mittausohjelma (engl. software measurement program). Ohjelma kertoo mitä metriikoita ohjelmistotiimillä ja organisaatiolla on käytössä. Termi ei ole kovin tarkasti määritelty vaan sillä pyritään korkealta tasolta kuvaamaan kaikkien niiden metriikoiden joukkoa, jotka on valittu seurattavaksi ja minkä takia. Vaikka metriikoiden käyttö on yleistynyt, metriikkaohjelma on usein jotain mikä muodostetaan vasta kun kehitys ei suju toivotulla tavalla tai on tarve vastata johonkin ulkoiseen vaatimukseen [FeN00]. Metriikkaohjelman tulisi kuvata tiimin tavoitteita ja tukea tavoitteiden saavuttamista [Wes05]. Yksi metriikka voi vastata johonkin seuraavista tavoitteista [Wes05]:

- Auttaa ymmärtämään lisää ohjelmistotuotteesta.
- Auttaa muodostamaan arvio ohjelmiston tietystä osa-alueesta ennalta määritetyn tavoitteen suhteen.
- Tarjota tarvittavaa tietoa, jonka avulla voidaan säädellä esimerkiksi toteutettavien toiminallisuuksien prioriteettia.
- Pyrkiä ennakoimaan ohjelmiston tulevia vaatimuksia.

Hyvässä metriikkaohjelmassa on valittuna metriikoita siten, että jokaista yllä esitettyä kohtaa voidaan seurata.

Aiemmin esimerkissä mainitun koodin rivimäärän lisäksi itse ohjelmiston osalta selkeitä ja helposti ymmärrettäviä metriikoita kuten testien koodikattavuus sekä koodin vaihtuvuus (engl. code churn) on edelleen paljon käytössä. Näiden metriikoiden ongelma on kuitenkin se, että ne tarjoavat tuloksia jälkikäteen, kun ohjelmistoa ja koodia on jo kehitetty. Näin ollen metriikoiden keskeinen vaatimus tuottaa eri sidosryhmille kuten johtohenkilöstölle tietoa ohjelmiston elinkaaren eri vaiheista, kustannuksista ja mahdollisista vioista etukäteen päätöksenteon tueksi ei täyty [FeN00]. Vaikka mittauskeinoja ja

metriikoita, jotka tukisivat päätöksentekijöitä myös ohjelmistojen kehitysvaiheissa erityisesti vikojen vähentämiseksi, on tutkittu ja kehitetty yhä enemmän, eivät ne ole levittäytyneet laajemmin käyttöön teollisuuden puolelle [FeN99, RHT13]. Ohjelmistoalan metriikat mahdollistavat tiedon keräystä ja mittausta vaikkakin suuri osa tiedon keräyksestä tapahtuu jälkikäteen. Metriikoita, jotka hyvissä ajoin ennakoivat ohjelmiston tuotannon aikaisten vikatilanteiden tiheyttä ei ole onnistuttu tuottamaan. Metriikoiden avulla on sen sijaan onnistuttu haastamaan oletuksia kuten, että isommat ohjelmistomoduulit aiheuttavat enemmän vikoja tai mikäli vain pieni osa ohjelmiston toiminnallisuuksista aiheuttaa useimmat tuotannon aikaiset ongelmat, on näiden toiminnallisuuksien osuus koko ohjelmiston koosta suuri [FeN99].

Regressiivisten metriikoiden rinnalle on yhä enemmän alettu tuoda metriikoita ja mittausmalleja, jotka pyrkivät huomioimaan ohjelmistojen eri suunnittelu- sekä tuotantovaiheissa puuttuvan ja epävarman tiedon. Tietoja, jota ei pystytä etukäteen luotettavasti arvioimaan ovat esimerkiksi ohjelmistotuotteen vajaat vaatimukset tai miten kehityksen aikaiset viat ja vikojen korjaaminen vaikuttavat tuotannon aikaisten vikojen ilmenemiseen. Yksi malli, joka on todettu hyödylliseksi näiden ongelmien ratkaisuun, on yksi Bayesilaisen verkon malleista (engl. BBN, Bayesian belief net) [FeN99, Hec08]. Ohjelmistotuotannossa mallin käyttö on yleistynyt algoritmien kehityksen sekä markkinoille tulleiden tuotteiden, jotka hyödyntävät mallia ansiosta [FNM08]. Historian valossa on kuitenkin nähtävillä, että metriikoiden kehityksestä huolimatta on edelleen tarvetta myös yritykskohtaisille metriikoille, jotka tukevat juuri kyseisen yrityksen tarpeita. Fenton ja Neil [FeN00] ovat esittäneet viisi kohtaa yhteenvetona ohjelmistotuotannon metriikoiden tarpeesta:

- Ohjelmiston metriikoiden tulisi tarjota johtotasolle määrällisen päätöksenteon tukea ohjelmiston elinkaaren aikana.
- Metriikoiden antaman tuen päätöksenteolle tulisi tukea riskien arviointia ja vähentämistä.
- Perinteiset regressiivisiä metriikoita hyödyntävät lähestymistavat kustannusten arvioon ja vikojen ennustamiseen tarjoavat heikosti tukea johtajille, jotka haluavat käyttää mittausta apuna riskien analysointiin ja minimointiin.
- Ohjelmiston metriikoiden tulevaisuus nojaa sen varaan, että käytetään yksinkertaisia olemassa olevia metriikoita rakentamaan johtotason päätöksenteon tueksi työkaluja, jotka yhdistävät useita eri ohjelmistokehityksen ja testauksen

näkökulmia sekä auttavat johtajia tekemään ennusteita ja arvioita ohjelmiston elinkaarenaikana.

- Metriikoista usein puuttuvia tekijöitä kuten kausaliteetti, epävarmuus ja aineiston yhdistely tulisi käsitellä hyödyntäen syyseuraus-mallinnusta (mm. BBN), empiiristä ohjelmistotuotantoa sekä useaan kriteeriin perustuvaa päätöksentekoa.

2.2 Ketterä ohjelmistotuotanto

Siitä huolimatta, että ohjelmistotuotanto on verrattain nuori ja kehittyvä ala esimerkiksi rakennusalaan verrattuna, on etenkin asiakkaiden kannalta pidettävä mielessä, että varsinkin suuret IT-projektit ovat kalliita, aikaa vieviä ja ajautuvat välillä umpikujaan, jonka seurauksena projektia ei saada toimitettua. Erilaisten ohjelmistojen käyttö ja kuinka paljon niiden parissa kulutamme aikaa päivittäin kasvaa koko ajan, jolloin ohjelmistoala onnistumiset ja epäonnistumiset ovat ymmärrettävästi laajemmin näkyvillä. Jo 90-luvulla on järjestelmällisesti tunnistettu keskeisiä syitä kuten hallinnollisen tuen puute, epäselvät vaatimukset sekä palautteen puuttuminen oikeilta käyttäjiltä ohjelmistokehityksen aikana, jotka johtavat projektien epäonnistumiseen [Cla95]. Perinteinen vanhan mallinen ohjelmistokehitys on selkeästi jaettu työvaiheisiin, jotka toteutetaan peräkkäin yksi vaihe kerrallaan. Yksi laajalti käytetty ohjelmistokehitysmenetelmä, joka toimii näin, on vesiputousmalli [Roy87]. Kun ohjelmistoa on mallin mukaan kehitetty vaihe vaiheelta, saadaan se lopulta asiakkaalle testattavaksi, jolloin etenkin nykypäivän nopeasti muuttuvien ja päivittyvien teknologioiden sekä kilpailutilanteen johdosta, ei tuote enää vastaa asiakkaan tarpeita [Boe96]. Suuret muutokset vaatimuksiin lähellä ohjelmistotuotteen käyttöönottoa ovat usein kalliita sekä aiheuttavat viivästystä projektiin [HiC01].

60-luvun lopun ohjelmistokriisin lisäksi [NaR69], on 2000-luvun vaihteesta tunnistettavissa muutos ohjelmistoalalla. Projektien epäonnistumisten seurauksena alkoi kehittyä metodeja ja tekniikoita, jotka pyrkivät muokkaamaan ja tukemaan ohjelmistotiimien päivittäistä työtä. Asiakkaiden tyytymättömyys sekä ohjelmistoalan ammattilaisten halu kehittyä olivat keskeinen motivaatio, joka johti vuoden 2001 ketterän ohjelmistotuotannon manifestiin [Bec01]. Manifesti välittää ketterien menetelmien keskeisiä arvoja, kuten toimivan ohjelmiston tärkeyttä kattavan dokumentoinnin ja suunnittelun sijaan sekä yksilön ja tiimin tärkeyttä erilaisten työkalujen ja tarkkojen prosessien sijaan. Ketterät menetelmät suhtautuvat positiivisesti muutoksen ja muutos on oikeastaan toivottua, jotta varmistutaan että tuote on valmistuessaan asiakkaan vaatimusten mukainen [HiC01]. Osaltaan

ketterien menetelmien ansiosta ohjelmistoprojektien loppuun saattaminen on lisääntynyt sekä projekteista luopuminen on vähentynyt [ElK08].

Keskeinen ero ketterän ohjelmistotuotannon ja perinteisen ohjelmistotuotannon välillä on projektien eri vaiheiden toteutusaikataulussa. Kun perinteinen ohjelmistotuotanto on nojannut ohjelmistotuotannon eri vaiheiden peräkkäiseen toteutukseen, ketterä ohjelmistotuotanto pyrkii toistamaan näitä vaiheita ja vaiheita suoritetaan usein samanaikaisesti. Yhtä sykliä, jossa ohjelmistoa suunnitellaan, kehitetään, testataan ja siirretään tuotantoon ja loppukäyttäjän ja asiakkaiden arvioitavaksi kutsutaan iteraatioksi [HiC01]. Iteratiivinen kehitys mahdollistaa tiiviimmän vuorovaikutuksen ohjelmistoprojektin kaikkien sidosryhmien kanssa läpi koko ohjelmistokehityksen elinkaaren.

Extreme Programming (usein lyhennetty XP) [Bec00] on ollut ketterä menetelmä suuressa osassa ketteriin menetelmiin keskittyneessä tutkimuksessa [DDA08]. XP tarjoaa ohjelmistotiimeille käytänteitä, joiden tavoite on tukea ohjelmiston kehitystä. Nämä käytännöt ovat tarkasta määritelty, joten niitä usein mielletään myös säännöiksi ja XP:n yksi tärkeä kulmakivi on, että tiimi sitoutuu noudattamaan kullakin hetkellä valittuja sääntöjä [Bec99]. Sana extreme (äärimmäinen) kuvaa hyvin myös sitä, että sääntöjen tiukka noudattaminen vaatii tiimin lisäksi myös muilta sidosryhmiltä kuten loppuasiakkaalta ymmärrystä ja sitoutumista ohjelmiston kehitykseen. Kehitystiimit voivat itse valita ja vaikuttaa mitä sääntöjä ne kullakin hetkellä noudattavat. Alla on listattu muutamia XP:n keskeisiä periaatteita ja käytänteitä [Bec99]:

- Yhteinen koodin omistajuus (engl. collective code ownership), jokainen ohjelmoija voi parantaa koodia missä kohtaa ohjelmaa tahansa, jos siihen on mahdollisuus.
- Suunnittelupeli (engl. planning game), asiakkaat päättävät julkaisujen aikataulun ja sisällön kehitystiimin arvioiden pohjalta.
- Kehitystyön tilaaja (asiakas) paikan päällä (engl. on-site customer), asiakas on täyspäiväisesti läsnä tiimin kanssa.
- Pariohjelmointi (engl. pair programming), kaikki tuotantokoodi tuotetaan kahden kehittäjän toimesta yhdellä tietokoneella.
- Jatkuva integrointi (engl. continuous integration), uusi koodi yhdistetään tuotantokoodiin muutaman tunnin kuluessa. Tässä yhteydessä kaikkien testien on mentävä läpi tai muutokset sivuutetaan.

- Testaus (engl. tests), Ohjelmoijat kirjoittavat yksikkötestejä yhdessä muun ohjelmakoodin kanssa. Testien tulee kaikkien mennä läpi samalla testiajolla. Kehitystyön tilaaja tuottaa iteraation toiminnalliset testit.

Jotkut XP:n käytänteistä (vain pieni osa listattu yllä), eivät ole käytännössä aina mahdollisia tiimeille. Esimerkiksi pariohjelmointia kuten XP sen esittää, ei voida toteuttaa hajautetuissa pääosin etänä työskentelevissä kehitystiimeissä. Tämä ei silti tarkoita, etteikö tiimi voisi valita omaan työskentelyynsä sopivia käytänteitä. Myös sääntöjen vaihtaminen on tiimeille sallittua missä kohtaa tahansa, kunhan tiimillä on sovittu etukäteen, miten he voivat arvioida muutoksen vaikutuksen omaan kehitystyöhönsä. Ei ole tarkoituksenmukaista pitää kiinni säännöistä, jotka eivät enää tue tiimin työskentelyä [Bec99].

Scrum ei ole varsinaisesti ketterä menetelmä, vaan kehys, jonka ympärille ohjelmistotiimit voivat rakentaa oman ohjelmistokehityksensä [ScS17]. Scrum ei sanele yksityiskohtaisesti, miten tuotetta tulee kehittää mutta se määrittää roolit, artefaktit sekä tapahtumat, joita kaikki Scrumia käyttävät tiimit noudattavat. Scrum-tiimissä on aina kolme roolia, tuoteomistaja, kehitystiimi sekä Scrum Master. Tuoteomistaja edustaa asiakasta ja muita sidosryhmiä kehitystiimin suuntaan ja valitsee mitä toiminnallisuuksia toteutetaan. Scrum Master avustaa kaikkia tiimin sekä sidosryhmien osapuolia Scrumin noudattamisessa. Kehitystiimi on itseohjautuva, eli tiimin rooleja ei sanella ulkopuolelta ja kaikki henkilöt kehitystiimissä ovat yhdenvertaisia, esimiehiä tai alaisia ei ole.

Scrumin määrittämät kolme artefaktia ovat työlista (engl. prodcuct backlog), sprintin työlista (engl. sprint backlog) sekä lisäys (engl. increment). Työlista on listaus kullakin hetkellä tiedossa olevista ohjelmistoprojektin vaatimuksista. Listan ylläpito on tuoteomistajan vastuulla. Sprintin työlista pitää sisällään kuhunkin sprinttiin valitut toiminnallisuudet. Ketterän ohjelmistokehityksen iteraatiota kutsutaan Scrumissa sprintiksi. Lisäys kuvaa sprintin jälkeen ohjelmistotuotteeseen lisättävää ja asiakkaalle toimitettavaa päivitystä.

Artefaktien ja roolien lisäksi Scrum määrittää viisi tapahtumaa. Sprintti kuten aiemmin jo mainittiin kuvaa ketterän ohjelmistokehityksen iteraatiota. Sprintin suunnittelu (engl. sprint planning) on ennen sprinttiä pidettävä tilaisuus Scrum-tiimin kesken, jossa kehitystiimi ja tuoteomistaja määrittävät mitä toiminnallisuuksia sprintille valitaan ja mikä on sprintin tavoite. Päivittäinen Scrum (engl. daily Scrum) on jokaisena työpäivänä järjestettävä lyhyt palaveri kehitystiimin kesken, jossa mm. tarkastellaan, miten sprintti etenee

jäljellä olevan työmäärän puitteissa. Sprintin katselmointi (engl. sprint review) on tilaisuus sprintin jälkeen, jossa tarkastellaan ja arvioidaan Scrum-tiimin sekä mahdollisten muiden sidosryhmien kanssa sprintin tuottamaa lisäystä. Viimeisenä tapahtumana on sprintin retrospektiivi (engl. sprint retrospecive), jossa Scrum-tiimi arvioi omaa onnistumaan päättyneellä sprintillä, pyrkii puuttumaan mahdollisiin ongelmiin ohjelmistokehityksessä ja miettii mitä seuraavalle sprintille voidaan oppia.

Scrum ja XP eivät rajaa toinen toisiaan pois, vaan niitä voidaan myös käyttää yhdessä [Kni15, MaS02]. Samankaltaisuuksia löytyy suunnittelunvaiheen suunnittelupelistä (XP) sekä sprintin suunnittelusta (Scrum), jossa asiakkaan toiveiden mukaisesti valitaan kehitettäviä tehtäviä. Myös kehitystyön tilaaja paikan päällä (XP) sekä tuoteomistaja, joka edustaa asiakasta kehitystiimille (Scrum) ovat hyvin vastaavia. Koska Scrum ei tarjoa selkeää kehitysohjeistusta, ovat jotkut tiimit päätyneet Scrumin ja XP:n yhdistämiseen siten, että noudatetaan Scrumin kehystä ja tuodaan kehitystiimille XP:n työkaluja kuten pariohjelmointia ja testausta [JeZ03, Kni15, MaS02, MuQ12]. Scrum on myös tarjonnut näissä tapauksissa parempia työkaluja työtehtävien priorisointiin tuoteomistajan roolin kautta.

Tässä tutkielmassa ketterällä ohjelmistotuotannolla tarkoitetaan ohjelmistotuotantoa käyttäen ketteriä menetelmiä kuten XP yhdistettynä myös Lean-ohjelmistotuotantoon sekä Lean-menetelmiin. Ketterä ohjelmistotuotanto sekä Lean-ohjelmistotuotanto eivät automaattisesti kuulu yhteen, vaikka esimerkiksi englanninkielinen termi ’Leagile’ on muodostettu kuvaamaan näiden kahden tuotantotavan yhdistelmää [WCC12]. Lean-ajattelu juontaa juurensa autonvalmistaja Toyotan tuotantolinjastolle ja nykyään ajattelua sovelletaan monilla toimialoilla. Muutama vuosi ketterän ohjelmistotuotannon manifestin jälkeen myös Lean-ajattelu alkoi kiinnittää ohjelmistoalan huomiota ja ensimmäiset sovellukset ohjelmistotuotantoon ovat vuodelta 2003 [PoP03]. Ketterät menetelmät kuten Scrum sekä XP olivat ensisijaisesti suunnattu ohjelmistotiimien työkaluiksi, mutta Lean pyrki tuomaan ketteriä työkaluja koko organisaation tasolle [RMO19]. Lean-ajattelun keskeisiä periaatteita on tutkittu ja määritelty myös toimialoittain mutta keskeiset viisi yleisesti tunnustettua alun perin Womackin ja Jonesin periaatetta on esitetty seuraavaksi [RMO19]. Jokaisen kohdan yhteyteen on myös lisätty esimerkki, miten tätä periaatetta voidaan soveltaa ohjelmistotuotannossa:

- Arvo (engl. value), on kaikkea mistä asiakas on valmis maksamaan. Arvon

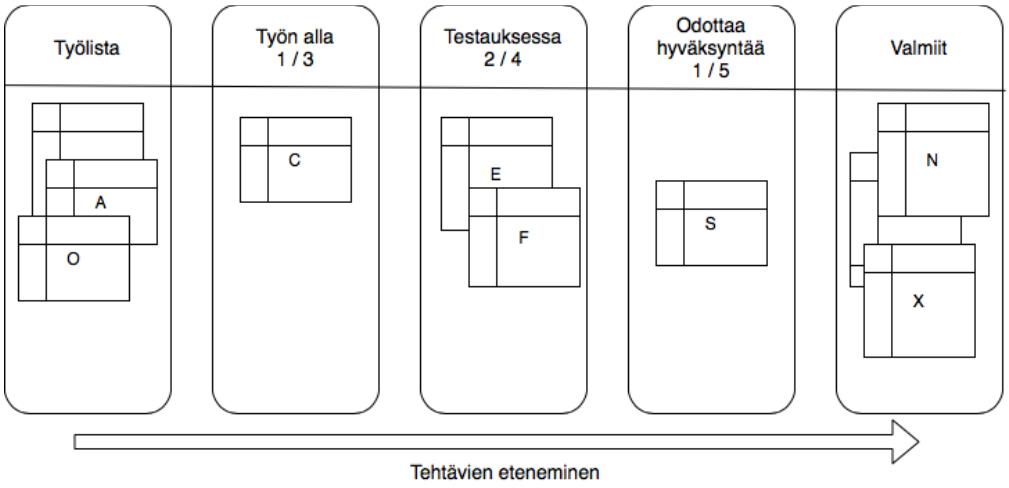
ymmärtäminen asiakkaan näkökulmasta on keskeistä Lean-ajattelussa ja jokaisen toimenpiteen tulisi tuottaa asiakasarvoa. Sen vastakohta on jäte (engl. waste), joka tarkoittaa kaikkea tekemistä, johon kuluu resursseja mutta se ei tuota arvoa. Ohjelmistotuotannossa tämä voi yksinkertaisimmillaan tarkoittaa tuotteen uutta ominaisuutta, joka osoittautuu hyödylliseksi tuottaen arvoa tai jätteeksi, jolloin ominaisuus on ollut turha.

- Arvovirta (engl. value stream) on optimoitu tuotannon päästä päähän kulkeva joukko toimintoja, joita vaaditaan tuotteen saattamiseksi asiakkaan tilauksesta ylläpitoon. Toimintoja on kolmenlaisia: ensimmäisenä toiminnot, jotka tuottavat asiakasarvoa kuuluvat arvovirtaan. Toisena toiminnot, jotka eivät tuota arvoa mutta ovat välttämättömiä arvon tuottamiseen myöhemmin kuuluvat arvovirtaan. Tavoite on, että näistä toiminnoista päästään myöhemmin eroon. Kolmas tyyppi ovat toiminnot, jotka eivät tuota mitään arvoa nyt eikä niistä ole hyötyä arvon tuottamiseen myöhemmin. Nämä tulisi eliminoida välittömästi. Ohjelmistotuotannossa arvoa kuvaava esimerkki on jo edellä mainittu hyvä uusi toiminnallisuus. Esimerkkinä toiminnosta, joka ei tuota arvoa mutta on välttämätön, olisi esimerkiksi teknisen velan maksaminen. Tavoitetila on, että teknistä velkaa ei ole, jolloin siihen ei tarvitse jatkossa käyttää resursseja (toki tämä ei ole kovin realistista ohjelmistotuotannossa). Esimerkkinä toiminnosta, joka voitaisiin ohjelmistotuotannossa karsia pois, on turha dokumentointi. Toki määritelmä sille mikä dokumentointi on turhaa, on harkittava tarkkaan.
- Virtaus (engl. flow) tarkoittaa, että tuotantotoimenpiteet pyritään järjestämään yhtenä jatkumona, jolloin tuotantoon ei tule keskeytyksiä. Ylimääräiset vaiheet tuotannosta pyritään poistamaan. Esimerkkinä ohjelmistoja saatetaan testata manuaalisesti, jolloin manuaalisen testauksen odotus saattaa hidastaa ominaisuuden päätymistä käyttäjille. Manuaalinen testaus voidaan siirtää automaattisen testauksen hoidettavaksi.
- Imu (engl. pull) tarkoittaa, että tuotteen tai ominaisuuden valmistusta ei aleta tuottamaan ennen kuin tuotantolinja on valmis. Yksinkertainen esimerkki ohjelmistojen puolelta tälle on se, että ominaisuutta, jolla poistetaan käyttäjiä järjestelmästä ei toteuteta ennen kuin järjestelmään voidaan ylipäätään lisätä käyttäjiä.
- Täydellisyys (engl. perfection, alun perin japaniksi Kaizen), tarkoittaa koko organisaation kattavaa toistuvaa oppimista ja optimointia. Ketterästä

ohjelmistotuotannosta vastaavanlainen oppimismahdollisuus tälle löytyy esimerkiksi Scrum-tiimin retrospektiivistä [Sch04], jossa ohjelmistotiimi voi säännöllisin väliajoin kehittää työskentelyään ja oppia aiemman työnsä pohjalta.

Lean-periaatteista sekä ketterän ohjelmistotuotannon manifestin arvoista löytyy paljon samankaltaisuuksia ohjelmistoalalle. On kuitenkin tyypillisempää hyödyntää Lean-ajattelua ja muita ketteriä menetelmiä yhdessä, pelkän Lean-ajattelun soveltaminen on harvinaista [RMO19].

Kanban on yksi Lean-periaatteiden työskentelytapa, jonka yhdistämisestä esimerkiksi Scrumin kanssa on paljon kokemusta [KnS10]. Kanbanin ajatus on visualisoida työn etenemistä käyttäen Kanban-taulua, jossa työtehtävät sijoitetaan sarakkeisiin. Kanban ei määritä miten ohjelmistokehitystä tehdään, vaan pyrkii tukemaan ohjelmistokehitystä näyttämällä reaaliaikaista kuvaa, miten kehitys edistyy. Jokainen sarake kuvaa työvaihetta kehitysprosessissa. Taulut voivat olla esimerkiksi tiimin työhuoneessa nähtävillä tai verkossa tiimin käyttämien työkalujen kautta saatavilla. Tiimi voi itse muokata miten he nimeävät sarakkeet omaan ohjelmistokehitykseensä sopiviksi. Kanban myös kannustaa tiimejä lisäämään rajoituksen per sarake, kuinka moni tehtävä voi milläkin hetkellä olla kyseisessä työvaiheessa. Tällä pyritään välttämään pullonkauloja ohjelmistokehityksessä, sillä tehtävien tulisi liikkua vasemmalta oikealle ennen kuin uusia voidaan lähteä tuomaan kehitykseen. Kanban ohjaa tiimejä mittamaan läpimenoaikaa työtehtäville (engl. lead time tai cycle time), joka kuvaa kuinka nopeasti työtehtävä sen jälkeen, kun se on lisätty Kanban-tauluun, on kulkenut taulun läpi ja valmistunut. Esimerkki Kanban-taulusta on nähtävillä kuvassa 2.



Kuva 2: Kanban-taulu

Ketterän ohjelmistotuotannon kehitys ja tutkimus keskittyy yhä enemmän tarpeeseen vastata käyttäjän vaatimuksiin, tarpeisiin ja muutoksiin nopeasti [RMO19]. Aito palaute tulee viime kädessä loppukäyttäjältä ja pienetkin päivitykset halutaan saada käyttäjille testattavaksi ilman viiveitä. Ohjelmistojen päivitys loppukäyttäjille tapahtuu nykyään perinteisen aikatauluun sidotun julkaisun tai versiopäivitysten lisäksi myös jatkuvasti. Tämän jatkuvan toimituksen (continuous delivery) pohjalla on kaksi teknologian mahdollistamaa metodia, jatkuva integrointi (continuous integration) [HuF10 s. 56-61] sekä jatkuva päivitys tuotantoon (continuous deployment) [HuF10 s. 266-273]. Jatkuvassa integroinnissa ohjelmistotiimi on koonnut itselleen ohjelmistokehitysympäristön ja työkalut, joiden avulla päivitykset koodiin ajetaan automaattisten testien läpi ja testien mennessä läpi, koodi ja uusi toiminnallisuus yhdistetään osaksi tuotetta. Tässä vaiheessa ominaisuus on siirrettävissä tuotantoon mutta vaatii vielä manuaalisen vaiheen ennen kuin ominaisuus näkyy loppukäyttäjille. Jatkuva päivitys tuotantoon vie automaation vielä yhden vaiheen pidemmälle, jolloin uudet ominaisuudet ovat loppukäyttäjän saatavilla ilman manuaalista vaihetta. Yritykset ovat löytäneet sekä täysin automaattisesta tuotantoon siirrosta että manuaalisesta tuotantoon siirrosta automaattisen integroinnin jälkeen sekä hyötyjä että haittoja [LMP15]. Käytäntö on yleistyvä ja suuret palveluntuottajat kuten Netflix sekä Facebook hyödyntävät jatkuvaa päivitystä käyttäjilleen [SDG16].

2.3 Mittaus ketterässä ohjelmistotuotannossa

Perinteistä ohjelmistotuotantoa ohjaa selkeä suunnitelma, joka laaditaan projektin alkuvaiheissa. Ketterässä ohjelmistotuotannossa tavoite on saada nopeasti jotain toimivaa toimitettua asiakkaalle, ja tämän jälkeen ohjelmiston kehitys jatkuu pieni lisäys kerrallaan. Ohjelmistokehitystä ohjaa vahvasti aiemman iteraation lopputulos ja suunnitelmat päivittyvät jatkuvasti. Perinteisessä ohjelmistotuotannossa osa metriikoista, joita halutaan seurata, voidaan valita jo projektin suunnitteluvaiheessa määritettyjen tavoitteiden perusteella. Esimerkiksi tiimille voidaan asettaa tuottavuustavoite, jonka pohjalta projektin pitäisi valmistua tietyssä ajassa. Etukäteen asetettu tavoite ja suunniteltu työmäärä on kuitenkin ketterän ohjelmistotuotannon periaatteita vastaan. Muutos on toivottua, jotta varmistutaan että lopullinen ohjelmisto täyttää vaatimukset mahdollisimman hyvin. Erilainen ohjelmistokehitystyyli vaikuttaa myös siihen, miten metriikoita voidaan valita ja käyttää [KMI15]. Tavoite metriikoille on kuitenkin sama, pyrkiä toimimaan päätöksenteon tukena sekä kuvata kehityksen tilannetta kaikille sidosryhmille.

Metriikoita ketterän ohjelmistotuotannon eri vaiheissa on tunnistettu suuri määrä [KMI15, OzK12, WnM17]. Tässä tutkielmassa metriikoita käsitellään kolmessa ryhmässä. Ensimmäinen ryhmä on ennen tuotannon aloitusta vaatimusmäärittelyn aikaiset metriikat. Toinen ryhmä on tuotannon aikaiset metriikat, jotka kuvaavat itse kehitettävän tuotteen ominaisuuksia mutta myös ohjelmistokehitysprosessia ja projektin edistymistä. Kolmas ryhmä sijoittuu tuotteen käytön ja ylläpidon vaiheeseen, sillä huomiolla että tässä ryhmässä on myös osittain päällekkäisyyttä toisen ryhmän kanssa. Erityisesti kolmas ryhmä pyrkii käsittämään metriikoita, joita on havaittavissa vasta kun tuote on loppuasiakkaan käytössä ja tuotannossa. Vaikka metriikat voidaan jakaa ryhmiin sen perusteella mihin vaiheeseen ohjelmiston kehityksen elinkaarta ne liittyvät on kuitenkin huomattu, että tutkimuksen tarjoamien metriikoiden lisäksi yrityksillä ja tiimeillä on käytössä omia metriikoita [KMI15]. Vastaava havainto on todettu myös aiemmissa tutkimuksissa [FeN00].

Kuten perinteisessä ohjelmistotuotannossa, myös ketterässä ohjelmistotuotannossa projektin alkuun täytyy kartoittaa vaatimuksia ja tehdä alustavia suunnitelmia, miten projektia lähdetään viemään eteenpäin. Tavoite on kuitenkin, että vaatimusten määrittely ja keräys ei ole liian raskas toimenpide sillä muutoksia voidaan olettaa tulevan [Bec01]. Erityisesti vaatimusten määrittelyn osalta ennen ohjelmistokehityksen aloitusta on huomattu, että metriikoiden käyttö on hyvin niukkaa [KMI15, WnM17]. Syyksi tähän on esitetty mm. se, että iteratiivisen kehityksen myötä varsinaiseen suunnitteluun etukäteen ei käytetä paljoa resursseja eikä sille anneta liian suurta painoarvoa. Myös Lean-periaate jätteen eliminoinnista on yksi selittävä tekijä suunnittelun metriikoiden vähäisemmälle tuntemukselle ja tutkimusnäytölle, sillä pelkän suunnittelun ja dokumentoinnin ei voi nähdä tuottavan arvoa. Kehitystiimi ei välttämättä pysty myöskään arvioimaan vaatimusten laatua, eli mikäli kaikkiin vaatimuksiin vastataan, saako asiakas tällöin mitä haluaa. Asiakkaalla eli työn tilaajalla on tärkeä rooli varmistaa, että hänen toiveensa ja tavoitteensa on huomioitu. Mittarit, joita vaatimusmäärittelyn jälkeen voidaan seurata kuten ohjelmistoprojektin arvioitu kokonaistyömäärä, muodostuvat vaatimusmäärittelyn aikana. Alustavan vaatimusmäärittelyn pohjalta esimerkiksi Scrum-tiimi muodostaa projektin työlistan ja tässä vaiheessa voisi olla tärkeää nähdä ja mitata miten alustavat työsuunnitelmat vastaavat muiden sidosryhmien toiveita ja vaatimuksia. Vaatimusten sekä suunnittelun arviointi tapahtuu usein ainoastaan osana ohjelmiston kehitysiteraatioita [KMI15]. Asiakkaalle se miten hyvin vaatimukset ovat muuntuneet työtehtäviksi kehitystiimille avautuu

usein vasta kun hän näkee vaatimusten pohjalta toteutetut ominaisuudet osana ohjelmistotuotetta.

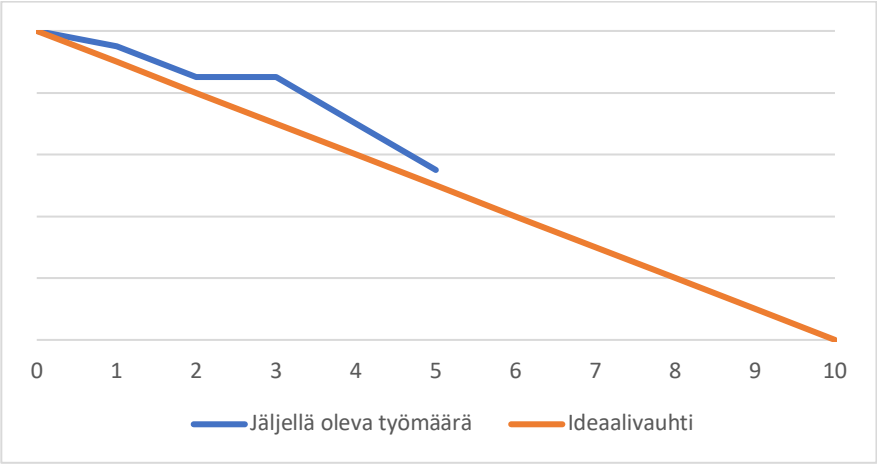
Ohjelmistoprojektin lähtiessä käyntiin ketterässä ohjelmistokehityksessä, alkavat ohjelmistokehitystavasta riippuen eri vaiheet kuten suunnittelu, testaus ja tuotantoon siirto toistaa toisiaan. Useassa eri tutkimuksessa on havaittu, että metriikoiden ja mittauksen painopiste keskittyy myös tähän vaiheeseen ohjelmistokehitystä kuten Kupiainen ym. kokoaa [KMI15]. Usein esiin nousseet metriikat kuten velositeetti (engl. velocity) sekä työmääräarvio (engl. effort estimate) tukevat ohjelmistotiimin suunnittelua sekä projektin etenemisen seurantaa. Osa ohjelmistokehityksen aikaisista metriikoista tuottaa mittaustuloksia automaattisesti ohjelmistokehityksen edetessä. Esimerkiksi koodin rivimäärä sekä testien rivikattavuus ovat kehitystiimin nähtävillä ja päivittyvät muutosten mukana. Sen sijaan metriikat kuten jäljellä oleva työmäärä sekä työtehtävän läpimenoaika vaativat manuaalisia toimenpiteitä. Jäljellä oleva työmäärä ei muutu, ellei työtehtävien valmistuessa käydä päivittämässä jäljellä olevaa työn määrää. Tätä voidaan osittain automatisoida yhdistämällä jäljellä olevan työn seuranta keskeneräisiin työtehtäviin, jolloin tehtävän valmistuessa kyseisen työtehtävän verran työmäärää vähennetään jäljellä olevan työmäärään kokonaisuudesta. Työtehtävän läpimenoaikaa vaatii myös tavan kerätä ja tallentaa läpimenoaikoja, jotta tiimi voi myöhemmin arvioida esimerkiksi keskimääräistä vaatimusten läpimenoaikaa.

Velositeetti pyrkii kuvaamaan tahtia, millä kehitystiimi saa toiminnallisuuksia ja ominaisuuksia tehtyä. Velositeetti lasketaan aina valitulle tarkastelujaksolle, joka voi olla esimerkiksi kuukausi, viikko tai iteraatio. Laskukaava on valmistunut työmäärä / aikajakso. Esimerkiksi Scrum-tiimeissä työmäärää kuvataan tehtäväpisteillä (engl. story point), jotka kuvaavat tehtävien suuruusluokkaa toisiinsa nähden. Tiimi päättää itse, miten haluavat pisteytystä käyttää tehtävien koon arviointiin, yleensä käytössä on pieniä lukuja väliltä 0 – 10. Näin ollen, kun tiimin kehitysvauhti on tiedossa, saadaan velositeetista tukea seuraavan sprintin työmäärän suunnitteluun ja arviointiin, jotta pystytään valikoimaan sprinttiin mukaan työmäärä, jonka pitäisi olla tehtävissä.

Työmääräarvio on oleellinen osa iteraatioiden suunnittelua, sillä työmäärän arvion kautta saadaan käsitys siitä, kuinka kauan toiminnallisuuksien ja muiden tehtävien tekeminen vaatii. Ajateltaessa työmääräarvioita metriikkana, voidaan tarkoittaa sitä, että jälkeenpäin tarkastellaan kuinka hyvin työmääräarviot ovat pitäneet paikkansa. Samalla tiimit saavat arvokasta tietoa miltä osin heillä on arvioinnissa kehitettävää, jotta arviot saadaan

vastaamaan hyvin lopullista työmäärää. Toisaalta työmääräarvio voi metriikkana olla työmäärää kuvaava tuntimäärä tai pistemäärä, joka tehtävälle on arvioitu. Iteraatioiden uunnittelussa voidaan huomioida myös ennakoitua uuden ominaisuuden arvoa liiketoiminnalle tai tarvittavaa resurssien määrää, jotta valmistuvat työtehtävät saadaan kattavasti testattua [KMI15].

Edellä mainittua velositeettia voidaan myös seurata iteraation aikana ja pyrkiä samalla ennakoimaan onko valittu työmäärä valmistumassa. Toinen useassa tutkimuksessa esille noussut metriikka on iteraation aikainen burndown ja burndown-kaavio [KMI15]. Se pyrkii esittämään työmäärän ja jäljellä olevan ajan suhdetta ja näyttämään ollaanko tavoitteet saavuttamassa. Alla kuvassa 3 on esimerkki burndown-kaaviosta kymmenen työpäivän pituisessa iteraatiossa.



Kuva 3: Burndown-kaavio

Ideaalivauhti esittää kuinka paljon jäljellä olevan työmäärän pitäisi päivittäin pienentyä. Jäljellä oleva työmäärä seuraa kuinka paljon työtä on jäljellä. Kaaviosta nähdään, että päivän viisi jälkeen, tiimi on hieman ideaalivauhtia jäljessä. Iteraatioiden aikaisten metriikoiden yksi tavallisesti mainittu tavoite on lisätä läpinäkyvyyttä sekä tiimin mutta myös muiden sidosryhmien osalta [KMI15].

Iteratiivisen ja inkrementaalisen ohjelmistokehityksen tavoite on, että iteraation jälkeen tiimillä on iteraatioon valitut ohjelmiston muokkaukset valmiina toimitettavaksi asiakkaalle. Iteraation lähentyessä loppuaan erilaiset metriikat kuten vikojen määrä voivat auttaa laadun varmistamiseksi ja virhetilanteiden vähentämiseksi tuotannossa. Vikoja voidaan yrittää ehkäistä mittaamalla esimerkiksi testien rivikattavuutta sekä testimäärän kasvua suhteessa koodimäärään kasvuun. Virheiden määrää tuotannossa voidaan käyttää

suoraan yhtenä metriikkana kuvaamaan ohjelmiston laatua. Mikäli virheitä tuotannossa kuitenkin esiintyy toistuvasti, voidaan kehityksen aikaisia mittareita kuten testauksen kriteerejä tarkastella ja muokata. Ketterässä ohjelmistotuotannossa metriikoiden ja mittauksen osuminen kohdalleen vaatii iteraatiota ja opettelua [OzK12]. Myös loppukäyttäjien ja työn tilaajan palaute on tärkeää arvioitaessa valmistuvan iteraation työn laatua ja vaatimusten täyttymistä. Käyttäjiltä voidaan aktiivisesti kerätä palautetta uusista ominaisuuksista ennen tuotantopäivitystä valikoimalla pieni osa käyttäjiä testaamaan uusia ominaisuuksia. Erilaiset metriikat ja mittaukset tuotteen elinkaareen loppupäässä tuotantokäytössä ja ylläpidossa tuottavat sekä automaattista että manuaalisesti kerättävää tietoa. Asiakkaan tyytyväisyyttä on vaikea arvioida kysymättä asiaa. Loppukäyttäjien tyytyväisyys ei myöskään ole automaattisesti mitattavissa vaikkakin tuotteen käytön perusteella voidaan tehdä arvioita. Metriikat kuten virhetilanteiden määrä valitulla ajanjaksolla sekä uusien vaatimusten määrä puolestaan kertyvät automaattisesti ohjelmistokehitystiimeille.

Läpinäkyvyys ja metriikat kuten tiedossa olevien vikojen määrä voivat auttaa tiimin motivoinnissa. Ketterässä ohjelmistotuotannossa luottamusta tiimiin korostetaan, joten metriikoita ei pidä käyttää ylemmän tason sanelemina valvontatyökaluina, sillä tällöin metriikoiden käyttö voi vääristyä. Vastaavanlainen ongelma havaittiin jo klassisten metriikoiden yhteydessä, kun koodin rivimäärä käytettiin metriikkana, joka johti siihen, että koodia kirjoitettiin turhan monimutkaisesti. Ketterässä ohjelmistotuotannossa metriikoita voidaan myös vääristellä esimerkiksi arvioimalla työmäärää liian suureeksi, jolloin velositeetti näyttää todellista suuremmalta. Liian kireät odotukset kehitysvauhdista voivat myös johtaa laadun heikkenemiseen, jotta vaaditut toiminnallisuudet saadaan valmiiksi. Aiemmin esimerkkeinä kuvatut ketterien tiimien käyttämät metriikat velositeetti, burndown, työmääräarviot ja testien kattavuus ovat kaikki niin sanottuja kovia metriikoita, jotka tuottavat määrällisiä arvoja. Pehmeitä metriikoita kuten motivaatio ja sitoutuminen työmäärän on kuitenkin paljon vaikeampi mitata. Sen lisäksi yksilön suoraa mittaamista tulisi välttää [Wes05].

2.4 Ketterän ohjelmistotiimin suorituskyky

Ketterien menetelmien manifesti nostaa esille tiimin sekä yksilön arvon osana ohjelmistokehitystä. Ohjelmistoprojektin onnistumiseen tai epäonnistumiseen vaikuttaa moni tekijä mutta tiimillä, joka suoriutuu hyvin, on todettu olevan positiivinen vaikutus ohjelmistoprojektien onnistumiseen [ChC08]. Tiimin jäsenten taidoilla sekä tiimin jäsenten

yhtäläisellä kokemuksella vaikuttaa olevan suurempi vaikutus tiimin työskentelyn jo vaatimusmäärittelyistä alkaen kuin ohjelmistokehityksen työkaluilla sekä menetelmillä [GCF98]. Yksi vastaus muilta aloilta sille, millä perusteella tiimi onnistuu, onkin esitetty tiimityö (engl. teamwork) [SSB05]. Organisaatiot ovat viime aikoina alkaneet ajatella tiimiä yhä tärkeämpänä organisaation osana yksilön sijaan ja tutkimus erityisesti tiimityöskentelyyn liittyen on ollut aktiivista eri toimialoilla viime vuosikymmeninä [DFD16, SSB05]. Yksi tavoitteista tutkimuksessa on tunnistaa tapoja, joiden avulla tiimit voivat parantaa työnsä lopputulosta. Ketterässä ohjelmistokehityksessä tiimityöskentelyyn ja tiimityöskentelyn tehokkuuteen liittyy vielä jokseenkin ristiriitaisia näkökulmia. Tutkimuksissa on ollut eroavia tuloksia sen osalta johtaako ketterien menetelmien seuraaminen tehokkaaseen tiimityöskentelyyn, vai ovatko ketterät menetelmät välttämättömiä mutta eivät yksinään riittävä työkalu hyvän tiimityöskentelyn saavuttamiseen [Str16]. Riippumatta siitä miten hyvään tiimityöskentelyyn päästään, on todettu, että hyvän tiimintyöskentelyn kautta myös ohjelmistotiimin suorituskyky voi parantua [LLL07].

Tiimille on annettu useita määritelmiä. Dingsøyrin ym. [DFD16] kuvaukseen pohjautuen tiimi on pieni joukko ihmisiä, joilla on toisiaan täydentäviä taitoja, jotka ovat sitoutuneet yhteiseen tavoitteeseen, asettavat tavoitteita sekä suhtautuvat työhönsä siten, että he ovat kaikki yhdessä vastuussa lopputuloksesta. Sen lisäksi tiimillä on yhteisiä jaettuja tehtäviä, jäsenet ovat sosiaalisesti tekemisissä toistensa kanssa sijoittuvat samalle tasolle organisaatiossa. Tiimien työskentelyyn ja tiimien suorituskykyyn on eri tieteenaloilla esitetty useita malleja [DiD12, DPF15]. Big Five -malli [SSB05] pohjautuu laajaan tutkimusaineistoon ja määrittää kolme koordinoivaa mekanismia, keskeinen luottamus tiimin jäsenten sisällä (engl. mutual trust), katkeamaton kommunikointi (engl. closed loop communication) sekä jaettu mielikuva omasta tiimistä ja tavoitteista (engl. shared mental models). Koordinoivat mekanismit ovat läsnä tehokkaassa tiimityössä. Nämä kolme koordinoivaa mekanismia puolestaan mahdollistavat viisi tiimityötä tukevaa komponenttia, tiimin johtajuus (engl. team leadership), sopetumiskyky (engl. adaptability), yhteinen suorituskyvyn seuranta (engl. mutual performance monitoring), tukitoimet (engl. back-up behavior) sekä tiimiin asennoituminen (engl. team orientation).

Verrattaessa mallia ketterään ohjelmistokehitykseen ja ohjelmistotiimiin, nousee esimerkiksi sopeutumiskyky ketterissä menetelmissä esille sitä kautta, että vaatimusten oletetaan muuttuvaan ja reagointi muutoksiin täytyy ketteriltä tiimeiltä sujua. Samoin yhteinen suorituskyvyn seuranta tukee ketterää ajattelumallia näkyvyyden parantamisesta kaikkien

ohjelmistoprojektin sidosryhmien osalta. Big Five -mallissa onkin paljon piirteitä, jotka soveltuvat hyvin ketterään ohjelmistokehitykseen ja ketteriin ohjelmistotiimeihin mutta mallin esittämä tapa, miten tiimin johtajuus toteutetaan, risteää itseohjautuvien tiimien kanssa [DiD12, Str16]. Mallissa esitetään johtajuutta perinteisemmin ylhäältä päin saneltuna, joka ei tue ketterän ohjelmistokehityksen periaatetta siitä, että tiimissä on yhdenvertaisia jäseniä. Big Five -mallin yhteenveto on koottu taulukkoon 1.

Taulukko 1: Big Five -mallin yhteenveto [SSB05]

OSATEKIJÄ	ROOLI	KUVAUS
KESKEINEN LUOTTAMUS	Koordinoiva ja mahdollistava	Jaettu luottamus, että jokainen tiimin jäsen suojelee tiimiläisten etua sekä toimii oman roolinsa mukaisesti.
KATKEAMATON KOMMUNIKOINTI	Koordinoiva ja mahdollistava	Tiedon kulku jäsenten välillä tiedonsiirto- ja kommunikointitavasta riippumatta.
JAETUT MIELIKUVAT	Koordinoiva ja mahdollistava	Yhteinen ymmärrys tiimin tavoitteista, jäsenten tehtävistä sekä tiimin tavoitteista.
TIIMIN JOHTAJUUS	Tiimityötä tukeva	Kyky johtaa ja tukea tiimiläisiä, koordinoida ja jakaa tehtäviä, suunnitella ja motivoida tiimin työskentelyä sekä seurata suorituskykyä ja mahdollistaa osaamisen kehittäminen.
SOPEUTUMISKYKY	Tiimityötä tukeva	Kyky sopeutua ja muokata tiimin toimintaa sekä jakaa resursseja uudelleen opitun perusteella.
YHTEINEN SUORITUSKYVYN SEURANTA	Tiimityötä tukeva	Yhteinen käsitys olosuhteista, joissa tiimi toimii ja miten suorituskykyä tulisi olosuhteissa tarkalleen seurata.
TUKITOIMET	Tiimityötä tukeva	Kyky jakaa työtaakkaa sekä ennakoida muiden tiimiläisten tarpeita heidän vastuualueensa perusteella.
TIIMIIN ASENNOITUMINEN	Tiimityötä tukeva	Muiden tiimiläisten huomioiminen sekä tiimin edun asettaminen oman edun edelle.

Muita malleja, joiden soveltuvuutta tiimityöskentelyyn ketterässä ohjelmistotuotannossa on tutkittu ovat mm. Dickinsonin ja McIntyren malli [DiD12] sekä jaettujen mielikuvien malli (engl. shared mental models) [YuP14]. Dickinsonin ja McIntyren tiimityöskentelymalli on suunniteltu itseohjautuville tiimeille ja se käsittelee tiimityötä oppimissilmukkana (engl. learning loop). Mallilla on useita samoja komponentteja Big Five -mallin kanssa mutta tiimin johtajuus on jaettua.

Dingsøyr ym. [DFD16] ovat puolestaan koonneet yhteen viisi ketterien ohjelmistotiimien suorituskykyyn positiivisesti vaikuttavaa tekijää muilla aloilla tunnistetuista tiimin suorituskykyyn vaikuttavista tekijöistä. Tämän jälkeen he ovat vertailleet näitä ketterien menetelmien periaatteisin. Taulukkoon 2 on koottu tunnistetut suorituskykyyn vaikuttavat

tekijät, sekä niiden vertautuminen ketterien menetelmien periaatteisiin. Tutkijat toteavat, että erityisesti se miten itseohjautuvuus vaikuttaa ohjelmistotiimin työskentelyyn ja ohjelmistoprojektin onnistumiseen vaatii jatkotutkimusta mutta uskovat ketterien periaatteiden linkityksen muihin tiimin suorituskykyä tukeviin tekijöihin kuvaavan hyvin, mitä hyötyjä joistain ketteristä periaatteista on tiimitasolla.

Taulukko 2: Tiimin suorituskykyyn vaikuttavat tekijät [DFD16]

OSATEKIJÄ	KETTERÄT PERIAATTEET	YHTENEVÄISYYS KETTERÄÄN OHJELMISTOTUOTANTOON
TIIMIN KOORDINAATIO	Ohjelmistoa tulisi toimittaa usein, lyhyet iteraatiot korostavat koordinoinnin tarvetta.	Esimerkkinä Scrumin päivittäiset palaverit, iteraatioiden suunnittelu sekä retrospektiivit tukevat tiivistä koordinoitia tiimin sisällä. Sprintin jälkeinen tuotelisäys asiakkaan suuntaan mahdollistaa asiakkaan vaikutaa projektiin tasaisesti.
TAVOITESUUNTAUTUNEISUUS	Toimiva tuote on edistymisen pääasiallinen mittari, itseohjautuvat tiimit.	Itseohjautuvuuden osalta ei ole selkeää näyttöä, miten se vaikuttaa suorituskykyyn. Ketterät menetelmät eivät myöskään ohjeista miten itseohjautumisessa huomioidaan projektin tavoitteet.
TIIMIN YHTEENKUULUVUUS	Eri sidosryhmien ja kehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.	Käytännöt kuten pariohjelmointi ja jaettu koodin omistajuus voivat tukea yhteenkuuluvuutta. Samoin sprintin suunnittelu sekä retrospektiivit ja päivittäiset palaverit tukevat eri sidosryhmien yhdessä tekemistä.
JAETTUJEN MIELIKUVIEN MALLIT	Tehokkain tapa välittää tietoa kehitystiimin sisällä on keskustelu kasvokkain.	Ketterät menetelmät itsessään ovat voimakkaita jaetun mielikuvan malleja. Selkeät käytänteet ohjeistavat kaikkia kehittäjiä selkeästi siitä, miten tehtäviä tulee suorittaa ja mikä on tavoite.
TIIMIN OPPIMINEN	Tiimi tarkastelee ja muokkaa toimintaansa säännöllisesti toimiakseen tehokkaammin	Retrospektiivit iteraatioiden päätteeksi tukevat prosessin kehitystä, pariohjelmointi sekä suunnittelu tukevat tiimin osaamista.

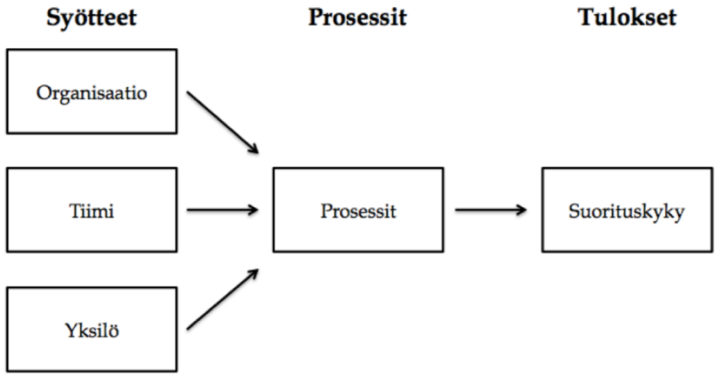
Yllä oleva taulukko lisää tukea aiemmin esitetyn Salasin Big Five -mallin lisäksi sille, että tiimityöskentelystä muilta aloita ja ketteristä menetelmistä löytyy yhtäläisyyksiä. Muiden tieteenalojen mallien soveltuvuus ketterään ohjelmistokehitykseen vaatii kuitenkin lisää tutkimusta [DiD12, Str16].

Suorituskykyä sekä suoriutumista kuvaa englanninkielinen termi ”performance”. Suoriutumisella tarkoitetaan sitä, miten tiimi on onnistunut vastaamaan projektin tavoitteisiin kuten aikatauluun, laatuun sekä kustannuksiin. Suorituskyky puolestaan jaetaan usein kahteen osaan, tehokkuuteen (engl. efficiency), joka tarkoittaa tavoitteen täyttämistä mahdollisimman nopeasti ja pienellä resurssimäärällä, sekä vaikuttavuuteen (engl. effectiveness), joka viittaa oikeiden ja arvokkaimpien tavoitteiden täyttämiseen [FIK15]. Suoriutuminen on tilanneriippuvaista, eli useampi tiimi voi suoriutua samasta tehtävästä

hyvin. Suorituskyky on jotain mikä tiimillä on ja mitä tiimi voi pyrkiä kehittämään. Ketterässä ohjelmistokehityksessä vaatimusten ja tavoitteiden vaihtuessa, myös suorituskyvyn mittarien, joiden avulla suorituskykyä esitetään ja metriikoiden, joiden avulla suorituskykyä mitataan, tulisi sopeutua. Ympäristö missä tiimi suoriutuu, on erittäin dynaaminen ja markkinatilanne saattaa vaihtua nopeasti. Kun alkuperäinen projektin tavoite muuttuu, eivät ennen projektin alkua määritetyt tavoitteet välttämättä enää kuvaa onnistunutta projektia. Myöskään tiimin aiempien projektien kautta tiedossa olevan suorituskyvyn pohjalta ei voida ennustaa projektien onnistumista [FIK15]. Osaa IT-projekteista vaivaa edelleen budjettien ja aikataulujen venyminen ja tiimin hyvä suorituskyky saattaa olla merkittävä osatekijä projektin onnistumiseksi [DPF15]. Aiemmin tässä luvussa mainittiin, että tiimillä, jossa tiimityöskentely toimii, näyttää olevan positiivinen vaikutus tiimin suorituskykyyn ja tätä kautta projektien onnistumiseen. Tiimien suorituskyvyn tutkimuksessa on alettu puhua huipputuottavista ohjelmistotiimeistä (engl. high performing software team). Yksi tulkinta huipputuottavalle ohjelmistotiimille tarkoittaa tiimiä, joka ylittää kaikki odotukset sekä tuottaa erittäin laadukkaita tuloksia [DPF15]. Huipputuottavia tiimejä kuvaillaan sellaisiksi, joiden jäsenet luottavat toisiinsa, joissa jäsenillä on yhteinen näkemys tavoitteista sekä joissa johtovastuuta on jaettu. Näissä tiimeissä myös kiinnitetään huomiota muiden jäsenten kehittymiseen ja onnistumisiin mikä lisää tiimin sitoutumista toisiinsa. Huipputuottavien tiimien toistamista on yritetty esimerkiksi panostamalla työntekijöiden osaamiseen ja osallistamalla kaikkia tiimin jäseniä päätöksentekoon mutta toimivaa kaavaa ei ole löydetty [DPF15]. Tästä huolimatta organisaatiot näkevät huipputuottavat tiimit arvokkaina ja pyrkivät toistamaan niitä [FIK15].

Yksi tapa mitata ohjelmistotiimin suorituskykyä on ohjelmistoprojektien tulosten kautta. Projektin tuloksia ovat esimerkiksi kehitetyn järjestelmän laatu, pysyikö kehitysprojekti sallituissa kustannuksissa ja haluaako tiimi myös jatkossa työskennellä yhdessä. Projektin tuloksia ja tiimin aikaansaannoksia seuratessa voi kuitenkin esiintyä sidosryhmästä riippuvia erimielisyyksiä siitä, kuinka hyvin tiimi on suoriutunut ja miten hyvin lopputulos vastaa odotuksia [FIK15, GCF98]. Syöte-Prosessi-Tulos (engl. Input-Process-Output, IPO) on yleisesti käytetty kehys tiimin suorituskyvyn tutkimiseen [MMR08]. Kehystä on useasti muokattu ja päivitetty alkuperäisestä kuvaamaan paremmin sovellusympäristöään tai huomioimaan että tiimien työskentely ei aina etene lineaarisesti alusta loppuun. Yhdessä versioista syöte on jaettu kolmeen tasoon, organisaatioon, tiimiin sekä yksilöön. Erilaiset syötteet vaikuttavat tiimiin, joka omien prosessiensa kautta muodostaa

lopputuloksia. Yksilötasolla tiimin syötteisiin vaikuttavat etenkin tiimin jäsenten pätevyys sekä persoonat, tiimitasolla tehtävien hallinnoinnilla ja tiimin johtajuudella on vaikutusta [MMR08]. Organisaation tasolla tiimin suorituskykyyn vaikuttavia syötteitä ovat mm. organisaation rakenne sekä työympäristö. Kolmen tason syötteet yhdessä säätelevät ja rajoittavat tiimin vuorovaikutusta keskenään. IPO-malli on esitetty kuvassa 4.



Kuva 4: IPO-malli [MMR08]

Prosessit ovat tärkeässä roolissa sillä ne määrittävät kuinka syötteet muuttuvat tuloksiksi. Tulosten kautta voidaan tarkastella tiimin prosessien seurauksia, suorituskyky on yksi näistä. Ohjelmistoalalla tiimin suorituskyvyn mittaamista on ehdotettu tehtävän kahdella tasolla, tiimin prosessien ja tiimin työn tulosten kautta [LLL07]. Prosesseja mitattaessa keskitytään erilaisiin metriikkoihin, joiden avulla voidaan seurata, kuinka hyvin ohjelmistokehitysprosesseja noudatettiin. Tällaisia metriikoita, joilla voidaan seurata itse prosessin onnistumista ovat esimerkiksi tiedossa olevien vikojen kasvu sallituissa rajoissa tai testikattavuuden säilyminen tavoitellulla tasolla. Myös tiimin oma kokemus projektista on osa tuloksia.

Arvioitaessa ohjelmistotiimin suorituskykyä valmistuneen työn osalta, tarkastellaan projektin lopputulosta taloudellisesti ja loppuasiakkaan näkökulmasta. Sitä onko projekti onnistunut vai ei voidaan arvioida neljän eri näkökulman kautta [ChC08]:

- Laatu, onko tuote tai projektin lopputulos hyvä.
- Laajuus, ovatko kaikki vaatimukset ja tavoitteet saavutettu.
- Aika, onko työ valmistunut ajallaan.
- Kustannukset, onko työn toimitus onnistunut arvioitujen kustannusten ja työmäärän puitteissa.

Mitattaessa ohjelmistotiimin suorituskykyä kahdella tasolla, tulosten sekä prosessien

kautta, ei voida selkeästi jakaa kumpi mittaustapa kertoisi enemmän suorituskyvyn vaikuttavuudesta ja kumpi enemmän suorituskyvyn tehokkuudesta. Se onko suorituskyvyn tulkinta enemmän tehokkuuteen vai vaikuttavuuteen rinnastettavaa, näyttää riippuvan metriikasta, jonka pohjalta suorituskykyä arvioidaan. Ohjelmistoalan ammattilaisilla on oma käsityksensä suorituskyvystä erilaisissa tilanteissa, ja he ovat tärkeässä roolissa arvioitaessa projektien onnistumista ja tavoitteiden täyttymistä yhdessä muiden sidosryhmien kanssa [FIK15].

Yleisellä tasolla ohjelmistotiimin suorituskykyyn projektien aikana vaikuttavat tekijät voidaan jakaa neljään tasoon [SFP11]:

- Tekniset tekijät, kuten ohjelmakoodin laatu, tuotteen vaatimukset, tiimin ohjelmistokehitysprosessit.
- Ei-tekniset tekijät, kuten tiimin johtaminen sekä tiimin ryhmähenki.
- Organisaatiotason tekijät, kuten palkitsemisjärjestelmät tai organisaation hajautuneisuus useisiin eri toimipisteisiin.
- Työympäristön tekijät, kuten välit asiakkaan kanssa sekä markkinan kilpailutilanne.

Teknisiä tekijöitä pystytään seuraamaan kovien, määrällisiä arvoja tuottavien metriikoiden avulla. Ei-tekniset tekijät sen sijaan ovat pehmeitä metriikoiden näkökulmasta, joiden mittaaminen on haastavaa. Vaikka tiimin käytöksen, tiimityön laadun ja tiimin kognition mittauksessa on edistytty muilla aloilla, ohjelmistoalalle kaivataan edelleen mittauskeinoja, joita voidaan soveltaa käytännössä tiimin mittauksessa [DiD12]. Sudhakar ym. ovatkin pyrkineet korostamaan nimenomaisesti pehmeiden tekijöiden vaikutusta tiimin suorituskykyyn ja pehmeillä tekijöillä on tunnistettu olevan iso rooli, kun tiimit yrittävät parantaa omaa suorituskykyään [DiD12, FIK15].

Vastaavasti yhdeksi kehitystiimin suorituskykyä laskevaksi tekijäksi on arveltu kehittäjien suoraa mittausta [KMI15]. Kehittäjän suoraa mittausta on esimerkiksi yksittäisen tiimin jäsenen tuottavuuden arviointi jonkun metriikan, kuten valmistuneiden työtehtävien määrän perusteella. Ketterät menetelmät olettavat tiimien kehittävän itse toimintaansa, jolloin yksilötason metriikat eivät ole oleellisia koko tiimin tuloksia tarkasteltaessa. Myös ketterien menetelmien vastaisen toiminnan on huomattu haittaavan tiimin suorituskykyä. Tehtävien toteutus väärässä järjestyksessä, kommunikaation puute sekä oppimismahdollisuuksien heikko hyödyntäminen voivat kaikki laskea tiimin suorituskykyä [SMD11].

Ketterät menetelmät korostavat korkealle priorisoitujen, eniten arvoa asiakkaalle tuottavien toiminnallisuuksien toteutusta mutta kehittäjät valitsevat välillä itselleen mieluisia tehtäviä prioriteetin sijaan. Ketterissä organisaatioissa saatetaan myös tehdä päätöksiä johtotasolla keskustelematta asiasta tiimin kanssa mikä on ketterän yhteistä päätöksentekoa ja kommunikointia vastaan. On myös havaittu, että päivittäisissä palavereissa tiimi ei raportoi ongelmia keskenään, ainoastaan tiimin vetäjälle [SMD11]. Retrospektiivit tarjoavat iteraatioiden päätteeksi tiimille tilaisuuden arvioida omaa työtään mutta on havaittu, että ongelmien tunnistamisesta huolimatta korjaustoimia ei onnistuta ottamaan käyttöön, jolloin oppimismahdollisuus jää hyödyntämättä.

2.5 Yhteenveto kirjallisuuskatsauksesta

Erilaisia metriikoita on ollut osana ohjelmistotuotantoa kauan. Metriikat antavat tyypillisesti tietoja takautuvasti, eli ohjelmistoa on täytynyt kehittää, jotta mittaustuloksia tulee saataville. Haasteena on ollut löytää metriikoita, joiden avulla ohjelmistoprojektien vaatimuksia sekä kustannuksia voidaan arvioida riittävän luotettavasti etukäteen. Metriikoita voidaan ottaa käyttöön systemaattisesti esimerkiksi Maali/Kysymys/Metriikka prosessin avulla. Metriikoiden valinta ei aina onnistu ja metriikoiden käyttössä voi ilmeitä haasteita tulosten väärinymmärryksen tai metriikoiden väärinkäytön takia.

Metriikoiden käyttö on yleistä ketterässä ohjelmistotuotannossa mutta useat metriikat vaativat edelleen ohjelmistotiimiltä tuotteen kehitystä, jotta tuloksia metriikoiden avulla voidaan tarkastella. Iteratiivisen kehityksen ansiosta kerättyjä mittaustuloksia voidaan kuitenkin hyödyntää seuraavien iteraatioiden suunnittelussa, jolloin metriikat osittain palvelevat tarvetta antaa luotettavaa tietoa ohjelmistoprojektin edistymisestä ja kustannuksista etukäteen.

Ohjelmistotiimin suorituskykyä voidaan seurata erilaisten metriikoiden avulla. Muiden alojen tiimien ja tiimityöskentelyn malleista on löydetty käytäntöjä, joiden avulla voidaan kehittää tiimityöskentelyä ja tiimien suorituskykyä myös ketterässä ohjelmistotuotannossa. Yhtenä vauhdittajana ohjelmistotiimien suorituskyvyn sekä itse tiimin prosessien tutkimuksessa ovat olleet havainnot siitä, että ohjelmistotiimin tiimityöskentelyn vaikutus ohjelmistoprojektien onnistumiseen on suuri. Ohjelmistoprojektin onnistumista arvioidaan tiimin lisäksi muiden sidosryhmien toimesta. Keskeisiä onnistumisen kriteerejä ovat edelleen aikataulussa pysyminen, vaatimusten täyttyminen sekä kustannusten säilyminen sovitulla tasolla.

3 Empiirisen tutkimuksen toteutus

Tätä tutkielmaa varten kerättiin tutkimusaineistoa sekä julkisen että yksityisen sektorin yrityksistä. Empiirisen tutkimuksen tavoitteena on löytää vastauksia tutkielman luvussa 1 esitettyihin tutkimuskysymyksiin. Tutkimuksen ei ole tavoite antaa vastausta, jonka perusteella metriikoiden ja mittauksen käyttöä voitaisiin yleistää ketterissä ohjelmistotiimeissä vaan selvittää miten erityisesti tutkimukseen osallistuneissa tiimeissä mittausta tehdään ja miten olemassa oleva tutkimustieto, jota on esitetty kirjallisuuskatsauksessa luvussa 2, ilmenee kentällä.

Alaluvussa 3.1 esitellään tässä tutkielmassa käytetty empiirinen tutkimusmenetelmä. Alakuku 3.2 kuvaa miten tutkimus sekä tutkimusaineiston keräys toteutettiin. Tutkimusaineiston analysointi on käyty läpi alaluvussa 3.3.

Tutkimukseen osallistuneet yritykset sekä vastaajien roolit omissa yrityksissään on esitelty seuraavaksi. Vastaajat ovat tutkielman laatijalle entuudestaan tuttuja ja valikoituivat mukaan sen perusteella. Tutkielman laatija työskentelee itse toisessa yrityksistä. Lupa yrityksen nimen käyttöön tutkielmassa saatiin molemmilta haastatelluilta.

Experq Oy on yksityisen sektorin kuuden hengen yritys (Y-tunnus: 2395765-7), jonka päätoimiala on TOL2008 mukaisesti 62010 Ohjelmistojen suunnittelu ja valmistus. Yrityksen omaa ohjelmistotuotetta käytetään mm. terveydenhuolto- sekä kiinteistöalalla. Haastateltu henkilö työskentelee osana yrityksen ohjelmistokehitystiimiä.

Tutkimukseen osallistui julkiselta sektorilta Ruokavirasto (Y-tunnus: 2911686-7). Vaikka yritys ei TOL2008 toimialansa 84130 Työvoima- ja elinkeinoasiain hallinto puolesta ole ohjelmistoyritys, on yrityksellä kuitenkin omaa sisäistä digitaalisten palveluiden kehitystä. Haastateltu henkilö työskentelee tuoteomistajien tukena ja linkkinä kehitystiimiin, sillä tuoteomistajat ovat tyypillisesti muun kuin ohjelmistoalan henkilöitä, joilta vaaditaan substanssiosaamista omasta aihealueestaan (esimerkiksi elintarvikkeiden testaus), jonka tueksi digitaalisia palveluita ja työkaluja kehitetään. Yrityksessä on useita ohjelmistokehitystiimejä ja nykyinen tiimi, jossa haastateltu henkilö työskentelee, on muodostettu alkuvuodesta käynnistetyn ohjelmistoprojektin ajaksi.

3.1 Tutkimusmenetelmä

Tutkielman empiirisenä tutkimusmenetelmänä käytettiin laadullista tapaustutkimusta.

Tapaustutkimus toteutetaan ilmiöiden omassa ympäristössä, mistä seuraa se, ettei tutkimus ei ole säädelty ja tutkimusaineiston keräys ja havainnointi tapahtuu luontaisessa ympäristössä. Toisaalta esimerkiksi ohjelmistotiimin tuominen eristettyyn tutkimusympäristöön saattaisi vaikuttaa tiimin toimintaan, joka puolestaan saattaisi vaikuttaa tiimin työskentelyyn ja erilaisten ilmiöiden näkyvyyteen. Tapaustutkimus soveltuu hyvin tutkimukseen, jonka tavoite on tutkia ja selvittää mitä todellisuudessa tapahtuu jonkin ilmiön ympärillä [RuH09]. Tapaustutkimuksia voidaan käyttää tutkimaan, kuvailemaan sekä selittämään ilmiöitä. Myös tutkittavan ilmiön joitain ominaisuuksia voidaan kehittää tapaustutkimuksen avulla, jolloin ollaan jo lähempänä toimintatutkimusta. Tämän tutkielman ensisijainen tavoite on keskittyä tutkimuskysymysten avulla selvittämään ja tutkimaan mitä todellisuudessa tapahtuu.

Tapaustutkimukset ovat luonteeltaan usein laadullisia tutkimuksia [RuH09]. Laadullinen tutkimusaineisto sisältää mm. sanallisia kuvauksia ja kaavioita, joiden avulla tuloksia pyritään muodostamaan. Laadullinen data on rikkaampaa ja sen avulla on mahdollista saada syvällisempi ymmärrys aiheesta. Laadullinen tapaustutkimus ei välttämättä ole tilastollisessa mielessä pätevää ja kritiikki liittyy usein siihen, että tuloksia on vaikea yleistää, tutkimusta on vaikea toistaa sillä tutkimus ei ole kontrolloitua ja tutkijan näkökulmalla voi olla vaikutusta. Yksi keino vastata kritiikkiin on soveltaa tutkimusmenetelmiä ja tapoja täsmällisesti. Esimerkiksi triangulaatio (engl. triangulation) on tärkeää empiirisen tutkimuksen tarkkuuden parantamiseksi. Tietoa voidaan kerätä useista eri lähteistä tai samasta lähteestä tutkimuksen eri vaiheissa, käyttää useita tarkkailijoita sekä useita tiedonkeräysmenetelmiä aineiston keräysvaiheessa.

3.2 Tutkimuksen toteutus sekä tiedonkeräys

Laadullisen tutkimusaineiston keräystavaksi valittiin puoliavoin haastattelu. Puoliavoimessa haastattelussa haastattelijalla on tukena haastattelun runko (Liite 1), jonka avulla hän pyrkii keräämään vastauksia haastattelunsa kysymyksiin ja teemoihin [NHW15]. Haastattelun runko pyritään jäsentämään siten, että siirtymineen eri aihealueista eteenpäin sujuu luontevasti. Avointen kysymysten lisäksi myös satunnaiset tarkat kysymykset voivat palvella haastattelijan tavoitteita hyvin, sillä yksinkertainen kysymys kuten ”*Onko mielestäsi mittauksesta hyötyä?*” avaan useita syventävien jatkokysymysten vaihtoehtoja. Avoin haastattelu mahdollistaa kunkin haastateltavan kohdalla sen, että haastattelun ja keskustelun voi antaa vapaasti myös välillä poiketa käsikirjoituksesta, jolloin tietoa

saadaan laajemmin haastattelun rungon ympäriltä. Myös se, että haastateltavan annetaan vapaammin kertoa käsikirjoituksen ulkopuolelta voi avata uusia näkökulmia tutkijalle. Täsmälliset kyselytyyliset haastattelut, joita toki on helppo toistaa laajallekin kohderyhmälle, rajaavat pois tiedon saantia asioista, joita ei ole ennalta päätetty kysyä [NHW15]. Avoin haastattelu toki vaatii enemmän myös itse haastattelun suorittajalta, jotta haluttuihin teemoihin ja kysymyksiin saadaan vastaus. Avoimet haastattelut ovat myös työmäärältään suuria, etenkin aineiston muuntamiseksi tekstiksi on syytä varata riittävästi aikaa.

Tiedonkeruun tekniikat voidaan jakaa kolmella tasolla ja haastattelu sijoittuu näistä ensimmäiselle tasolle. Ensimmäisellä tasolla tutkija on suoraan tekemisissä tutkimukseen osallistuvien kanssa mikä mahdollistaa vaikuttamisen tutkimusaineiston muodostumiseen ja laatuun [RuH09]. Toisella tasolla tiedonkeruu on epäsuoraa esimerkiksi videoitujen tilanteiden havainnointia. Kolmannella tasolla tutkimusaineisto on usein valmiiksi saatavilla analysointia varten mutta aineisto on kerätty alun perin jotain muuta kuin kyseistä tutkimusta varten.

Tämän tutkielman haastattelut järjestettiin maaliskuussa 2020 käyttäen Microsoft Teams ohjelmistoa haastatteluihin. Haastateltavat henkilöt olivat entuudestaan haastattelijan tuntemia, joten kutsut ja haastatteluaikataulut järjestettiin vapaamuotoisella mobiiliviestinnällä. Ennen haastattelua tutkielman aihealue oli mainittu haastateltaville mutta valmistautumisohjeita ei ollut annettu haastateltaville. Haastattelut nauhoitettiin haastateltavan luvalla, jotta haastattelija pystyi haastattelun aikana paremmin keskittymään siihen mitä haastateltava kertoo ja ohjaamaan keskustelua eteenpäin. Haastattelun runko (Liite 1) pohjautuu luvussa 1 esitettyihin tutkimuskysymyksiin mutta kysymyksiä ei suoraan esitetty haastateltaville. Haastattelut oli jaettu seuraaviin teemoihin:

- Ohjelmistokehitysmenetelmät ja metriikat (tutkimuskysymys 1).
- Metriikoiden seuranta (tutkimuskysymys 2).
- Metriikoiden vaikutukset (tutkimuskysymys 3).

Haastattelun jälkeen kopio tallenteesta lähetettiin myös vastaajalle, jotta tämä voi tarvittaessa korjata tai täydentää vastauksiaan. Varsinaisen haastattelun jälkeen ei tullut täydennyksiä tutkimusaineistoon.

3.3 Tutkimusaineiston analysointi

Laadullisen tutkimusaineiston analysoinnissa on useita muodollisuuden tasoja.

Tapaustutkimukselle tyypillinen taso on joko muokkaava lähestymistapa tai toimintamalliin perustuva lähestymistapa [RuH09]. Muokkaavassa lähestymistavassa suurin osa aineiston luokittelusta tapahtuu analysoinnin aikana, kun taas toimintamalliin perustuvassa lähestymistavassa luokittelu on muodollisempaa ja sitä on tehty osittain jo etukäteen tutkimuskysymysten avulla. Tässä tutkielmassa käytettiin muokkaavaa lähestymistapaa ja tutkimusaineisto luokiteltiin analysoinnin yhteydessä.

Tutkielmaa varten kerätyn tutkimusaineiston käsittely aloitettiin kirjoittamalla haastattelut tekstiksi. Tekstiaineistoa tiivistettiin kirjoittamalla haastattelijan puheenvuoroista lyhyesti haastattelijan kommentit sekä johdattelu seuraavaan aiheeseen. Haastateltavan puheenvuoroista tekstissä karsittiin pois täytesanoja mutta suorissa lainauksissa nauhoitteesta tarkistettiin sanasta sanaan mitä haastateltava oli sanonut. Tämän jälkeen tekstimuotoinen tutkimusaineisto analysoitiin induktiivisella eli aineistolähtöisellä tavalla. Induktiivisessa analysoinnissa tutkimustulokset muodostetaan aineiston perusteella. Tutkijan tavoite on antaa aineiston kuvata tutkittavaa ilmiötä ilman ennakkokäsityksiä ja liikoja ennakkoon tehtyjä hypoteeseja mitä aineistosta löytyy [RuH09].

Tekstimuotoinen tutkimusaineisto jaettiin koodeihin käyttäen apuna ATLAS.ti ohjelmistoa. Aineiston koodauksen tavoite on auttaa tutkijaa käsittelemään tutkimusaineistoa kokonaisuutena. Ensimmäisellä koodauskierroksella aineiston yksittäisiin lauseisiin tai kappaleisiin merkittiin yksi tai useampi aihealue koodien avulla. Koodit ovat yleisiä kuvauksia mitä aineistossa tapahtuu ja pyrkivät kertomaan mistä koodatussa osiossa puhutaan. Ensimmäisen koodauskierroksen tulos on esitetty Liitteessä 2.

Toisella koodauskierroksella ensimmäisen kierroksen koodauksia tarkennettiin, useimmiten pilkkomalla ensimmäisen kierroksen koodattuja osioita pienempiin osiin ja liittämällä näihin koodeja. Ensimmäisen kierroksen jälkeen haastattelutekstin yksi puheenvuoro oli tyypillisesti koodattu muutamalla koodilla mutta toisen kierroksen jälkeen koodit kohdistuivat tarkemmin sanoihin sekä lauseisiin. Osa ensimmäisen kierroksen koodeista tarkennettiin yleisen tason termistä tarkempaan termiin. Esimerkkinä tällaisesta tarkennuksesta ensimmäisen ja toisen kierroksen välillä, oli yleisen koodin ”metriikoita” jakaminen kahdeksi tarkemmaksi koodiksi ”testikattavuus” sekä ”buildin tila”. Ensimmäisen kierroksen koodauksia ei poistettu, vaan tutkimusaineistoon jäi erikseen nähtäville ensimmäisen ja toisen koodauskierroksen merkinnät. Toisen koodauskierroksen tulokset ovat myös esitetty Liitteessä 2. Koodauskierrosten jälkeen tutkimusaineistosta muodostettiin seuraavassa luvussa 4 esitetyt tutkimustulokset.

4 Tutkimustulokset

Tässä luvussa käsitellään tutkimuksen tuloksia tapaustutkimusaineiston pohjalta. Luku on jaettu alalukuihin tutkimushaastattelun teemojen mukaisesti. Alaluvussa 4.1 esitellään tapaustutkimusaineistosta tunnistetut ohjelmistokehitysmetriikat. Alaluvussa 4.2 kerrotaan miten erilaisia metriikoita ja mittareita tutkimukseen osallistuneissa tiimeissä seurataan. Metriikoiden ja mittauksen vaikutuksia on käsitelty alaluvussa 4.3. Viimeisessä alaluvussa 4.4 käsitellään esiin nousseita havaintoja, jotka eivät suoraan sovi muiden teemojen alle. Tutkimukseen osallistuneiden luvussa 3 esiteltyjen kahden yrityksen vastaajia merkitään seuraavasti:

- V1 Experq Oy:n haastateltu henkilö.
- V2 Ruokaviraston haastateltu henkilö.

4.1 Ohjelmistokehitysmenetelmät ja metriikat

Molemmissa tutkimukseen osallistuneissa tiimeissä käytettiin ohjelmistokehitykseen ketteriä menetelmiä. V2 kertoi tiimin käyttävän Scrumia ja haastattelusta esiin nousseet ohjelmistokehityksen käytänteet sekä erilaiset roolit ml. Scrum Master tiimissä antavat luotettavan kuvan siitä, että menetelmää pyritään seuraamaan tarkasti. Tiimi kehittää tällä hetkellä uutta ohjelmistotuotetta. V1 kuvasi, että ohjelmistokehitys on ketterää mutta tarkkaa määritystä ei voi antaa. Ohjelmistokehitys on iteratiivista ja inkrementaalista ja tiimillä on käytössä oma sekoitus Scrumia, Kanbania ja XP:n periaatteita. Yrityksen ohjelmistotuote on tällä hetkellä tuotantokäytössä ylläpitovaiheessa, ohjelmistokehitys on kevyttä ja osa teknisen tiimin resursseista on kiinni ylläpidossa ja tukitoimissa.

Tunnistetut metriikat kerättiin tekstimuotoisesta tutkimusaineistosta perehtymällä tekstin osiin, jotka oli merkitty koodilla metriikoita tai itse metriikan nimellä. Yhteenveto tunnistetuista metriikoista on koottu taulukkoon 3. Metriikan yhteydessä on mukana tieto haastattelusta, jossa metriikka tuli ilmi. Sen lisäksi metriikat ovat luokittelu ohjelmistotuotannon elinkaaren vaiheisiin sen perusteella, miten haastateltavat kuvasivat niiden käyttöä. Jotkin metriikat esiintyivät useassa eri ohjelmistotuotannon vaiheessa. Ohjelmistotuotannon elinkaaren vaiheita on yhdistelty siten, että vaihe ennen tuotannon aloitusta sisältää vaatimusten määrittelyn sekä ohjelmiston suunnittelun, tuotannon aikaiseen vaiheeseen kuuluu ohjelmiston kehitys sekä testaus ja ylläpitovaiheeseen kuuluu käyttöönotto sekä ylläpito.

Taulukko 3: Tunnistettujen metriikoiden yhteenveto

	OHJELMISTOTUOTANNON VAIHE		
TUNNISTETTU METRIIKKA	ENNEN TUOTANNON ALOITUSTA	TUOTANTO	YLLÄPITO
BUILDIN TILA		V1	
KÄYTÖSSÄ OLEVAT RESURSSIT	V2	V2	
LAADUNVARMISTUS		V1	V1, V2
LOPPUKÄYTTÄJÄN TYYTYVÄISYYS		V1	V1
MUUTOSTEN MÄÄRÄ SPRINTIN AIKANA		V1	
PROJEKTIN KESKENERÄISTEN TYÖTEHTÄVIEN MÄÄRÄ		V2	
SPRINTIN JÄLJELLÄ OLEVA TYÖMÄÄRÄ		V1, V2	
SPRINTIN TAVOITE		V1, V2	
TESTIEN MÄÄRÄ		V1, V2	
TESTIEN RIVIKATTAVUUS		V1	
TIEDOSSA OLEVIEN VIRHEIDEN MÄÄRÄ		V1	V1
TYÖMÄÄRÄARVIO	V2	V1, V2	V1
TYÖMÄÄRÄARVIO JA TOTEUTUNEEN VERTAILU		V1, V2	
TYÖN PISTEMÄÄRÄ	V2	V2	
VELOSITEETTI	V2	V1, V2	

Tunnistetuista metriikoista työmääräarvio on ainoa, joka ilmeni kaikissa ohjelmistotuotannon vaiheissa. V1:n yrityksen tuote on ylläpidossa mutta uusia ominaisuuksia ja tuotekehitystä tehdään jatkuvasti, joten työmäärää arvioidaan myös ylläpitovaiheessa. Metriikoita tunnistettiin yhteensä 15, jokainen metriikoista oli haastattelujen pohjalta käytössä ainakin tuotekehityksen vaiheessa. Kahdeksan metriikkaa ilmeni käytön perusteella vain yhdessä vaiheessa ohjelmiston elinkaarta, tuotannossa. Kuusi metriikkaa, jotka ilmenivat kahdessa ohjelmistotuotannon vaiheessa kolmesta jakautuivat tasan tuotannonaikaisen vaiheen lisäksi joko vaiheeseen ennen tuotannon aloitusta tai ylläpitovaiheeseen. Metriikoita ilmeni V1:n haastattelussa 12 ja V2:n haastattelussa puolestaan 10.

4.2 Metriikoiden seuranta

Molemmat tutkimukseen osallistuneet vastaajat kuvasivat, että metriikoita ja mittaustuloksia seurataan metriikasta riippuen myös varsinaisen kehitystiimin ulkopuolella. V1 kuvasi, että yrityksen ollessa pieni, osallistuu koko yrityksen henkilöstö esimerkiksi sprinttien jälkeiseen sprintin katselmointiin. Tällä tavoin yrityksen myynti sekä hallinto saavat säännöllisesti katsauksen mitä ohjelmistokehityksessä on tehty ja näkevät ohjelmistotuotteen uusia ominaisuuksia. Vastaavasti päivittäinen ketteristä menetelmistä peräisin oleva päivittäinen ohjelmistokehityspalaveri on laajennettu koko yrityksen laajuiseksi, joten tilannepäivitys mitä yrityksessä tapahtuu ja tiedonvälitys ohjelmistotiimistä muille yrityksen osapuolille on päivittäistä.

V2:n organisaatiossa on yhteensä yli 1000 henkeä ja ohjelmistokehityksen raportointi on perinteisempää raportointia esimiehille ja johtotasolle. Päivittäiset ohjelmistokehityspalaverit, sprintin katselmoinnit ja suunnittelut käydään vain kehitystiimin kesken. Päivittäinen näkyvyys ohjelmistokehitykseen rajautuu tiimin sisälle. Varsinainen tuoteomistaja ei ole osa ohjelmistokehitystiimiä, vaan tuoteomistaja työskentelee asiantuntijana tuoteryhmässä, johon myös haastatellun henkilön tiimi kuuluu. Yhdessä tuoteryhmässä on useita ohjelmistotiimejä monien samanaikaisten ohjelmistokehitysprojektien parissa, yksi tiimi per projekti. Tuoteryhmän tasolla seurataan kaikkien tuoteryhmän ohjelmistokehitystiimien ohjelmistoprojektien etenemistä määräajoin, tiimit raportoivat projektien tilannetta tuoteryhmälle.

Ainoastaan toisessa haastatteluista haastateltava kuvasi metriikoiden käyttöä ennen tuotannon aloitusta. V2:n tiimi on alkuvuodesta 2020 aloittanut uuden ohjelmistokehitysprojektin, jonka määrittelyn ja suunnittelun tukena oli käytetty tiimin kehittäjien aiempien projektien perusteella muodostunutta velositeettia, työmääräarvioita sekä työtehtävien pisteytystä. Käytössä olevien tuotekehitysresurssien määrää oli myös arvioitu osana projektin suunnittelua. Uusien asiakastilausten kautta esiin nousseiden toiminnallisuuksien työmäärää arvioidaan myös säännöllisesti V1:n ohjelmistotiimissä, sillä vaikka ohjelmisto on ylläpitovaiheessa, on jatkokehitys säännöllistä.

Tuotannon aikaista metriikoiden käyttöä kuvattiin molempien haastateltavien toimesta runsaasti. V1 kertoi kehitystiimin seuraavan buildin tilaa. Jatkuvan integroinnin työkalujen avulla heidän versionhallinnassansa on jatkuvasti nähtävillä kehitysversioiden sekä pääversion tilanne. V2 mainitsi puolestaan, että käytössä olevien resurssien määrää

seurataan myös tuotannon aikana. Tuoteryhmässä saatetaan väliaikaisesti siirtää kehittäjiä tiimistä toiseen resurssitarpeen perusteella. Laadunvarmistusta, loppukäyttäjän tyytyväisyyden seurantaa sekä muutosten määrää sprintin aikana seurattiin V1:n tiimissä. V1:n haastattelusta selvisi, että hänen tiimissänsä tapahtuu sprintin aikana muutoksia työlistaan, mikä kertoo, ettei Scrumia noudateta tarkasti kuten haastateltava kuvasi. Tiimin ohjelmistokehityksestä mainittiin, että se on tällä hetkellä asiakasvetoista ja tiimi pyrkii reagoimaan uusiin vaatimuksiin loppukäyttäjän puolelta tehokkaasti. Tiimi seuraa muutoksia sprinttiin kerran viikossa erillisellä viikon suunnittelulla. Sprintin kesto on kolme viikkoa. Tarkemmin Scrumia noudattava V2 ei maininnut, että työlistaan tulisi sprinttien aikana muutoksia.

V1:n yritys osallistuu kokonaisuudessaan sprinttien katselmointiin. Kaikki sprintin tuotokset katselmoidaan yhdessä, jolloin palautetta saadaan muilta kuin kehitystiimin jäseniltä säännöllisesti. Laatua seurataan sprintin aikana luottamalla automaattisin testeihin sekä riittävään tarkkaan kuvaukseen (definition of done) minkä perusteella tehtävä on valmis. V1:n yrityksen oma tuote on SaaS-mallinen tuote, joten asiakasyritysten kanssa ollaan tiiviimmin dialogissa. Tiimi seuraa loppukäyttäjien käyttökokemusta tuotannon virheiden ja tukipyyntöjen kautta mutta myös pyytämällä loppuasiakkaan käyttäjiltä palautetta uusista toiminnallisuuksista sekä päivityksistä. Asiakaskohtaiset muutostyöt myös hyväksytetään asiakkailla osana ohjelmistokehitystä. Loppukäyttäjän tyytyväisyyttä seurataan tyypillisesti ohjelmistokehitystiimin ulkopuolella, mutta palaute on saatavilla myös V1:n tiimille.

Ohjelmistoprojektin keskeneräisten työtehtävien määrää seurattiin V2:n tiimissä. Uutta ohjelmistotuotetta kehittävällä tiimillä on työlista projektille, jota seuraamalla voidaan arvioida, miten jäljellä oleva työmäärä suhteutuu tavoiteltuun aikatauluun. Jäljellä olevaan työmäärää raportoidaan säännöllisesti tuoteomistajalle sekä erinäisille johtoryhmille sekä tietohallintojohtajalle. V2 ei itse tee ohjelmistokehitystä, joten ei osannut kertoa, miten kehittäjät seuraavat ohjelmistokehityksen metriikoita. V1 sen sijaan teknisempänä henkilönä kuvasi monta etenkin testaukseen liittyvää metriikkaa, joita kehitystiimi seuraa osana päivittäistä ohjelmistokehitystä. V1:n tiimissä seurataan jokaisen kehitystehtävän yhteydessä testien rivikattavuutta ja samoin tiedossa olevien virheiden määrää seurataan. Virheeksi tiimi laskee myös erilaisten staattisten koodianalysointityökalujen raportoimat virheet, jotka eivät ohjelmiston käytön yhteydessä ole näkyvillä. Näitä ei kuitenkaan aina korjata. V2 tiesi ainoastaan, että testausta tehdään ja tiimissä on myös erillinen testaaja.

Velositeetti, sprintin jäljellä oleva työmäärä, sprintin tavoite, työmääräarvio sekä arvioidun työmäärän ja toteutuneen työmäärän vertailu nousivat esiin molemmissa haastatteluissa. Työmääräarvion sekä toteutuneen työmäärän suhdetta seurattiin molemmissa yrityksissä myös kehitystiimin ulkopuolella. Molemmissa yrityksissä ulkopuolinen seuranta oli taloudellista. V1:n yrityksessä seurattiin toteutuneita tunteja tarkemmin asiakkaiden tilaaman maksullisen työn osalta. Asiakkaiden suuntaan raportoitiin käytettyjen työtuntien määrää. Erillistä näkymää, josta kertyneitä työtunteja seurataan ei ole vaan tunnit raportoidaan erikseen kuukausittaisen laskutuksen yhteydessä. V2 organisaatio on julkisen hallinnon alainen, joten varojen käyttöä seurataan tarkasti organisaation eri osissa. Ohjelmistotiimin kehittäjät ovat konsultteja, joiden työtuntimäärää seurataan, jotta määrätyissä budjeteissa pysytään. Raportointi tehdään määräajoin yhdessä muun projektin etenemistä kuvaavan raportoinnin yhteydessä.

Sprintin tavoitetta kehitystiimit käyttivät eri tavalla. V2 tiimissä sprintin tavoitteena mainittiin oppiminen sen osalta, kuinka paljon työtä saadaan yhteen sprinttiin mahtumaan. V1 kuvasi, että sprintin tavoite on useimmiten asiakkaalle luvatun työn valmistuminen, muun työn siirtymistä eteenpäin ei pidetty ongelmana. V1 tiimillä on kaksi tuotekehityslistaa, joista sprintteihin poimitaan tehtäviä, asiakastöiden lista sekä ylläpidon ja oman kehityksen lista. Näille kummallekaan ei kuitenkaan kuvattu selkeää tavoitetta.

> *”Me tavallaan eritellään asiakkaiden toiveet kahteen eri kategoriaan, jos asiakas haluaa sen heti ni sen saa käytännössä maksamalla normaalin syklin ulkopuolella. Muussa syklissä ne menee sitten tähän ns. loputtomaan toiveiden tynnyriin josta niitä sitten tota tulee asiakkaille sitä mukaan kun me niinku katsotaan. Tätä tynnyriä ei mitenkään systemaattisesti hallinnoida että sieltä tehtäis nostoja vaan katsotaan sprinttikohtaisesti mikä tähän sprint planin yhteydessä kannattaa poimia.”* (V1)

Myös V2 mainitsi, että kun sopivaa työmäärää opetellaan, oli ensimmäisissä sprinteissä jäänyt tehtäviä tekemättä.

Sprintin jäljellä olevan työmäärän osalta kumpikaan vastaaja ei maininnut, että työmäärän esittämiseen olisi käytetty mitään kaavioita tai kuvaajia.

> *”Joo, mullekin Jira on vieras, oon ennen ollut Trellon käyttäjä ja kyl mä tiedän että Jirasta näitä kaavioita sais. Me ei olla lähetty nyt tekeen niitä.”* (V2)

V1 tiimissä on ennen arvioitu työmäärää tarkemmin muunkin kuin asiakaskohtaisen työn

osalta, mutta tällä hetkellä sitä ei tehdä. V2 seurasi tiimin kesken valmiiden tehtävien määrää. Velositeetin osalta vastaavasti kuin työmääräarvioiden osalta V1 mainitsi, että he voisivat sitä seurata mutta koska tällä hetkellä ohjelmistokehitys on enemmän ylläpitävää sitä ei tehdä. V2 kehitystiimi seuraa velositeettia sprinttitasolla, kuinka monta pistettä saadaan tehtyä. V2 myös mainitsi sprintin suunnittelun yhteydessä käytetyn tehtävien pisteytyksen 2, 4, 8 ja epic, jonka avulla sprintin työmäärää voidaan arvioida. Tiimillä on myös tapana suunnitella sprintit riittävän täyteen, jotta kesken sprintin työ ei pysähdy.

> *”Mutta meil on sellainen tapa et mielummin laitetaan muutama ylimääräinen tota itemi sinne sprinttiin et sit jos hommat lähtee rullaamaan ni on ainakin niin sanotusti tekemistä.”* (V2)

Laadunvarmistus oli molemmille tiimeille yksi ohjelmiston ylläpitovaiheessa seurattava mittari. Asiakkaiden palaute toimii molemmille tiimeille myös laadunvarmistuksessa ohjelmiston käytöstä saatavan palautteen perusteella. V2 mainitsi, että laadunvarmistusta ei varsinaisesti tehdä ohjelmistokehitysprojektin aikana mutta kaikilla organisaation ohjelmistokehitystiimeillä on tarkka prosessi, miten laadunvarmistus tehdään projektin lopussa tuotantoon siirron yhteydessä.

> *”Me kutsutaan sitä korsetiksi, mikä on siis tällainen laadunvarmistustaulukko minkä digiosasto on tehnyt. Siel on kaikkia tietoturvajuttuja elikkä pitää sit käydä läpi siin projekti loppuvaiheessa kohta kohdalta taulukko. Se on aika massiivinen taulukko, siinä on siitä miten käytettävyys on huomioitu ym., varmaan niitä on 50 eri kohtaa hyvin kattavasti ja se sitten katselmoidaan tietyn porukan toimesta ja se saatetaan sitten palauttaa Se on suoraan sanottuna melko inhottava.”* (V2)

Loppukäyttäjän tyytyväisyyttä ei V2:n tiimissä aktiivisesti seurata. Ohjelmistotuotteet päätyvät projekteista tyypillisesti suoraan tuotantokäyttäjille, joilta kyllä saadaan palautetta mutta haastateltava ei kokenut, että tiimillä olisi varsinaisesti käsitystä loppukäyttäjien tyytyväisyydestä. Samoin kuin kehitysvaiheessa, V1:n tiimi seurasi loppukäyttäjän tyytyväisyyttä järjestelmän käyttöön sekä virheilmoitusten avulla. Tuotannon aikaisten virheiden määrää seurataan automaattisen monitoroinnin kautta ja virheiden kautta tunnistettujen vikojen korjaukseen kuluvaa työmäärää arvioidaan usein kesken sprinttien.

Taulukkoon 4 on koottu tunnistettujen metriikoiden yhteenveto perustuen siihen, mitä mittareita eri ohjelmistoprojektin sidosryhmät sekä yksittäiset kehittäjät seuraavat.

Seuranta voi olla joko aktiivista ja omatoimista tai raportointiin perustuvaa. V2 on taulukossa tulkittu tuoteomistajaksi, sillä hänen oman kuvauksensa perusteella hänen roolinsa tiimissä on tuoteomistajamainen. Varsinainen tuoteomistaja, joka on V2:n tiimin ulkopuolinen henkilö on taulukossa asiakkaan, eli työn tilaajan roolissa. V1:n kertoman perusteella, hänen tiimissään ei ole varsinaista tuoteomistajaa, joten sarake on hänen tiiminsä osalta tyhjä.

Taulukko 4: Metriikoiden seuranta haastatelluissa yrityksissä

	SIDOSRYHMÄ				
SEURATTU METRIIKKA	OHJELMISTOKEHITTÄJÄ	OHJELMISTOTIIMI	TUOTEOMISTAJA	MUU ORGANISAATIO	ASIAKAS
BUILDIN TILA	V1	V1			
KÄYTÖSSÄ OLEVAT RESURSSIT			V2	V2	
LAADUNVARMISTUS		V1	V2	V1, V2	V2
LOPPUKÄYTTÄJÄN TYYTYVÄISYYS		V1		V1	
MUUTOSTEN MÄÄRÄ SPRINTIN AIKANA	V1	V1			
PROJEKTIN KESKENERÄISTEN TYÖTEHTÄVIEN MÄÄRÄ		V2	V2	V2	V2
SPRINTIN JÄLJELLÄ OLEVA TYÖMÄÄRÄ	V1, V2	V1, V2	V2	V1	
SPRINTIN TAVOITE		V1, V2	V2		
TESTIEN MÄÄRÄ	V1	V1	V2		
TESTIEN RIVIKATTAVUUS	V1	V1			
TIEDOSSA OLEVIEN VIRHEIDEN MÄÄRÄ		V1			
TYÖMÄÄRÄARVIO		V1, V2	V2	V2	V1, V2
TYÖMÄÄRÄARVIO JA TOTEUTUNEEN VERTAILU		V1	V2	V1, V2	V1
TYÖN PISTEMÄÄRÄ	V2	V2	V2		
VELOSITEETTI		V2	V2		

Ohjelmistokehittäjän ja ohjelmistotiimin välillä metriikan seurannan erot saattavat johtua tarkkuudesta millä haastattelija kuvasi mittaria seurattavan. Esimerkiksi V1 (ohjelmistokehittäjä tiimissään) mainitsi, että tiimissä seurataan loppukäyttäjän tyytyväisyyttä mutta koska hän ei selkeästi kuvannut itse sitä seuraavansa tai yleisesti kehittäjien sitä

seuraavan, ei merkintää taulukkoon laitettu. Taulukosta voidaan nähdä myös, että vaikka V1 mainitsi velositeetin yhtenä metriikoista haastattelussaan, hän myös totesi, että sitä ei tällä hetkellä seurata. Haastateltavien mainitsemien metriikoiden ja seurattiinko metriikoita haastattelua tehtäessä välillä, on eroja. Sidosryhmittäin yhteenlaskettuna (haastateltavan esiintymiskerta taulukossa) V1:n eri metriikoiden seurantaluku eri sidosryhmissä on 22 ja V2:n vastaava luku on 26.

4.3 Metriikoiden vaikutukset

Vastaajien erilainen rooli omissa tiimeissään heijastui suoraan siihen, millä tasolla he pääasiassa ohjelmistokehitystä seuraavat ja minkälaisten metriikoiden vaikutusta he pystyivät arvioimaan. V1 seuraa pienessä organisaatiossa tuotteen elinkaarta niin koodin, ohjelmistokehityksen sekä liiketoiminnan tasolta. Hän on ollut mukana tuotteen kehityksessä alusta asti. Automaattista testausta tehdään paljon ja uusi toiminnallisuus voidaan yhdistää pääversioon, mikäli koodin rivikattavuus jatkuvan integroinnin testipalvelimen raportilla on 100%. Koodikattavuuden osalta on ilmennyt sekä positiivisia että negatiivisia seurauksia.

> *”On todettu hyödylliseksi, meillähän on toi tulkattu kieli käytössä jolloin tota varsinaisesti koodia ei niin hyvin tarkastella vasta kun ajon aikana ja ilman rivikattavuutta tulis helposti bugeja varsinkin niissä yhteyksissä kun päivitetään frameworkeja tai jotain isompaa ja hajoaa helposti jotain. Vuosien varressa on kertynyt ominaisuuksia ja muita tällaisia ja rivimäärä alkaa olla niin iso että on mahdoton testata kaikkea käsin. Huonona puolena tietysti että kehittämistä se hidastaa kun testipatteristo alkaa olla niin iso et se ajaminen kestää”.* (V1)

Myös testien ajossa mainittiin satunnaisesti tapahtuvan häiriöitä, jolloin testit joudutaan ajamaan uusiksi. Näiden satunnaisten tiedossa olevien testien ongelmien ei ole nähty olevan kehitysajan arvoista, eli tiedossa olevat virheet testeissä eivät aiheuta toimenpiteitä. Myös laadunvarmistuksen käytänteet vaikuttavat positiivisilta.

> *”Koska me myös käydään aktiivisesti dialogia meidän asiakkaiden kanssa niin sitten se tulisi jos siellä olis niinkun ongelmia, se tulis heidän kautta esiin ja meidän prosessi on osoittanut että sinne tuotantoon ei ui mitään kriittistä koska ei oo myöskään palaute asiakkailta ollut sellaista että olisi ongelmia*

tämän asian kanssa." (V1)

Muista V1:n kuvaamista metriikoista selkeästi vaikutusta ohjelmistokehitykseen ilmeni asiakkaan suuntaan tehtävien maksullisten tuntitöiden nosto sprinttiin kesken kaiken, joka toimenpiteenä vaikutti sprintin muun työn valmistumiseen sekä sprintin tavoitteeseen. Haastattelusta ei kuitenkaan käynyt ilmi oliko sprintin tavoitteen vaihtumisella vaikutusta sprintin muiden töiden valmistumiseen ja koettiinko muuttuva sprintin työlista ongelmallisena. Asiakaskohtaisen työn osalta tiimi myös seuraa arvioidun sekä toteutuneen työmäärään vastaavuutta. Haastattelusta ei käynyt ilmi minkälainen vaikutus arvioidun työmäärän ylityksellä on työn tilaajan tai tiimin suuntaan.

V2:n haastattelusta esiin nousseet metriikat sekä niiden vaikutukset painottuivat selkeästi ohjelmistokehitykseen tuoteomistajan näkökulmasta. Työmääräarvioiden, tehtävien pisteytyksen sekä velositeetin kautta tiimi pystyy sprinteittäin parantamaan arviointiaan tuleville sprinteille sekä kehitysprojektin aikatauluarviointia koko työlistan tasolla.

> *"Aateltiin et tehdään tälleen oppivasti tätä et katotaan siinä sprintissä et minkä verran ne tota työllistää ja sen sitten näkee sen sprintin aikana ja sitte me taas siitä viisastumme seuraavan sprinttiin. Elikkä tota meillä on siellä estimaatit mikä on iso tehtävä ja mikä on vähän pienempi ja tota sanotaan ett toisen tai kolmannen sprintin aikana meil alkaa olee paljon parempi käsitys siitä mikä on se työmäärä siinä koko projektin aikana."* (V2)

Resurssien käyttöä seurataan kuukausitasolla. Koska V2:n tiimin projektille on asetettu budjetti sekä aikataulutavoite tiimin ulkopuolelta on se, että jäljellä olevaa työmäärää pystytään arvioimaan mahdollisimman tarkasti tärkeää haastateltavalle. V2 mainitsi myös, että tiimiin on mahdollista tuoda hetkellisesti toisesta saman tuoteryhmän tiimistä lisää työvoimaa, mikäli tuotannon vauhti näyttäisi siltä, että aikatauluun ei ehditä. Resurssien tarpeen säännöllinen arviointi auttaa ennakoimaan mahdolliset lisäresurssien tarpeet.

4.4 Muut havainnot tapaustutkimusaineistosta

Haastateltavien kanssa keskusteltaessa ilmeni molempien haastattelujen alussa, ettei heidän ohjelmistotiimeissään tehtäisi mittausta. Kuitenkin keskustelun edetessä myös haastateltavien käsitys siitä, että tehdäänkö mittaamista vai ei muuttui, koska yritykset seurasivat metriikoita passiivisen tiedonkeruun avulla. Molemmilla haastateltaville oli

lähtöoletus, että suorituskyvyn mittaaminen on aktiivista tekemistä.

V1 kertoi, että yrityksen asiakkailla on käytössä eri versioita yrityksen tuotteesta. Tuotantoversioiden hallinnointiin ei ollut koottua näkymää, josta tilannetta voisi seurata mutta tiedon sai tarvittaessa tarkistettua.

V2 nosti esiin, että joskus ohjelmistoprojektikohtaisen tiimin kehittäjät joutuvat palaamaan hetkellisesti edellisten projektiensa pariin, jolloin sprinttiin voi yllättäen tulla resurssivajetta. Yllättävien poissaolojen vaikutus on nopeasti nähtävillä haastavana tekijänä sprintin tavoitteiden saavuttamisessa. Tämän seurauksena myös V2:n ohjelmistokehitystiimi joutuu sopeutumaan ulkopuolisiin muutoksiin siitä huolimatta, että tiimi pyrkii noudattamaan Scrumia tarkasti. V2 kertoi myös, että käynnissä olevalla ohjelmistokehitysprojektilla on tavoiteaikataulu, jolloin uuden ohjelmistotuotteen pitäisi olla valmis. Lopussa V2 täydensi vielä minkä hän itse kokee olevan arvokas tapa seurata ohjelmistoprojektien etenemistä myös hänen aiempien projektien perusteella.

> *”Tossa niinkun melkein tekis mieli sanoa et kokemus on niinku paras, siis kokenut tiimi on paras mittari tai kykenevä arvioimaan sitä että mikä on se heidän kulloinenkin aavistus että ehditäänkö me vai eikö me ehditä.”* (V2)

5 Pohdinta

Tässä luvussa käsitellään tapaustutkimustulosten pohjalta löydettyjä vastauksia tutkimuskysymyksiin sekä tulkitaan tuloksia hyödyntäen myös luvussa 2 esitettyä kirjallisuutta. Käsittely on jaettu alalukuihin tutkimuskysymyksittäin. Alaluvussa 5.1 käsitellään tapaustutkimuksessa tunnistettuja metriikoita suhteessa aiempaan luvussa 2 esitettyihin metriikoihin. Alaluvussa 5.2 käydään läpi, mitä yhtäläisyyksiä ja toisaalta eroavaisuuksia metriikoiden ja mittauksen käytössä tapaustutkimukseen osallistuneissa yrityksissä on aiempaa tutkimustietoon verrattuna. Alaluvussa 5.3 arvioidaan tapaustutkimuksessa ilmenneitä mittauksen hyötyjä ja haittoja. Alaluvussa myös esitellään tämän tutkielman perusteella muodostuneita jatkotutkimusajatuksia. Viimeisessä alaluvussa 5.4 arvioidaan tutkielman sekä tutkimustulosten laatua ja luotettavuutta.

5.1 Tunnistetut metriikat

Tämän tutkielman tutkimuskysymyksen 1 tavoite oli selvittää mitä metriikoita ketterillä ohjelmistotiimeillä on käytössä oman suorituskyvyn seurantaan. Tapaustutkimuksen tuloksissa esiteltiin 15 metriikka, joita tutkielmaan osallistuneissa ohjelmistotiimeissä seurataan osana ohjelmistokehitystä. Tunnistettujen metriikoiden käyttö painottui ohjelmistokehityksen tuotannonaikaisiin vaiheeseen. Luvussa 2.3 todettiin, että ketterässä ohjelmistotuotannossa metriikoiden ja mittauksen on tunnistettu keskittyvän vastaavasti iteraatioiden aikaiseen ohjelmistokehitykseen [KMI15]. Samassa luvussa 2.3 todettiin myös, että vaatimusmäärittelyn ja suunnittelun aikaisia metriikoita käytetään niukasti [KMI15, WnM17]. Tämän tutkielman viidestätoista tunnistetusta metriikasta neljää mainittiin käytettävän ennen ohjelmistotuotannon aloitusta vaatimusmäärittelyjen aikana. Osuus lähentelee yhtä kolmasosaa tunnistetuista metriikoista, joten tämän tapaustutkimusaineiston puitteissa löytyi eroavaisuus aiempaan tutkimustietoon nähden metriikoiden käytöstä suunnitteluvaiheessa.

Tapaustutkimuksen tulosten osalta metriikoiden jakautumista ohjelmistotuotannon eri vaiheisiin selittää osittain se minkälaisessa vaiheessa ohjelmistotiimit olivat omassa ohjelmistokehityksessään. Toisen ohjelmistotiimin tuote oli tuotantokäytössä ja ylläpidossa, minkä seurauksena ylläpidon aikaisia metriikoita ilmeni pääosin vai toisessa haastatteluista. Vastaavasti toinen tiimeistä oli hiljattain aloittanut uuden ohjelmistotuotteen kehityksen minkä seurauksena vaatimusmäärittelyä ja suunnittelua koskevia metriikoita

ilmeni myös vain toisessa haastatteluista. Myös haastateltavan henkilön roolilla omassa ohjelmistotiimissään saattoi olla vaikutusta siihen minkälaisista metriikoista ja mittauksista hän osasi kertoa.

Ketterän ohjelmistotuotannon luonteen takia vaatimuksia tarkennetaan pitkin ohjelmistoprojektia. Tapaustutkimuksen tuloksista nähtiin, että työmääräarviot olivat ainoa mittari, jota seurattiin läpi kaikkien ohjelmistotuotannon vaiheiden. Työmääräarvio metriikkana voi tarkoittaa monia eri asioita kuten luvussa 2.3 todettiin [KMI15]. Tapaustutkimuksen perusteella työmääräarvion tarkoitus oli ensisijaisesti auttaa jäljellä olevan työn arvioinnissa joko tuntimäärällisesti tai pisteyttämällä työtehtäviä. Yksi työmääräarvion käyttötavoista, tarkastella jälkeenpäin, kuinka hyvin työmääräarviot ovat pitäneet paikkansa, tunnistettiin omana metriikkanaan. Luvussa 2.3 mainittiin että yrityksillä on tyypillisesti käytössä myös omia metriikoitaan [FeN00, KMI15]. Tämän tapaustutkimuksen tulokissa ei kuitenkaan tunnistettu yhtään metriikkaa, jota ei olisi esitetty muissa tutkimuksissa. Muiden tutkimusten perusteella [KMI15] yleisesti käytetyt metriikat velositeetti, työmääräarvio ja tiedossa olevien vikojen määrä tunnistettiin myös tämän tapaustutkimuksen tuloksissa.

5.2 Metriikoiden käyttö

Tutkimuskysymys 2 etsi vastausta siihen, ketkä seuraavat ja käyttävät metriikoita ja mittaustuloksia tapaustutkimukseen osallistuneissa ohjelmistotiimeissä ja yrityksissä. Luvussa 2.4 kuvattiin ohjelmistotiimin suorituskyvyn arviointia ohjelmistoprojektin tulosten tai prosessien kautta [LLL07]. Samassa luvussa mainittiin neljä näkökulmaa tulosten arviointiin, laatu, laajuus, aika sekä kustannukset [ChC08]. Tapaustutkimuksessa tunnistettujen metriikoiden perusteella, molemmissa tutkimukseen osallistuneissa yrityksissä seurataan mittareita, joiden avulla ohjelmistoprojektien tuloksia ja tulosten kautta tiimin suorituskykyä voidaan arvioida. Kumpikin tiimeistä tekee laadunvarmistusta sekä seuraa tavoitteiden saavutusta mm. sprintin tavoitetta seuraamalla. Myös kustannuksia seurataan molemmissa tiimeissä mutta seurattavissa kustannuksissa on eroavaisuuksia. Toinen tiimeistä seuraa kustannuksia tarkemmin vain asiakaskohtaisen erikseen laskutettavan työn puitteissa, kun taas toinen yrityksistä seuraa kustannuksia jatkuvasti ohjelmistokehitysresurssien käytön perusteella. Ohjelmistokehityksen aikataulua seurattiin tarkemmin toisessa yrityksistä.

Metriikoita kuten buildin tila sekä testien rivikattavuus raportoitiin seurattavan toisessa

yrityksistä. Näiden pääasiallinen tarkoitus oli tukea ohjelmistotiimin laadunvarmistusta uusien ohjelmiston toiminnallisuuksien kehityksen aikana. Toisaalta esimerkiksi tiimin kyky ylläpitää 100% testien rivikattavuutta voidaan ajatella ohjelmistotiimin prosessien arvioinnin kautta hyvänä suorituskykynä. Näitä teknisempiä metriikoita seurattiin ainoastaan ohjelmistotiimin ja yksittäisten kehittäjien tasolla. Toisen yrityksen osalta vastaavien metriikoiden käyttö jäi epäselväksi sillä haastateltu henkilö ei osannut kertoa ohjelmistokehityksen tarkemmista yksityiskohdista. Hänen yrityksessään laadun varmistusta tehtiin ohjelmistoprojektien lopuksi. Luvussa 2.1 kuvattiin että metriikoita käytetään joskus vain sen takia, että on tarve vastata johonkin ulkoiseen vaatimukseen [FeN00]. Tapaustutkimuksen tuloksista muodostui käsitys, että toisessa tiimeistä laadunvarmistus oli enemmän ulkopuolelta asetettu käytäntö, jota piti tehdä ohjelmistotiimin näkökulmasta epämiellyttävällä tavalla.

Tarkasteltaessa metriikoiden seurantaa yrityksissä sidosryhmittäin, muodostui käsitys, että molemmissa yrityksissä sidosryhmiin suhteutettuna mittareita seurattaisiin lähes saman verran. Vertailuluvut olivat 22 sekä 26 ja yksi selkeä ero näkyi sen osalta, että vain toisessa yrityksistä oli tuoteomistaja. Toisessa yrityksessä ei kuvattu olevan selkeää tuoteomistajaa vaan koko yrityksen osallistuessa sprinttien katselmointiin muodostui käsitys, että ominaisuuksia hyväksytään koko tiimin kesken. Haastattelusta jäi kuitenkin epäselväksi, miten esimerkiksi asiakaskohtaisen töiden hyväksytys käytännössä tapahtuu työn tilaajan kanssa ja kenen toimesta. Toinen havainto mittareiden seurannassa liittyi ohjelmistoprojektin jäljellä olevaan työmäärään, jonka vain toinen haastateltavista mainitsi. Koska kyseinen ohjelmistotiimi kehitti uutta ohjelmistoa, oli projektilla selkeä työlista ja tavoite määriteltynä, jonka jälkeen projekti valmistuu.

Taulukossa, johon sidosryhmittäin vertailulukuja kerättiin (taulukko 4), merkittiin metriikan seuranta vain sille sidosryhmälle, joka haastatteluista ilmeni. Esimerkiksi tiedossa olevien virheiden määrä on tieto, joka toisessa yrityksistä saattaisi olla myös kehittäjien ja mahdollisesti esimerkiksi niiden työntekijöiden tiedossa, jotka ovat asiakkaisiin yhteydessä tuotteen käyttöön liittyen. Haastatteluista kuitenkin pystyttiin muodostamaan kokonaiskäsitys siitä, että molemmissa yrityksissä mittauksia seurataan kaikissa sidosryhmissä metriikasta riippuen. Mitä kauemmas liikutaan varsinaisesta ohjelmistokehityksestä kohti muita sidosryhmiä kuten asiakkaita, sitä enemmän metriikat, joita seurataan ovat kustannuksiin, aikatauluihin ja laatuun liittyviä. Ohjelmistotiimin ja yritysten metriikoiden valinnasta riippumatta, tavoite on pystyä seuraamaan ohjelmisto-

kehitysprojektien etenemistä kaikkien sidosryhmien osalta.

Tapaustutkimuksen tuloksista saatiin osittain käsitystä sille, onko metriikoiden seuranta aktiivista vai passiivista ja raportointiin perustuvaa. Teknisempien metriikoiden seuranta tapahtui toisessa ohjelmistotiimeistä osana päivittäistä työskentelyä aktiivisesti, kun taas molempien ohjelmistotiimien sprinttien lopputuloksia ja työn edistymistä raportoitiin määräajoin muille sidosryhmille. Organisaatioiden ero on yksi selittävä tekijä sille, että pienen yrityksen tiedonjako päivittäin on mahdollista koko henkilöstön kesken. Luvussa 2.1 mainittiin, että metriikoita käytetään joskus väärin [FeN99] mutta tämän tapaustutkimuksen kautta ei saatu selkeää näyttöä väärälle käytölle. Toisaalta yksi henkilö per yritys on hyvin suppea määrää antamaan riittävän kattavaa kokonaiskuvaa, josta voidaan luotettavasti arvioida, onko mittareiden käyttö ja tulosten tulkinta täysin moitteetonta.

Tapaustutkimuksen tuloksissa ilmeni vielä se, että osaa metriikoista, joita yritykset ovat jossain vaiheessa seuranneet ei tällä hetkellä seurattu. Syyksi sille, että mm. velositeettia ei toisessa ohjelmistotiimissä enää seurattu oli se, ettei tietoa tarvittu. Luvussa 2.3 todettiin, että mittaus itsessään vaatii myös iteraatiota ja opettelua [OzK12]. Ohjelmistotiimien tulee tunnistaa mahdollisia puutteita seuraamissaan metriikoissa. Tapaustutkimuksen valossa ainakin toinen yrityksistä on ajan myötä muokannut mittauksiaan.

5.3 Metriikat ohjelmistokehityksen tukena

Tämän tutkielman tutkimuskysymys 3 etsi vastausta siihen tukevatko ohjelmistotiimien käyttämät metriikat ohjelmistokehitystä ja voidaanko mittauksen avulla havaita ongelmia tai kehittää tiimin suorituskykyä. Tapaustutkimuksen tulosten valossa molemmilla tutkimukseen osallistuneilla ohjelmistokehitystiimeillä oli käytössä metriikoita, joiden avulla ohjelmistokehityksen eri vaiheisiin saatiin tukea. Toisessa yrityksistä haastateltavan henkilön ollessa ohjelmistokehittäjä, saatiin tietoa esimerkiksi testauksen metriikoista, joilla oli positiivinen vaikutus ohjelmistotuotteeseen laadun näkökulmasta. Tiimi toisaalta katselmoi sprintin tuloksia yhdessä koko yrityksen kanssa sekä käy aktiivisesti keskustelua asiakkaiden kanssa kehitystyön ohessa ja haastattelusta ei selvinnyt kuinka suuri vaikutus sillä, että asiakas voi kesken kehitystyön kommentoi kehitysversiota, on lopulliseen asiakkaan tyytyväisyyteen. Vastaavasti koko yrityksen kanssa tehdyllä katselmoinnilla saatetaan huomata vikoja, joita automaattinen testaus ei huomaa. Tiimissä myös seurattiin buildin tilaa, tehtiin työmääräarvioita ja aktiivisesti selvitettiin asiakkaan tyytyväisyyttä. Nämä metriikat on todettu tärkeiksi aiemmissa tutkimuksissa [KMI15] myös muissa

ohjelmistotiimeissä.

Haastateltu henkilö, joka toimi tuoteomistajan roolissa ohjelmistotiimin päin, nosti puolestaan esille ohjelmistoprojektin seurantaa tukevia metriikoita kuten velositeetin, sekä sprinttien ja iteratiivisen ohjelmistokehityksen tarjoaman mahdollisuuden oppia aiemmasta. Näiden avulla tiimi pystyy tarvittaessa etukäteen raportoimaan yrityksessään, mikäli tavoiteaikataulusta ollaan jäämässä jälkeen ja hankkimaan lisää resursseja ohjelmistokehityksen tueksi. Luvussa 2.3 mainittiin, että yksi tavoite sprintin aikaisille metriikoille on lisätä läpinäkyvyyttä ohjelmistotiimin sekä muiden sidosryhmien osalta [KMI15]. Tässä tavoitteessa tapaustutkimuksen valossa paremmin vaikuttaa onnistuvan pieni kuuden hengen yritys, sillä monia metriikoita seurataan myös kehitystiimin ulkopuolella päivittäin. Suuren, monen ohjelmistotiimin yrityksen haaste oli se, että ohjelmistotiimistä ulospäin raportointia tehdään määräajoin. Tutkimustuloksissa ei kuitenkaan käy ilmi, tapahtuuko jossain muussa suuremman yrityksen tiimeistä raportointi jotenkin muuten.

Ohjelmistotiimien suorituskyvyn mittaukseen pehmeiden metriikoiden avulla kaivataan edelleen käytäntöön soveltuvia keinoja kuten myös luvussa 2.4 mainittiin [DiD12]. Tähän tutkielmaan haastatelluista ohjelmistotiimeistä ei tunnistettu, että niissä olisi seurattu pehmeitä metriikoita, kuten tiimityön laatua tai tiimin ryhmähenkeä. Haastateltavat kuvasivat määrällisiä arvoja tuottavia kovia metriikoita. Asiakkaan tyytyväisyys voisi olla myös pehmeä metriikka mutta haastattelusta ei selvinnyt, että sitä olisi mitattu muuten kuin vaatimusten täyttymisen tai vikojen kautta.

Tapaustutkimuksessa ei selvitetty miten ohjelmistotiimeissä on alun perin valittu metriikat ja mittarit, joita heillä oli käytössä. Näin ollen arviota sille, onko mittareiden valinta ollut onnistunut ei voida arvioida. Toisessa ohjelmistotiimistä toki kerrottiin laadunvarmistuksen olevan ulkopuolelta määritelty käytäntö. Tapaustutkimuksen tutkimusaineisto ja sitä kautta tutkimustulokset kuvaavat melko vähän myös sitä, miten mittaus vaikutti ohjelmistokehitystiimien suorituskykyyn. Toinen vastaajista mainitsi, että kehitystiimin jäseniä joudutaan satunnaisesti antamaan muiden tiimien käyttöön, jolloin tiimin suorituskyky laskee. Tämä ei kuitenkaan ole mittauksen seuraus, mutta vaikutus näkyy mittareissa kuten velositeetti.

Tapaustutkimuksessa tunnistetut mittarit kerryttävät mittausdataa ohjelmistokehityksen myötä, joten mittaus ei välttämättä kuormita ohjelmistotiimiä ja tätä kautta vaikuta

negatiivisesti suorituskykyyn. Molemmilla haastateltavilla oli myös aluksi ajatus, ette he eivät varsinaisesti mittaa mitään juuri sen takia, että käytetyt metriikat kerryttävät dataa muun päivittäisen työskentelyn ohessa. Toinen yrityksistä kuvasi asiakastyön vaikuttavan sprinttien työlistaan ja sprintin tavoitteeksi kuvattiin tällöin asiakaskohtaisen työn saaminen valmiiksi. Arvioitaessa suorituskykyä tulosten kautta, tiimin suorituskyky voitaisiin tulkita hyväksi tilanteissa, jossa sprintin tavoitteet asiakaskohtaiseen työhön saavutetaan. Toisessa ohjelmistotiimeistä sprintit suunniteltiin tarkoituksella hieman liian täyteen, jolloin tehtäviä yleensä siirtyi eteenpäin. Tämän perusteella on vaikea arvioida suorituskykyä sprintin tavoitteen kautta. Haastatteluista ei ilmennyt, miten tiimit kokevat onnistuneensa, mikäli sprintin tavoite saavuteaan.

Ohjelmistotiimin suorituskyvyn mittaus erityisesti pehmeiden metriikoiden avulla on mielenkiintoinen aihe jatkotutkimukselle. Tässä tapaustutkimuksessa ohjelmistotiimeistä tunnistettiin tiimityöhän positiivisesti vaikuttavia, luvussa 2.4 [SSB05] esitettyjä piirteitä kuten sopeutumiskykyä muutoksin sprintin tavoitteen muuttuessa, katkeamaton kommunikointi tiivisti koko yrityksen tasolla pienessä yrityksessä sekä yhteinen suorituskyvyn seuranta esimerkiksi velositeetin avulla. Aiemmissa tutkimuksissa [LLL07, SSB05] on esitetty tiimityön positiivisia vaikutuksia tiimin suorituskykyyn ja tämän mittaus ja havainnointi ohjelmistoalalla kaipaa lisää tutkimusta. Myös se, miten ohjelmistoyritykset käytännössä valitsevat seuraamiaan mittareita ja metriikoita olisi mielenkiintoinen aihe jatkotutkimukselle.

5.4 Tutkielman laadun ja luotettavuuden arviointi

Tapaustutkimukselle on tyypillistä, että sen toistaminen ja samoihin tuloksiin päätyminen on vaikeaa. Tähän vaikuttavat useat tekijät kuten tutkimusajankohta sekä tutkijan oma ennakkokäsitys tutkimuksen kohteesta. Tämä tutkielman laatija työskentelee itse toisessa tutkimukseen osallistuneista yrityksistä. Tutkijan tulee pysyä neutraalina ja tulkita vain käytössä olevaa tutkimusaineistoa ja pyrkiä sulkemaan muu olemassa oleva tieto pois, minkä seurauksena toisen yrityksen tutkimusaineistoa on tulkittu hieman varovaisemmin hiljaisen tiedon välttämiseksi.

Tapaustutkimukselle on tärkeää, että aineiston analysointia tehdään heti kun sitä on saatavilla ja samalla uuden aineiston kerääminen on yhä käynnissä [RuH09]. Tällöin tutkija pystyy muokkaamaan toimintaansa kentällä ja päivittämään esimerkiksi haastattelurunkoaan, mikäli tunnistetaan jotain, mikä vaatii lisää huomiota tutkimusaineistoa kootessa.

Yksi tämän tutkielman tutkimustuloksissa näkyvä tekijä on se, että aineisto kerättiin lyhyellä aikataululla ja analyysi tutkimusaineistolle tehtiin myöhäisessä vaiheessa. Tämän seurauksena huomattiin, että haastattelujen runkoa olisi ollut hieman tarvetta muokata mahdollisille seuraaville haastatteluille, jotta tiimin oma käsitys suorituskyvystä sekä miten metriikat on valittu, olisi paremmin saatu kartoitettua. Tutkimusaineiston analysoinnin yhteydessä tunnistettiin myös muutamia rooleja yrityksissä, joilta olisi ollut mielenkiintoista saada haastattelu osaksi tutkimusta. Tutkimushaastatteluista kertyi niukasti muita havaintoja kuin mitä haastattelussa lähdettiin selvittämään. Toisaalta haastattelut onnistuivat selvittämään sen mitä oli tarkoituskin avoimen haastattelun rungon avulla mutta avoimet haastattelut myös mahdollistaisivat asioiden, joita ei ole etukäteen osattu ennakoida ilmenemisen. Haastattelun runko on saattanut olla liian ohjattu.

Tutkimusaineistoa kerättiin henkilöiltä, joilla on omassa ohjelmistokehitystiimissään erilainen rooli, minkä ansiosta tutkimustuloksissa on eri näkökulmia edustettu. Silti aineisto on melko suppea, joten yleistyksiä sen pohjalta on vaikea tehdä. Pieni kuuden hengen yritys saattaa myös seurata ja kiinnittää huomiota erilaisiin mittareihin kuin esimerkiksi pieni aloitteleva ohjelmistoalan startup-yritys, jonka täytyy nopealla aikataululla kehittää ohjelmistotuotettaan. Suuremman organisaation ohjelmistotiimi koostui konsulteista mutta tuoteomistajan näkökulmasta saatu haastattelu antaa kohtalaisen hyvin yleistettävän kuvan siitä, mitä tuoteomistajat tyypillisesti seuraavat. Tuloksista löytyi yhtäläisyyksiä ja eroja aiempien tutkimusten tuloksiin verrattuna. Tämän tapaustukimuksen tuloksien pohjalta löytyi muutama suunta, mihin tutkimusta voisi jatkaa.

Tutkielman luku 2 on koostettu ilman etukäteen laadittua suunnitelmaa kirjallisuuskatsauksen totutuksesta. Kirjallisuuskatsaukseen käytetyt ensimmäiset hakutermit muodostettiin tutkielman otsikon ja avainsanojen kautta ja tämän jälkeen haku laajentui pääosin seuraamalla löydettyä aineistoa eteenpäin (engl. backward snowballing). On mahdollista, että jotain tärkeää aiempaa tutkimustietoa ei ole huomioitu tapaustutkimuksen tuloksia analysoitaessa. Lähdeaineistoa lukuun 2 ei ole arvioitu yhtenäisten kriteerien avulla.

6 Yhteenveto

Tässä tutkielmassa tutkittiin tapaustutkimuksen avulla mitä metriikoita ketterät ohjelmistotiimit käyttävät ohjelmistokehityksessään suorituskykynsä tukena. Tunnistetut mittarit, joita tutkimukseen osallistuneissa yrityksissä seurattiin, antoivat ohjelmistotiimille tietoa kaikissa ohjelmiston elinkaaren vaiheissa. Metriikoiden käyttö painottui ohjelmistojen kehityksessä iteraatioiden yhteyteen. Tämä vahvistaa aiempien tutkimuksien tuloksia siitä, että mittaaminen ei ole yhtä aktiivista ennen ohjelmiston kehityksen aloitusta tai ylläpitovaiheessa kuin mitä se on ohjelmistotuotteen tuotanto- sekä testausvaiheessa.

Tutkielmassa selvittiin myös ketkä ohjelmistoyrityksissä seuraavat metriikoita. Tapaustutkimustulosten pohjalta metriikoita seurataan yksittäisestä ohjelmistokehittäjästä aina ohjelmistoprojektien asiakkaisiin sekä muihin sidosryhmiin asti. Sidosryhmittäin löytyi eroja seurattavien mittareiden välillä. Lähellä ohjelmiston kehitystä ja kooditasoa metriikat ovat teknisempiä ja niiden tulkinta ja käyttö tapahtuu pääasiassa ohjelmistotiimien ja ohjelmistokehittäjien toimesta. Ohjelmistoprojektien muille sidosryhmille mittaukset antoivat tämän tutkielman perusteella näkemystä korkeammalla tasolla ohjelmistokehityksen etenemisestä suhteessa aikatauluun, kustannuksiin sekä vaatimuksiin liittyen. Tapaustutkimuksessa ilmeni, että osa seuratuista mittauksista tuki tavoitetta antaa ohjelmistoprojektien kaikille sidosryhmille tietoa projektin etenemisestä ja mahdollisista haasteista.

Tunnistettujen metriikoiden ja mittauksen seurannan selvityksen lisäksi tämä tutkielma selvitti vielä, miten mittaus vaikuttaa ohjelmistotiimien suorituskykyyn ja voidaanko metriikoiden avulla kehittyä ohjelmistotiimin tasolla tai havaita ongelmia. Tapaustutkimuksen tuloksissa ei noussut esiin, että mittaaminen olisi koettu suorituskykyä haittaavaksi tekijäksi iteraatioiden aikana. Molemmat tutkimukseen osallistuneet ohjelmistotiimit olivat löytäneet metriikoita, joita seuraamalla ne saivat tukea esimerkiksi laadun varmistukseen, virheiden ehkäisemiseen sekä ohjelmistotuotannon aikataulujen tarkempaan arviointiin. Ohjelmistoprojektien tulosta pidetään yhtenä keinona kuvata ohjelmistotiimin suorituskykyä ja tämän pohjalta tiimien seuraamat mittarit tukivat paremman suorituskyvyn saavuttamista tukemalla tiimejä aikataulujen ja ohjelmistotuotteen laadun osalta.

Ohjelmistotiimien suorituskyvyn tutkimuksessa jatkossa tulisi tämän tutkielman havaintojen perusteella keskittyä selvittämään, miten seurattavat metriikat valitaan käytännössä ja tunnistamaan tapoja mitata tiimien toimintaympäristön, tiimityöskentelyn ja muiden pehmeiden tekijöiden vaikutusta ohjelmistotiimin suorituskykyyn.

Lähteet

Bas92 Basili, V. R. (1992). *Software modeling and measurement: the Goal/Question/Metric paradigm*.

Bec99 Beck, K. (1999). Embracing change with extreme programming. *Computer*, *32*(10), 70-77.

Bec00 Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Bec01 Beck, K. et al. (2001). Manifesto for Agile Software Development. http://agilemanifesto.org/. [17.3.2020].

Boe96 Boehm, B. (1996). Anchoring the software process. *IEEE software*, *13*(4), 73-82.

ChC08 Chow, T., & Cao, D. B. (2008). A survey study of critical success factors in agile software projects. *Journal of systems and software*, *81*(6), 961-971.

Cla95 Clancy, T. (1995). The CHAOS Report. *The Standish Group*.

DDA08 Dingsøyr, T., Dybå, T., & Abrahamsson, P. (2008, August). A preliminary roadmap for empirical research on agile software development. In *Agile 2008 Conference* (pp. 83-94). IEEE.

DiD12 Dingsøyr, T., & Dybå, T. (2012, June). Team effectiveness in software development: Human and cooperative aspects in team effectiveness models and priorities for future studies. In *2012 5th international workshop on co-operative and human aspects of software engineering (chase)* (pp. 27-29). IEEE.

DFD16 Dingsøyr, T., Fægri, T. E., Dybå, T., Haugset, B., & Lindsjørn, Y. (2016). Team performance in software development: research results versus agile principles. *IEEE software*, *33*(4), 106-110.

DPF15 Dutra, A. C., Prikladnicki, R., & França, C. (2015, August). What do we know about high performance teams in software engineering? Results from a systematic literature review. In *2015 41st Euromicro Conference on Software Engineering and Advanced Applications* (pp. 183-190). IEEE.

ElK08 El Emam, K., & Koru, A. G. (2008). A replicated survey of IT software project failures. *IEEE software*, *25*(5), 84-90.

FIK15 Fagerholm, F., Ikonen, M., Kettunen, P., Münch, J., Roto, V., & Abrahamsson, P. (2015). Performance Alignment Work: How software developers experience the continuous adaptation of team performance in Lean and Agile environments. *Information and Software Technology*, *64*, 132-147.

FeN99 Fenton, N. E., & Neil, M. (1999). Software metrics: successes, failures and new directions. *Journal of Systems and Software*, *47*(2-3), 149-157.

FeN00 Fenton, N. E., & Neil, M. (2000, May). Software metrics: roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 357-370).

FNM08 Fenton, N., Neil, M., & Marquez, D. (2008). Using Bayesian networks to predict software defects and reliability. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, *222*(4), 701-712.

GMP14 Gray, D., Micheli, P., & Pavlov, A. (2014). *Measurement madness*. Wiley.

GCF98 Guinan, P. J., Cooprider, J. G., & Faraj, S. (1998). Enabling software development team performance during requirements definition: A behavioral versus technical approach. *Information systems research*, *9*(2), 101-125.

Hec08 Heckerman, D. (2008). A tutorial on learning with Bayesian networks. In *Innovations in Bayesian networks* (pp. 33-82). Springer, Berlin, Heidelberg.

HiC01 Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer*, *34*(9), 120-127.

Him09 Himbert, M. E. (2009). A brief history of measurement. *The European Physical Journal Special Topics*, *172*(1), 25-35.

HuF10 Humble, J., & Farley, D. (2010). Continuous Delivery: Reliable Software Releases through Build. *Test, and Deployment Automation. Addison-Wesley*, 2013-01.

JeZ03 Jensen, B., & Zilmer, A. (2003, May). Cross-continent development using Scrum and XP. In *International Conference on Extreme Programming and Agile Processes in Software Engineering* (pp. 146-153). Springer, Berlin, Heidelberg.

KPF95 Kitchenham, B., Pfleeger, S. L., & Fenton, N. (1995). Towards a framework for software measurement validation. *IEEE Transactions on software Engineering*, *21*(12), 929-944.

KnS10 Kniberg, H., & Skarin, M. (2010). *Kanban and Scrum-making the most of both*. Lulu. com.

Kni15 Kniberg, H. (2015). Scrum and XP from the Trenches. http://tscherning.mono.net/upl/10004/110224ScrumAndXp-FromTheTrenches.pdf. [5.4.2020]

KMI15 Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development–A systematic literature review of industrial studies. *Information and Software Technology*, *62*, 143-163.

LMP15 Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V. P., Itkonen, J., Mäntylä, M. V., & Männistö, T. (2015). The highways and country roads to continuous deployment. *Ieee software*, *32*(2), 64-72.

LLL07 Liang, T. P., Liu, C. C., Lin, T. M., & Lin, B. (2007). Effect of team diversity on software project performance. *Industrial Management & Data Systems*.

MaS02 Mar, K., & Schwaber, K. (2002). Scrum with XP. *Informit. com*.

MMR08 Mathieu, J., Maynard, M. T., Rapp, T., & Gilson, L. (2008). Team effectiveness 1997-2007: A review of recent advancements and a glimpse into the future. *Journal of management*, *34*(3), 410-476.

MuQ12 Mushtaq, Z., & Qureshi, M. R. J. (2012). Novel hybrid model: Integrating scrum and XP. *International Journal of Information Technology and Computer Science (IJITCS)*, *4*(6), 39.

NaR69 Naur, P., & Randell, B. (1969). Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th-11th October 1968.

NHW15 Newcomer, K. E., Hatry, H. P., & Wholey, J. S. (2015). Conducting semi-structured interviews. *Handbook of practical program evaluation*, *492*.

OzK12 Oza, N., & Korkala, M. (2012, March). Lessons Learned In Implementing Agile Software Development Metrics. In *UKAIS* (p. 38).

PoP03 Poppendieck, M., & Poppendieck, T. (2003). Lean software development: an agile toolkit.

RHT13 Radjenović, D., Heričko, M., Torkar, R., & Živkovič, A. (2013). Software fault prediction metrics: A systematic literature review. *Information and software technology*, *55*(8), 1397-1418.

RMO19 Rodríguez, P., Mäntylä, M., Oivo, M., Lwakatare, L. E., Seppänen, P., & Kuvaja, P. (2019). Advances in using agile and lean processes for software development. In *Advances in Computers* (Vol. 113, pp. 135-224). Elsevier.

Roy87 Royce, W. W. (1987, March). Managing the development of large software systems: concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering* (pp. 328-338).

RuH09 Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131.

SSB05 Salas, E., Sims, D. E., & Burke, C. S. (2005). Is there a “big five” in teamwork?. *Small group research*, *36*(5), 555-599.

SDG16 Savor, T., Douglas, M., Gentili, M., Williams, L., Beck, K., & Stumm, M. (2016, May). Continuous deployment at Facebook and OANDA. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*(pp. 21-30). IEEE.

Sch04 Schwaber, K. (2004). *Agile project management with Scrum*. Microsoft press.

ScS17 Schwaber, K., & Sutherland, J. (2017). The scrum guide-the definitive guide to scrum: The rules of the game. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf. [5.4.2020]

SMD11 Stray, V. G., Moe, N. B., & Dingsøyr, T. (2011, May). Challenges to teamwork: a multiple case study of two agile teams. In *International conference on agile software development* (pp. 146-161). Springer, Berlin, Heidelberg.

Str16 Strode, D. (2016). Applying Adapted Big Five Teamwork Theory to Agile Software Development. *arXiv preprint arXiv:1606.03549*.

SFP11 Sudhakar, G. P., Farooq, A., & Patnaik, S. (2011). Soft factors affecting the performance of software development teams. *Team Performance Management: An International Journal*.

WCC12 Wang, X., Conboy, K., & Cawley, O. (2012). “Leagile” software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, *85*(6), 1287-1299.

Wes05 Westfall, L. (2005). 12 Steps to useful software metrics. *The Westfall Team*.

Wir08 Wirth, N. (2008). A brief history of software engineering. *IEEE Annals of the History of Computing*, *30*(3), 32-39.

WnM17 Wnuk, K., & Maddila, K. C. (2017, October). Agile and lean metrics associated with requirements engineering. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement* (pp. 33-40).

YuP14 Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. *Information and software technology*, *56*(8), 911-921.

Liite 1. Tutkimushaastattelun runko

Alustus

Alussa lyhyt alustus ja kertaus vielä aihepiiristä haastattelijan toimesta. Tämän jälkeen haastattelija pyytää haastateltavaa kertomaan omasta roolistaan organisaatiossa / tiimissä. Selvitetään myös, kuinka kauan henkilö on ollut tiimissä ja onko tiimi uusi tai mahdollisesti projektikohteinen vai pysyvä ohjelmistokehitystiimi. Kysytään haastateltavalta, minkälainen rakenne organisaatiossa on ja onko tiimejä useita.

Ohjelmistokehitysmenetelmät ja metriikat

Tämän jälkeen pyydetään henkilöä kuvailemaan vapaasti, miten ohjelmistokehitystä tehdään. Haastattelija avaa keskustelua mittauksesta ja pyritään antamaan vastaajan kertoa mitä tiimi seuraa osana ohjelmistokehitystään. Mikäli metriikoita ei nouse paljoa esille, haastattelija voi kysyä esimerkin omaisesti seurataanko teillä metriikkaa x.

Metriikoiden seuranta

Kun haastattelijalla on käsitys metriikoista, joita tiimissä seurataan, pyydetään haastateltavaa kuvailemaan ketkä seuraavat metriikoita. Tarkennetaan myös organisaation tasolla, seurataanko jotain mittareita tiimin ulkopuolelta. Annetaan haastateltavan kuvailla miten tulokset ovat nähtävillä ja onko näkyvyys rajattua?

Metriikoiden vaikutukset

Selvitetään ovatko mittarit / mittaaminen vaikuttaneet ohjelmistokehitykseen. Kysytään, tehdäänkö mittareiden valossa muutoksia ohjelmistokehitykseen tai kiinnitetäänkö joihinkin mittareihin aktiivisesti huomiota ja pyritään parantamaan tuloksia. Kuka kiinnittää huomiota, miten päätetään tehdä muutoksia mittareiden perusteella? Voidaanko mittareita seuraamalla ennakoida jotain tiimin suorituskyvystä / suoriutumisesta?

Viimeiseksi kysytään, onko mittareista haastateltavan mielestä hyötyä vai onko enemmän mittausta mittauksen takia tai ovatko mittarit ovat saatavilla mutta niitä ei oikein käytetä tai osata hyödyntää? Jos on hyötyä tai haittaa pyydetään haastateltavaa tarkentamaan.

Yhteenveto

Loppuun annetaan vielä vastaajalle mahdollisuus vapaasti lisätä tai tarkentaa mikäli hänellä on tullut haastattelun aikana vielä jotain mieleen käsiteltyihin teemoihin liittyen.

Liite 2. Tutkimusaineiston koodaus

Tässä liitteessä on esitetty tutkimusaineiston koodauksen lopputulos ensimmäisen ja toisen tutkimusaineiston koodauskierroken jälkeen. Ensimmäisen kierroksen jälkeinen tilanne on esitetty kuvassa 1 ja vastaavasti toisen kierroksen jälkeinen tila kuvassa 2.

Kuva 1: Ensimmäinen koodauskierros

Aika	1
Aikataulu	6
Asiakas	10
Backlog	4
Bugi	1
Burndown	2
Daily standup	2
Definition of done	3
Jatkuva integrointi	1
Kanban	1
Käytettävyys	1
Ketterät menetelmät	8
Kilpailu	1
Kokemus	1
Kokenut tiimi	1
Kommunikointi	2
Laadunvarmistus	9
Läpinäkyvyys	3
Metriikan vaikutus	6
Metriikoita	10
Monta Backlogia	2
Organisaatio	6
Palaute	1
Priorisointi	7
Rahoitus	6
Resurssit	14
Retrospektiivi	6
Rooli	7
Scrum	2
Scrum master	1
Sidosryhmä	7
Sprintin suunnittelu	5
Sprintin tavoite	6
Sprintti	14
Testaus	5
Tletoturva	2
Tiimi	13
Toimialaosaaminen	1
Tuotantopäivitys	9
Tuotekehitys	11
Tuotekehitysprojekti	5
Tuoteomistaja	8
Tuotteen elinkaari	7
Työmääräarvio	11
Vaatimusmäärittely	3
Velositeetti	3
Versiohallinta	5
Ylläpito	1
48 Code(s)	

Kuva 2: Toinen koodauskierros