Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

SAMULI KANKAANPÄÄ

Ketterän ohjelmistokehitysprojektin sisäänrakennetun laadun kehittäminen

JOHTAMISEN JA PALVELULIIKETOIMINNNAN YAMK-TUTKINTO-OHJELMA
2023

TIIVISTELMÄ

Kankaanpää, Samuli: Ketterän ohjelmistokehitysprojektin sisäänrakennetun laadun kehittäminen
Opinnäytetyö, ylempi AMK
Johtamisen ja palveluliiketoiminnan YAMK-tutkinto-ohjelma
Toukokuu 2023
Sivumäärä: 94 + liitteet

Tässä tutkimuksellisessa kehittämistyössä tarkasteltiin ketterän ohjelmistokehitysprojektin sisäänrakennetun laadun kehittämistä. Työn tavoitteena oli esittää konkreettisia kehittämisehdotuksia, joiden avulla tutkimuksen kohteena olevan ketterän ohjelmistokehitys projektin sisäänrakennettua laatua voidaan parantaa. Tietoperustassa käydään läpi ohjelmistokehityksen ketteriä periaatteita, esitetään merkittävimmät erot ketterien ja perinteisen ohjelmistokehitys menetelmien välillä, käydään läpi ketterän ohjelmistokehityksen menetelmiä ja syvennytään tutkimuksen kohteena olevan projektin käytössä oleviin SAFe- ja Scrum-viitekehyksiin. Lopuksi käydään läpi SAFe-viitekehyksen ydinarvot ja sisäänrakennetun laadun elementit.

Tutkimuksellisen kehittämistyön tutkimusmenetelmäksi valittiin kvalitatiivinen eli laadullinen tutkimus. Empiirisen tutkimuksen aineisto kerättiin teemahaastattelujen avulla. Haastattelujen teemat perustuivat ilmiöön liittyvään kirjallisuuteen ja aiempiin tutkimuksiin. Tutkimuksen kohteena oli kahdeksan tutkimuksen kohteena olevassa projektissa työskentelevää henkilöä. Tämän jälkeen tutkimusaineisto analysoitiin teemoittelun avulla peilaten tutkimustuloksia teorialähtöisesti aiempiin tutkimuksiin.

Tutkimustulosten perusteella syntyi monipuolinen läpileikkaus teemoista, joiden nähdään vaikuttavan tutkimuksen kohteena olevan ketterän ohjelmistokehitysprojektin sisäänrakennettuun laatuun ja sen parantamiseen. Merkittävimmiksi teemoiksi koettiin: SAFe-viitekehyksen roolien ja tapahtumien oikeanlainen hyödyntäminen, sujuva arvonvirtaus, laadunvarmistus ja testaus, jatkuva parantaminen sekä välitön ja onnistunut kommunikaatio. Näiden tutkimustulosten sekä aiempien tutkimusten tulosten perusteella tehtiin johtopäätökset tuloksista. Lisäksi esitettiin tutkimuksen tavoitteena olleita konkreettisia kehitysehdotuksia sisäänrakennetun laadun kehittämiseksi tutkimuksen kohteena olevassa ketterässä ohjelmistokehitysprojektissa.

Laadulliselle tutkimukselle ominaisesti tutkimustulokset eivät ole yleistettävissä koskemaan suoraan muita ohjelmistokehitysprojekteja. Tutkimuksen perusteella saatiin kuitenkin tietoa tutkimuksen kohteena olevan projektin sisäänrakennettua laatua koskevien kehittämistoimenpiteiden osalta sekä esitettiin niiden pohjalta konkreettisia kehittämisehdotuksia.

Avainsanat: ketterä ohjelmistokehitys, ketterät periaatteet, ketterät menetelmät, sisäänrakennettu laatu, SAFe, Scrum

Abstract

Kankaanpää, Samuli: Developing the built-in quality of an agile software development project
Master's thesis
Master's degree program in management and service business
May 2023
Number of pages: 94 + attachments

In this research, the development of the built-in quality of an agile software development project was examined. The goal of the work was to present concrete development proposals that can be used to improve the built-in quality of the agile software development project that is the subject of the research. In the database, the agile principles of software development are reviewed, the most significant differences between agile and traditional software development methods are presented, the methods of agile software development are reviewed, and the SAFe and Scrum frameworks used by the project under study are explored. Finally, the core values of the SAFe framework and the elements of built-in quality are reviewed.

Qualitative research was chosen as the research method for the research development work. The material of the empirical research was collected by means of theme interview. The themes of the interviews were based on the literature related to the phenomenon and previous studies. The subjects of the study were eight people working in the project under research. After this, the research material was analyzed with the help of thematization, mirroring the research results in theory-oriented previous research.

Based on the research results, a diverse cross-section of themes emerged, which are seen to affect the built-in quality of the agile software development project under research and its improvement. The most important identified themes were: proper utilization of SAFe framework roles and events, smooth value flow, quality assurance and testing, continuous improvement and immediate and successful communication. Based on the results of these researches and the results of previous researches, conclusions were made about the results. In addition, concrete development proposals that were the goal of the research were presented for the development of built-in quality in the agile software development project that is the subject of the study.

Characteristic of qualitative research, the research results cannot be generalized to directly apply to other software development projects. However, on the basis of the research, information was obtained regarding the development measures regarding the built-in quality of the project under study, and concrete development proposals were presented based on them.

Keywords: agile software development, agile principle, agile methodologies, built-in quality, SAFe, Scrum

SISÄLLYS

TERMI- JA LYHENNELUETTELO

Agile	Ketterä
Agile Release Train (ART)	Julkaisujuna
Backlog	Kehitysjono
Development team	Kehitystiimi
Definition of Done (DoD)	Valmiin määritelmä
Definition of Ready (DoR)	Työ, joka on valmis aloitettavaksi
Feature	Ominaisuus
Inkrementti	Tuotteen julkaisukelpoinen versio
Iteraatio	Lyhyt työjakso (esim. sprintti)
Iteratiivisesti	Asteittain
Inspect&Adapt	Tutki & sopeuta
Kanban	Näkyvä taulu työnkulun seuraamiseen
Minimum Viable Product (MVP)	Pienin julkaisukelpoinen tuote
Product Backlog	Tuotteen kehitysjono
Product Manager (PM)	Tuotepäällikkö
Product Owner (PO)	Tuoteomistaja
Program increment	Julkaisujunan inkrementti
Program Increment Planning	Julkaisujunan suunnittelupalaveri
Retrospektiivi	Katsaus jo tapahtuneisiin asioihin
Scaled Agile Framework (SAFe)	Skaalautuva ketterä viitekehys
Scrum	Projektinhallinnan viitekehys
Scrum Master	Scrum tiimin palveleva johtaja
Sprint	Sprintti, 1-4 vko työskentelyjakso
Sprint Planning	Sprintin suunnittelukokous
Sprint Review	Sprintin katselmus
Story / User story	Käyttäjätarina
Velositeetti	Tiimin nopeuden mittari

1 JOHDANTO

1.1 Opinnäytetyön tausta ja aiheen esittely

Tässä tutkimuksellisessa kehittämistyössä tarkastellaan ketterää ohjelmistokehitystä sisäänrakennetun laadun kehittämisen näkökulmasta. Ohjelmistokehitys tarkoittaa ohjelmiston suunnittelua, kehittämistä ja ylläpitoa. Se sisältää useita eri vaiheita, kuten vaatimusmäärittelyä, suunnittelua, toteutusta, testausta ja julkaisua. Ohjelmistokehitys voi kattaa erilaisia ohjelmistoja, kuten sovelluksia, verkkosivustoja, tietokantajärjestelmiä ja mobiilisovelluksia. Ohjelmistokehityksen tavoitteena on tuottaa laadukkaita ja toimivia ohjelmistoja, jotka vastaavat käyttäjien tarpeita ja ovat yhteensopivia erilaisten laitteistojen ja käyttöjärjestelmien kanssa. (Rouse, 2022.)

Teknologisen uudistuksen tarve lähtee usein asiakkaan tunnistetusta tarpeesta, jonka jälkeen tunnistettu tarve muuttuu liiketoiminallisiksi määrittelyiksi, joiden perusteella lähdetään kehittämään uutta ohjelmistoa. Uudistuksen toteuttamiseksi tarvitaan projekti tai hanke, joka johtaa muutoksen asetetuista tavoitteista kohti konkreettista toteutusta. Ketterässä ohjelmistokehityksessä tämä muutoksen hallinta tapahtuu projektimaisessa viitekehyksessä, joka eroaa päivittäisestä operatiivisesta johtamisesta. (Sweeney, 2022.)

Ohjelmistokehityksen ketterät periaatteet perustuvat ketterän ohjelmistokehityksen julistukseen (Manifesto for Agile Software Development), joka on julkaistu vuonna 2001 (Beck ym., 2001). Ketterillä menetelmillä tarkoitetaan ohjelmistokehityksestä tutuksi tulleella kehitys ja johtamismallia, jossa on tarkoituksena saada aikaan nopealla syklillä tavoiteltu teknologinen lopputulema esimerkiksi ohjelmisto (Dybå ym., 2014, s.11).

Monien tutkimusten mukaan ohjelmistokehitysprojekteissa ketterät menetelmät ovat nykyään suositumpia kuin aiemmin enemmän käytetty vesiputousmalli, sillä ne ovat osoittautuneet tehokkaiksi projektien onnistumisen kannalta. Voidaankin todeta, että ketterät menetelmät ja vesiputousmalli eroavat toisistaan suunnittelun lähestymistavassa, projektin edistymisessä ja asiakkaan roolissa projektin aikana. Ketterät menetelmät tarjoavat joustavuutta ja nopeaa kehitystä, kun taas vesiputousmalli tarjoaa yksityiskohtaisen suunnitelman ja tiukemman hallinnan projektin aikana. Ketterien menetelmien käyttö ei kuitenkaan automaattisesti takaa projektin onnistumista, vaan niiden käyttö edellyttää tarkkaa harkintaa ja huolellista suunnittelua. (Sweeney, 2022.)

Tämän tutkimuksellisen kehittämistyön kohteena on suomalaisen finanssialan yrityksen käynnissä oleva ohjelmistokehitysprojekti. Tämä ohjelmistokehitysprojekti toteutetaan ketterin ohjelmistokehityksen menetelmin. Uuden kehitettävän ohjelmiston tavoitteena on korvata olemassa oleva vanha järjestelmä uudella nykyaikaisella ohjelmistolla sekä saattaa yrityksen asiakkaille suunnatut palvelut digiaikaan ja vastata koronan kiihdyttämän muutokseen asiakaskäyttäytymisen muutoksessa kohti digitalisempaa tulevaisuutta. Toimittajana projektissa toimii suuri kansainvälinen IT-alan yritys ja tilaajan organisaation tuotetiimi vastaa liiketoiminnallisten vaatimusten täyttymisestä. Ohjelmistokehitysprojektissa voidaan käyttää valmisohjelmistoja, joita voidaan myös muokata asiakkaan tarpeiden mukaiseksi tai voidaan tehdä kokonaan uusi ohjelmisto täysin alusta alkaen (It-wiki, 2023). Tässä projektissa uusi ohjelmisto ja sen taustajärjestelmät toteutetaan rakentamalla ohjelmistot alusta alkaen, ilman valmisohjelmistoja.

Muutosajureina tässä ohjelmistokehitysprojektissa toimii halu päästä eroon vanhasta ohjelmistosta sekä sen taustajärjestelmistä, sillä ne eivät enää palvele nykyisiä liiketoiminnan tarpeita. Lisäksi vanhojen tietojärjestelmien ylläpitäminen ja päivittäminen ovat erittäin kallista. Tämän perusteella on tultu siihen tulokseen, että tarvitaan täysin uusi järjestelmä sekä siihen liittyvät taustajärjestelmät, joiden avulla tilaaja organisaatio mahdollistaa tulevaisuudessa asiakkailleen toimialansa parhaan digitalisen palvelun.

Ohjelmistonkehitysprojektin tavoitteena on nostaa automatisoinnin astetta mahdollisimman korkealle ja siirtää työntekijöiden voimavarat rutiininomaisten työtehtävien sijaan asiantuntijatyöhön. Uuden tietojärjestelmän tavoitteena on olla mahdollisimman käyttäjäystävällinen sekä tehostaa sisäisiä ja ulkoisia prosesseja.

Tämän ohjelmistokehitysprojektin sisäänrakennetun laadun kehittäminen on ajankohtainen aihe tutkijalle sekä organisaatiolle, sillä tutkimuksen tekijä työskentelee kyseisessä ohjelmistokehitysprojektissa määrittelijänä. Ohjelmistokehitysprojektin ensimmäinen vaihe on tulossa päätökseen vuoden 2023 aikana kun MVP (minimun viable product) julkaistaan. Projekti tulee kuitenkin jatkumaan heti tämän ensimmäisen vaiheen jälkeen, sillä ensimmäisessä vaiheessa julkaistavassa tuotteessa ei ole kaikkia tulevaisuuden liiketoiminnassa vaadittavia ominaisuuksia. Joten tämän tutkimuksellisen kehittämistyön toteuttaminen juuri nyt on hyvin ajankohtaista, jotta tutkimuksen tuloksia voidaan hyödyntää projektin seuraavassa vaiheessa.

1.2 Opinnäytetyön tarkoitus, tavoitteet ja tutkimuskysymykset

Tutkimuksen kohteena olevan ohjelmistokehitysprojektin ensimmäisen vaiheen toteutuksen aikana on huomattu, että hyväksymistestaukseen tulevista toiminallisuuksista löytyy melko paljon virheitä, eli bugeja. Tämän lisäksi toiminallisuudet eivät aina myöskään toimi kuten niiden on määritelty toimivan, mikä voi johtua esimerkiksi teknisistä rajoitteista sekä riippuvuuksista muihin järjestelmiin. Näitä löytyneitä virheitä on jouduttu korjaamaan jälkikäteen, josta on seurannut ongelmia projektin aikatauluun sekä kustannukset ovat nousseet alkuperäisestä.

Demingin (Agile alliance, 2023) mukaan testaus ei takaa tai paranna laatua, vaan testausvaiheessa esiin tuleva laatu on jo rakennettuna tuotteeseen sen toteutusvaiheessa. Laatua ei voida myöskään testata tuotteeseen tai palveluun, vaan se tulee rakentaa siihen jo toteutusvaiheessa. Tämän ketterän laadun periaatteen mukaisesti kyseisessä ohjelmistokehitysprojektissa halutaan

parantaa ohjelmiston sisäänrakennettua laatua, jotta testausvaiheessa ilmenevien virheiden määrä olisi pienempi. Tästä johtuen on tunnistettu tarve löytää keinoja rakennettavan ohjelmiston sisäänrakennetun laadun parantamiseen, jotta nämä ohjelmiston virheet huomattaisiin yhä aiemmin ja voitaisiin myöskin korjata ennen hyväksymistestausta. Tämä säästäisi kustannuksia ja mahdollistaa laadukkaampien toiminallisuuksien nopeamman julkaisun tilaajaorganisaatiolle ja sen ulkoisille asiakkaille projektin seuraavassa vaiheessa.

Opinnäytetyön tarkoituksena on selvittää millaisia menetelmiä ja käytäntöjä voidaan käyttää ketterässä ohjelmistokehitysprojektissa sisäänrakennetun laadun kehittämiseen. Lisäksi halutaan ymmärtää, miten ketteriä menetelmiä tulisi hyödyntää, jotta sisäänrakennettua laatua saadaan parannettua juuri tässä ohjelmistokehitysprojektissa. Opinnäytetyön tavoitteena on tuottaa konkreettisia kehitysehdotuksia, miten nykyisiä toimintamalleja tulisi kehittää sisäänrakennetun laadun parantamiseksi projektin seuraavassa vaiheessa. Kehitysehdotusten on tarkoitus olla helposti sovellettavissa käytäntöön.

Jotta ketterän ohjelmistokehitysprojektin sisäänrakennettua laatua voidaan kehittää onnistuneesti, niin tulee ymmärtää millaisia ketteriä menetelmiä ja toimintatapoja voidaan käyttää sisäänrakennetun laadun parantamiseen ketterässä ohjelmistokehityksessä. Opinnäytetyön teoreettisessa katsauksessa tarkastellaan aiheesta aiemmin tehtyjä tutkimuksia sekä kirjallisuutta, jotta voidaan tunnistaa mitkä menetelmät ja käytännöt sopivat parhaiten juuri sisäänrakennetun laadun kehittämiseen. Näihin menetelmiin ja käytänteisiin perehdytään opinnäytetyön teoreettisessa katsauksessa, jotta tutkija pystyy keskittymään empiirisessä tutkimuksessa tutkimuskohteena olevan ilmiön kannalta relevantteihin teemoihin.

Ketterän ohjelmistokehityksen ja sisäänrakennetun laadun kehittämisen ymmärtämiseksi tulee tutkia myös, miten näitä käytäntöjä ja menetelmiä tulisi hyödyntää, jotta sisäänrakennettua laatua voidaan parantaa juuri tutkimuksen kohteena olevassa ketterässä ohjelmistokehitysprojektissa. Tästä johtuen tämän tutkimuksellisen kehittämistyön tutkimuskysymyksenä on:

- Miten ketterän ohjelmistokehitysprojektin sisäänrakennettua laatua voidaan parantaa?

Lisäksi ketterän ohjelmistokehityksen sisäänrakennetun laadun kehittämisen kannalta on tarpeen tarkastella ketteriä periaatteita, käytäntöjä ja menetelmiä, joilla sisäänrakennettua laatua voidaan kehittää. Tämä johdosta alatutkimuskysymyksenä on:

- Mitkä ovat keskeisimpiä ketteriä menetelmiä ja käytäntöjä sisäänrakennetun laadun parantamisen näkökulmasta?

Tutkimuskysymyksiin vastaamiseksi opinnäytetyössä tarkastellaan, miten sisäänrakennettua laatua voidaan parantaa ketterässä ohjelmistokehitysprojektissa. Tutkimuksessa keskitytään erilaisten ketterien menetelmien ja käytänteiden vaikutukseen sisäänrakennettuun laatuun ja sen parantamiseen. Erityisesti tullaan keskittymään seuraaviin kokonaisuuksin: jatkuvan testauksen merkitykseen, jatkuvaan parantamiseen sekä laadunhallintamenetelmiin, kuten laadunvarmistukseen sekä jatkuvaan arvonvirtaukseen.

1.3 Tutkimusmenetelmä ja lähestymistapa

Tässä tutkimuksellisessa kehittämistyössä käytetään laadullista eli kvalitatiivista tutkimusmenetelmää. Laadullisella tutkimuksella tarkoitetaan mitä tahansa tutkimusta, jonka avulla pyritään löydöksiin ilman tilastollisia menetelmiä tai muita määrällisiä keinoja. Laadullisen tutkimuksen tarkoituksena on ilmiön kuvaaminen, ymmärtäminen ja mielekkään tulkinnan antaminen tutkimuksen kohteiden näkökulmasta. Laadullisessa tutkimuksessa tavoitteena on yksittäisen ilmiön syvällinen ymmärtäminen ja siinä ollaan kiinnostuneita erityisesti tutkimuksen kohteen ajatuksista ja käsityksistä, joita pyritään selvittämään erilaisten aineistonkeruumenetelmin. Laadullisessa tutkimuksessa tutkija on kiinnostunut prosesseista, merkityksistä ja ilmiön ymmärtämisestä sanojen, tekstien ja kuvien avulla. Laadullisessa tutkimuksesta saatava tieto on tutkijan tulkintaa kerätystä aineistosta ja sen suhteesta käsiteltyyn teoriaan.

Tarkoituksena ei ole määrällisen tutkimuksen tavoin yleistää tutkittavaa ilmiötä, vaan sen avulla halutaan saada selvyys siitä, mistä tutkittavassa ilmiössä on kyse. Laadullista tutkimusta käytetään silloin, kun tutkittavaa ilmiötä ei vielä tunneta. (Kananen 2008, s. 24-25.)

Laadullisessa tutkimuksessa on tärkeää kiinnittää huomiota tutkittavien henkilöiden valintaan. Tavoitteena on löytää informanteiksi sellaisia henkilöitä, jotka tuntevat tutkittavan ilmiön mahdollisimman hyvin. Tutkittavien määrä voi vaihdella yhdestä henkilöstä useampaan, ja joskus tutkimusasetelma voi rajoittaa valinnan vain yhteen henkilöön. Tärkeintä on, että tutkimusaineistoa on riittävästi ja että uusien tapausten lisääminen ei enää muuta tulkintaa. Laadullisessa tutkimuksessa aineiston laatu ja syvyys ovat tärkeämpiä tekijöitä kuin aineiston laajuus ja määrä. (Kananen 2008, s. 33-37.) Tämän tutkimuksellisen kehittämistyön tutkimusmenetelmäksi valittiin kvalitatiivinen tutkimusote, koska tavoitteena on saada mahdollisimman syvällinen kuva tutkittavasta ilmiöstä. Tutkimuksen kohteena olevan ohjelmistokehitysprojektin syvällisesti tuntevien henkilöiden vähäinen määrä puolestaan rajasi pois kvantitatiivisen eli määrällisen tutkimusotteen käytön.

Tässä tutkimuksellisessa kehittämistyössä käytetään lähestymistapana tapaustutkimusta eli case-tutkimusta, joka usein liitetään erityisesti laadulliseen tutkimukseen ja menetelmiin. Tapaustutkimus tutkii ilmiöitä sen luonnollisessa ympäristössä käyttäen hyväksi suppeita empiirisiä aineistoja, jotka voivat olla kvantitatiivisia tai kvalitatiivisia. Tapaustutkimuksessa perehdytään yhteen tapaukseen syvällisesti ja pyritään tuottamaan uutta tietoa sen kehittämiseen. Tapaustutkimus auttaa vastaamaan kysymyksiin ”miten” ja ”miksi” sekä mahdollistaa syvällisemmän ymmärryksen työntekijöiden välisestä vuorovaikutuksesta ja toimintatavoista sekä sopii myös erilaisten prosessien ja ihmisten käyttäytymisen tutkimiseen. Tapaustutkimus on iteratiivinen menetelmä, jossa tutkimuksen erivaiheista, voidaan palata aiempaan vaiheeseen ja edetä uudestaan täydennettyjen tietojen perusteella. (Moilanen ym., 2015, luku 3.)

Ojasalon ym. (2020, s. 36-37) mukaan tapaustutkimus soveltuu hyvin lähestymistavaksi, kun halutaan ymmärtää syvällisesti jonkin organisaation tilannetta

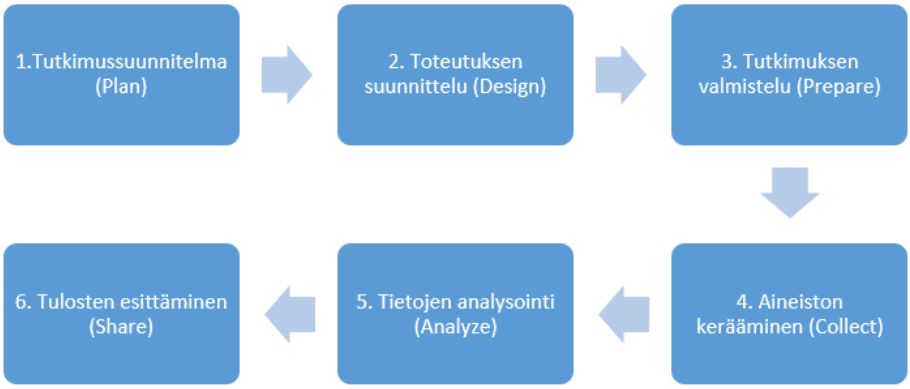
ja tavoitteena on tuottaa tutkimuksen keinoin kehittämisehdotuksia sekä uusia ajatuksia tutkittavasti ilmiöstä. Puhtaassa tapaustutkimuksessa ei käytännössä viedä muutosta eteenpäin tai varsinaisesti luoda mitään konkreettista, vaan tutkimusten tulosten avulla luodaan uutta tietoa ja kehittämisehdotuksia havaittuun ongelmaan. Tässä tutkimuksellisessa opinnäytetyössä pyrittiin selvittämään, miten juuri tämän ohjelmistokehitysprojektin sisäänrakennettua laatua saadaan parannettua ketteriä menetelmiä käyttäen. Joten tapaustutkimus soveltuukin hyvin tämän tutkimuksen tavoitteen saavuttamiseen - tuottamaan kehittämisehdotuksia ja uutta tietoa tutkimuksen kohteena olevasta ohjelmistokehitysprojektista.

Tapaustutkimus alkaa alustavan tutkimussuunnitelman (Plan) laatimisella. Alustavan tutkimussuunnitelman avulla tutkijan on helpompi hahmottaa tutkimuskohteen taustoja sekä analysoida sopivia lähestymistapoja. Suunnittelun tavoitteena on tunnistaa kehittämistehtävä tai ongelma, tunnistaa alustavat tutkimuskysymykset ja varmistua tapaustutkimuksen soveltuvan tutkimuksen lähestymistavaksi. (Yin, 2018, luku 1.) Suunnitteluvaiheessa (Design) pyritään tarkentamaan tutkimussuunnitelmaa, mikä luo vahvan perustan onnistuneelle tapaustutkimukselle. Tärkeintä on varmistaa tutkimusongelman merkityksellisyys sekä laatia yksityiskohtainen tutkimussuunnitelma, joka sisältää muun muassa tutkimuksen tarkoituksen ja tavoitteet, tutkimuskohteen määrittelyn sekä tutkimuskysymysten tarkentamisen. Lisäksi tulee pohtia, mitkä käsitteet käyvät parhaiten juuri tämän tapauksen analysointiin sekä tutustua tarkemmin kyseiseen ilmiöön liittyviin aiempiin tutkimuksiin. Suunnitteluvaiheessa on hyvä keskittyä tausta-aineiston keräämiseen ja relevanttien kirjallisuuslähteiden löytämiseen, jotta tutkija ymmärtää tutkimuksen kohteena olevaa ilmiötä mahdollisimman hyvin. Lopputuloksena syntyy tutkimussuunnitelma sisältäen myös aikataulun. (Yin, 2018, luku 2.)

Tutkimuksen valmisteluvaiheessa (Prepare) pyritään varmistamaan, että kaikki oleellinen on otettu huomioon ennen aineiston keräämistä. Tässä vaiheessa tarkistetaan usein alkuperäinen suunnitelma ja sen toimivuus tutkimuskohteeseen. Lisäksi valmisteluvaiheessa tulee valita tutkimuksen aineiston-

keruunmenetelmä, joka sopii tutkimuskysymykseen vastaamiseen mahdollisimman hyvin. Tarpeen vaatiessa tutkimuskysymyksiä voidaan täsmentää alakysymysten avulla tai harkita lisätutkimuksen tarvetta. (Yin, 2018, luku 3.) Ojasalon ym. (2020, s. 52-54) mukaan tapaustutkimuksen kehittämiskohde voi täsmentyä tai muuttua tutkimusprosessin aikana. Lisäksi heidän mukaan tulee varmistaa, että tutkija on tutustunut tutkimuksen kohteena olevaan ilmiöön tarpeeksi ennen siirtymistä tutkimuksen empiiriseen vaiheeseen.

Empiirinen tutkimus aloitetaan aineiston keräämisellä (Collect), jossa pyritään aloittamaan aineiston keruu valituilla menetelmillä. Tutkimuksen laadun varmistamiseksi on suositeltavaa kerätä tutkimusaineisto useiden eri menetelmien avulla ja eri lähteistä, noudattaen valitun tutkimustavan periaatteita. Kerätty aineisto validoidaan vertaamalla sitä tieteelliseen kirjallisuuteen ja lähdeaineistoon. Laadun parantamiseksi on hyödyllistä perehtyä kehityskohteen ja tutkimusosuuteen liittyvään kirjallisuuteen ennen keräämisvaiheen aloittamista. Lisäksi ennen tutkimuksen keräämistä tulee olla selkeä ja systemaattinen keräämisvaiheen suunnitelma, joka vastaa kysymyksiin: milloin, kuka, mitä ja miten. (Yin, 2018, luku 4.) Tietojen analysointivaiheessa (Analyze) pyritään käsittelemään ja analysoimaan kerättyä tietoa systemaattisesti sisällönanalyysin, kuten esimerkiksi teemoittelun avulla. Tavoitteena on varmistaa, että kerätty aineisto on riittävän laaja ja syvä sisällöltään, jotta sitä analysoimalla voidaan saada vastauksia tutkimuskysymyksiin. (Yin, 2018, luku 5.)
Tulosten esittämisvaiheessa (Share) pyritään esittämään tutkimustulokset ja niiden pohjalta tehty analyysi mahdollisimman objektiivisesti (Yin, 2018, luku 6). Lopuksi aiemman tutkimustiedon ja empiirisessä tutkimuksessa tehtyjen havaintojen perusteella voidaan laatia kehitysehdotuksia (Ojasalo ym., s. 52-54). Tapaustutkimuksen vaiheet esitetty visuaalisesti kuviossa 1.



Kuvio 1. Tapaustutkimuksen vaiheet (Yin, 2018).

Schultzen ja Avitalin (2011, s.1-16) mukaan haastattelut ovat hyvä keino kerätä tutkimusaineistoa, kun halutaan varmistaa tutkijan ja informanttien välitön vuorovaikutus. Haastattelut soveltuvatkin heidän mukaansa esimerkiksi kyselylomaketta paremmin, kun halutaan kartoittaa tutkittavien kokemuksia ja tulkintoja tutkimuksen kohteena olevasta ilmiöstä. Tässä tutkimuksellisessa kehittämistyössä on tarkoituksena haastatella tutkimuksen kohteena olevassa ohjelmistokehitysprojektissa työskenteleviä henkilöitä, jotta kyseisessä projektissa työskentelevät asiantuntijat pääsevät esittämään omia kokemuksiaan ja näkemyksiään tutkittavasta ilmiöstä. Haastateltavat henkilöt valitaan monipuolisesti projektin erilaisista rooleista, jotta saadaan laajempi näkemys projektin nykyisistä toimintamalleista ja miten niiden koetaan vaikuttavan sisäänrakennetun laadun rakentamiseen. Haastatteluissa ilmi tulleita projektin toimintamalleja ja havaittuja ongelmia peilataan aiheeseen liittyvään kirjallisuuteen sekä aiempiin tutkimuksiin ja näin pyritään löytämään kehitysehdotuksia, joiden avulla projektissa rakennettavan ohjelmiston sisäänrakennettua laatua voidaan parantaa. Aineistonkeruumenetelmiä, tutkimuksen toteuttamista sekä tutkimusaineiston analysointimenetelmiä käsitellään luvussa neljä. Haastatteluiden tulokset sekä pohdinta empiiristen tulosten ja aikaisempien tuloksien yhteneväisyyksistä ja eroista käydään läpi luvussa viisi. Lopuksi luvussa kuusi esitetään tuloksista tehtyjä johtopäätöksiä sekä tutkimuksen tavoitteena olleita kehitysehdotuksia.

1.4 Teoreettinen viitekehys, aiemmat tutkimukset ja rajaukset

Tutkimuksen teoreettinen viitekehys koostuu aiemmista tutkimuksista tai kirjallisista lähteistä ja keskeisten käsitteiden määrittelystä, jotka ovat olennaisia tutkittavan aiheen kannalta. Teoreettinen viitekehys auttaa tutkijaa valitsemaan näkökulman, josta hän tarkastelee aihetta, ja toimii ohjaavana tekijänä koko tutkimusprosessin ajan. Teoreettista viitekehystä voidaan myös kutsua tietoperustaksi, joka auttaa tutkijaa ymmärtämään tutkimuskentän käsitteitä ja teorioita. (Ojasalo ym., 2020, s. 34–35.)

Tämän tutkimuksen teoreettinen viitekehys perustuu ketteriin periaatteisiin, ketterän ohjelmistokehityksen menetelmiin ja käytänteisiin, sisäänrakennetun laadun käsitteeseen sekä ketterin menetelmin ja käytännöin tapahtuvaan sisäänrakennetun laadun kehittämiseen liittyviin tutkimuksiin ja muihin kirjallisiin lähteisiin. Tämän tutkimuksen teoreettinen viitekehys tarjoaa ymmärryksen ketterän ohjelmistokehityksen ja sisäänrakennetun laadun välisestä suhteesta sekä siitä, miten sisäänrakennettua laatua voidaan parantaa ketterien menetelmien ja käytänteiden avulla. Tutkimuksen teoreettinen viitekehys on esitetty kuviossa 2.

Kuvio 2. Tutkimuksen keskeiset käsitteet ja niiden suhteet.

Ketterät periaatteet ovat toimintatapoja, joissa korostetaan joustavuutta, nopeaa palautetta ja jatkuvaa parantamista. Näihin ketteriin periaatteisiin perustuu myös ketterä ohjelmistokehitys, joka on saanut alkunsa ketterän ohjelmistokehityksen julistuksesta, joka on julkaistu vuonna 2001 ja tunnetaan nimellä ”Manifesto for Agile Software Development” (Beck ym. 2001). Ketterä ohjelmistokehitys on puolestaan kattotermi joukolle menetelmiä ja käytäntöjä, jotka perustuvat ketterän ohjelmistokehityksen julistukseen ja sen taustalla oleviin 12:een ketterään periaatteeseen. Ketterät menetelmät ovat nousseet suosituiksi viime vuosina, koska ne mahdollistavat nopean ja joustavan kehityksen ja soveltuvat erityisesti monimutkaisten ja muuttuvien projektien hallintaan. (Agile alliance, 2023.) Ketteriä menetelmiä ja niiden taustalla olevia ketteriä periaatteita käsitellään tarkemmin luvussa kaksi.

Scaled Agile Frameworkin (2021) mukaan kyky toimittaa uusia toiminallisuuksia lyhimmällä kestävällä läpimenoajalla ja mukautua nopeasti muuttuviin liiketoimintaympäristöihin riippuu ohjelmiston laadusta. Laatu on tärkeä osa ketterää ohjelmistokehitystä, ja se tarkoittaa ohjelmiston kykyä täyttää asiakkaan vaatimukset ja odotukset sekä toimia tarkoitetulla tavalla. Ketterässä ohjelmistokehityksessä laatu on koko tiimin vastuulla ja se tulee sisällyttää kehitysprosessin kaikkiin vaiheisiin eli toisin sanoen laatu rakennetaan ohjelmiston sisään. Sisäänrakennettu laatu onkin yksi tärkeimmistä ketterien menetelmien arvoista. Briandin ym. (2006, s. 69-85) mukaan sisäänrakennettu laatu tarkoittaa laatua, joka on osa ohjelmiston kehitysprosessia ja joka on integroitu ohjelmistoon sen koko elinkaaren ajan. Tämä tarkoittaa, että ohjelmiston laatu ei ole vain testien tai tarkistusten lopputulos, vaan se on osa koko kehitysprosessia ja koodin laatua arvioidaan jatkuvasti. Sisäänrakennettu laatu kattaa esimerkiksi ohjelmiston arkkitehtuurin ja suunnittelun, koodin laadun, jatkuvan testauksen, dokumentaation ja ylläpidettävyyden.

Tätä laatua voidaan todentaa laadunvarmistuksella. Laadunvarmistus ketterässä ohjelmistokehityksessä tarkoittaa menetelmiä ja käytäntöjä, joita käytetään varmistamaan, että kehitettävä ohjelmisto vastaa asiakkaan tarpeita ja on korkealaatuista. Laadunvarmistuksen tulisi olla jatkuvaa ja myös se tulisi sisällyttää ohjelmistokehitysprojektiin varmistamaan, että tuote on korkealaatuinen

ja täyttää asiakkaan odotukset ja vaatimukset. Ketterässä ohjelmistokehityksessä laadun varmistaminen ei ole pelkästään tiimin vastuulla, vaan myös asiakkailla on merkittävä rooli laadunvarmistuksessa. Asiakkaat voivat antaa palautetta tuotteesta ja auttaa tiimiä ymmärtämään, mitä he haluavat. Tämä auttaa tiimiä kehittämään tuotetta sekä parantamaan toimintaansa itsearvioinnin ja jatkuvan parantamisen kautta, jotta lopputulos vastaa asiakkaiden tarpeita ja odotuksia. Toisaalta taas erinomaisellakaan laadunvarmistuksella ei pystytä parantamaan jo rakennetun ohjelmiston laatua, vaan ainoastaan toteamaan laatu, oli se sitten hyvä tai huono. (Agile Alliance, 2001: Beck ym., 2001.)

Ketterän ohjelmistokehityksen sisäänrakennetun laadun kehittämiseen on olemassa useita menetelmiä ja käytäntöjä, kuten esimerkiksi: jatkuva arvovirta, jatkuva testaus sekä koodin tarkastus ja arviointi. Nämä menetelmät ja käytännöt auttavat varmistamaan ohjelmiston laadun koko kehitysprosessin ajan ja takaavat, että ohjelmisto vastaa asiakkaan vaatimuksia. (Niazi ym., 2018, s. 32-60.) Sisäänrakennettua laatua, sen kehittämistä ja siihen liittyviä menetelmiä ja käytäntöjä käsitellään luvussa kolme.
Tutkimuskirjallisuudessa on tutkittu ketterän kehityksen sisäänrakennetun laadun kehittämistä useissa eri konteksteissa. Esimerkiksi Malik ym. (2018) tutkivat laadun varmistusta ketterässä kehityksessä ja korostivat, että laadunvarmistus on integroitava osaksi kehitysprosessia. Lisäksi Mustafee ja Taylor (2016) tarkastelivat ketterän kehityksen käytäntöjä laadunvarmistuksen näkökulmasta ja esittivät käytännön esimerkkejä, joita voidaan soveltaa ketterän kehityksen sisäänrakennetun laadun kehittämiseen.

Sánchez-Gordónin ym. (2018) tutkimuksessa ”Assessing the impact of agile practices on software quality” arvioitiin, miten ketterät menetelmät vaikuttavat ohjelmiston sisäänrakennettuun laatuun sekä käyttäjän havaitsemaan laatuun. Tutkimuksessa havaittiin, että ketterien käytäntöjen noudattaminen on yhteydessä parempaan ohjelmiston laatuun, ja erityisesti asiakastyytyväisyyteen. Tutkimus tarjoaa tärkeää tietoa ketterän ohjelmistokehityksen laadun parantamisesta ja voi auttaa organisaatioita ymmärtämään paremmin ketterän kehityksen vaikutuksia laatuun.

Lisäksi, ketterän kehityksen sisäänrakennetun laadun kehittämistä koskevaan kirjallisuuteen kuuluu myös erilaisia käytännön oppaita ja työkaluja, kuten Agile Alliance:n ketterän kehityksen periaatteet (Agile Alliance, 2001), Schwaberin ja Sutherlandin tekemä Scrum –viitekehyksen opas (2020) sekä Scaled Agile Frameworki:n (2021) SAFe-opas.

Näistä esimerkeistä ilmenee, että ketterän ohjelmistokehityksen sisäänrakennettuun laatuun liittyvää tutkimusta on tehty viime vuosina enenevissä määrin ja tutkimuksia on tehty erilaisista näkökulmista sekä käyttäen erilaisia lähestymistapoja. Lisäksi ketterästä ohjelmistokehityksestä sisäänrakennetun laadun ja sen parantamisen näkökulmasta on Suomessa tehty lukuisia Pro Gradu –tutkielmia sekä opinnäytetöitä, joskin niiden määrä finanssialalla on melko vähäinen.

Ketterien menetelmien käyttö on yleistynyt viime vuosina myös muilla aloilla kuin ohjelmistokehityksessä, kuten projektinhallinnassa ja liiketoiminnan kehittämisessä. Siltikin tämä tutkimus on rajattu koskemaa ainoastaan ohjelmistokehitysprojekteja, jotka toteutetaan ketterin menetelmin. Näin ollen tutkimuksen ulkopuolella on rajattu kaikki muut kuin ohjelmistokehitysprojektit sekä myös ohjelmistokehitysprojektit, jotka toteutetaan vesiputousmallilla ketterien menetelmien sijaan. Lisäksi tutkimuksessa keskitytään ketteristä menetelmistä vain Scrum- ja SAFE -viitekehyksiin, sillä ne ovat yleisimmät ketterän ohjelmistokehityksessä käytettävät viitekehykset ja myöskin tutkimuksen kohteena olevassa projektissa käytössä olevat viitekehykset. Tässä tutkimuksessa keskitytään projektin läpiviennin vaiheeseen, joka sisältää määrittelyn, toteutuksen sekä testauksen. Tutkimuksen ulkopuolelle rajataan mm. ohjelmistoprojektin valmistelu, toimittajan valinta sekä kehitetyn ohjelmiston ylläpito.

2 KETTERÄT PERIAATTEET JA MENETELMÄT

2.1 Ketterät periaatteet

Sanan Ketteryys (Agile) tarkka määrittely on hankalaa, koska se on yleisesti käytetty kattotermi tarkemmin määritellyille menetelmille ja käytännöille. Joidenkin tutkijoiden mukaan Ketteryys on määriteltävissä filosofiaksi. Cockburnin ja Highsmithin (2001, s. 120-127) mukaan ketterä tarkoittaa sitä, että on ”tehokas ja ohjattava. Ketterä prosessi on sekä kevyt, että myöskin riittävä. Keveys tarkoittaa sitä, että pysytään ohjattavana. Riittävyys on tarpeen siinä määrin, että pysytään pelissä mukana.” Ohjattavuudella tarkoitetaan sitä, että ketterissä projekteissa ollaan valmiita muuttamaan suuntaa, kun tarve vaatii. Boehm ja Turner (2005) puolestaan kuvailee ketteryyttä nopean mallintamisen ja kehityskokemuksen jatkumona, sekä filosofian uudelleen herättämisenä siitä, että ohjelmointi on kädentaito eikä teollinen prosessi. Vuonna 2001 ketterän ohjelmistokehityksen julistuksen ”Manifesto for Agile Software Development” seurauksena perustettiin kansainvälinen voittoa tavoittelematon organisaatio Agile Alliance, joka on omistautunut edistämään ketteriä menetelmiä ohjelmistokehityksessä ja muilla aloilla. Agile Alliance (2023) puolestaan määrittelee Ketteryyden olevan kyky luoda muutoksia ja vastata niihin sekä tapa käsitellä epävarmaa ja myrskyisää ympäristö sekä onnistua siinä lopulta.

Yleisesti nykyisten ketterien ohjelmistokehityksien menetelmien perustana pidetään vuonna 2001 julkaistua ketterän ohjelmistokehityksen periaatteiden ja arvojen kokoelmaa, joka tunnetaan nimellä ”Manifesto for Agile Software Development”. Tämä ketterän ohjelmistokehityksen julistus syntyi, kun 17 kokenutta ohjelmistokehittäjää tapasivat Utahissa vuonna 2001 keskustellakseen ohjelmistokehityksen parantamisesta. Heidän tarkoituksensa oli löytää parempia tapoja kehittää ohjelmistoa ja luoda kestävämpiä ohjelmistokehityksen käytäntöjä. Tässä kokouksessa syntyi asiakirja, joka kuvaa kehitystä, jossa tuotekehityksen painopiste on toimivassa ohjelmistossa, joka on kykeneväinen mukautumaan muuttuviin vaatimuksiin. (Fowler & Highsmith, 2001 s. 28-35.)

Agile Manifesto sisältää neljä arvoa, jotka korostavat ketterän ohjelmistokehityksen periaatteita. Nämä arvot ovat:

1. Yksilöt ja vuorovaikutus prosessien ja työkalujen sijaan.
2. Toimiva ohjelmisto kattavan dokumentaation sijaan.
3. Asiakasyhteistyö sopimusten noudattamisen sijaan.
4. Muutokseen reagoiminen suunnitelman seuraamisen sijaan.

Näiden arvojen tarkoituksena on korostaa ihmisten välistä vuorovaikutusta, toimivan ohjelmiston merkitystä, asiakkaan roolia kehitysprosessissa ja joustavuuden tärkeyttä suunnitelman noudattamisen sijaan, jotta lopputuloksena olisi laadukkaampi ja paremmin asiakkaan tarpeisiin vastaava ohjelmisto. (Agile Alliance, 2001.)

Näiden neljän arvon pohjalta luotiin myös 12 ketterää periaatetta, joiden tarkoituksena on kuvastaa käytännön toimenpiteitä, jotka auttavat ohjelmistokehitystiimejä noudattamaan Agile Manifeston arvoja. Agile Manifeston (Agile Alliance, 2001) mukaan ketterien menetelmien tulisi noudattaa seuraavia 12:ta periaatetta:

1. Tärkein tavoite on korkea asiakastyytyväisyys nopealla ja jatkuvalla ohjelmisto
2. Muutokset ovat tervetulleita jopa hyvin myöhäisessä vaiheessa.
3. Toimivaa ohjelmistoa toimitetaan säännöllisesti tiiviissä yhteistyössä asiakkaan kanssa.
4. Yhteistyö liiketoiminnan ja kehitystiimin välillä.
5. Projekti tulee rakentaa motivoituneiden yksilöiden ympärille ja heillä tulee olla onnistumista tukeva työympäristö
6. Vuorovaikutus kasvokkain on tehokkain tapa kommunikoida.
7. Toimiva ohjelmisto on edistymisen tärkein mittari.
8. Ketterät toimintatavat edistävät ohjelmistokehitystä, tiimin tulisi pystyä ylläpitämään tasaista työtahtia.
9. Jatkuva teknisen laadun huomiointi sekä jatkuva suunnittelu edistävät ketteryyttä.

10. Yksinkertaisuus – keskitytään tärkeimpiin asioihin.
11. Itseorganisoituvat tiimit kehittävät parhaiten arvoa asiakkaalle.
12. Säännölliset tiimin itsearvioinnit auttavat kehittämään toimintatapoja.

Nämä arvot ja periaatteet ovat siis molemmat tärkeitä ketterän ohjelmistokehityksen käytännön toteutuksessa. Arvot ohjaavat ketterän kehityksen kulttuuria ja mielentilaa, kun taas periaatteet tarjoavat konkreettisia käytännön ohjeita ja toimenpiteitä, joita tiimit voivat noudattaa. Highsmithin ja Fowlerin (2001, s. 28-35) mukaan nämä periaatteet ovat auttaneet ohjelmistokehittäjiä ja organisaatioita parantamaan ohjelmistojen laatua, lyhentämään kehitysaikoja ja lisäämään asiakastyytyväisyyttä. Vaikka nämä ketterät periaatteet eivät olekaan ketteryyden virallinen määritelmä, niin ne tarjoavat ohjenuoran miten laadukasta ohjelmistokehitystä voidaan tehdä ketterällä tavalla. Nämä Agile Manifeston ketterät arvot ja periaatteet inspiroineet monia ketteriä menetelmiä, joita käytetään tänä päivänä ohjelmistokehityksessä, kuten Scrum, Kanban ja Lean. Lisäksi Agile Manifesto on myös innoittanut laajemmin ketteriä toimintatapoja käytettäväksi kaikenlaisissa organisaatioissa, ei vain ohjelmistokehityksessä (Sutherland & Schwaber, 2011.)

2.2 Ohjelmistokehityksen menetelmät

Ohjelmistokehitysmenetelmät jaetaan nykyään kahteen eri pääluokkaan: perinteisiin kehitysmenetelmiin ja ketteriin menetelmiin. Perinteiset ohjelmistokehitysmenetelmät tarkoittavat sellaisia menetelmiä, joissa kehitysprosessi etenee vaiheittain vain yhteen suuntaan ja ennen seuraavan vaiheen käynnistymistä edellinen vaihe päättyy, kuten vesiputousmallissa. Ketterät ohjelmistokehitysmenetelmät puolestaan etenevät syklisesti ja ohjelmistoa kehitetään iteroiden, joka tarkoittaa, että ohjelmisto suunnitellaan ja kehitetään pienissä palasissa, ja työtehtäviin palataan syklisesti vastaten vesiputousmallin portaittaisia vaiheita. Tarkoituksena on mahdollistaa nopeampi reagointi ohjelmiston muuttuviin vaatimuksiin. (Sweeney, 2022.)

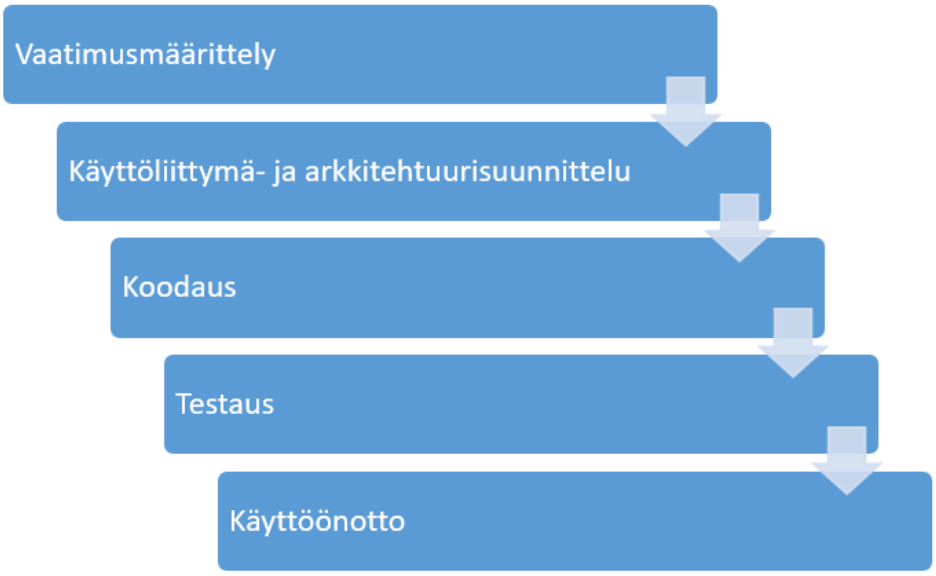
Sekä perinteinen ohjelmistokehityksen prosessi, että ketterä ohjelmistokehityksen prosessi sisältää seuraavat samat vaiheet:

1. Vaatimusten määrittely (requirement analysis)
2. Suunnittelu (design)
3. Toteutus (coding and implementation)
4. Testaus (testing and debugging)
5. Julkaisu ja ylläpito (release and maintance)

Näiden yhteisten vaiheiden lisäksi ketterissä ohjelmistokehityksen menetelmissä suurta roolia näyttelee asiakkaiden antama palaute, jota ei puolestaan perinteisissä menetelmissä juurikaan hyödynnetä. Asiakaspalautteen avulla ohjelmistoa voidaan kehittää kohti asiakkaan vaatimuksia. (Kannan ym., 2014.) Seuraavaksi käydään tarkemmin läpi perinteisiä ohjelmistokehitysmenetelmiä ja niiden periaatteita, jotta voidaan ymmärtää paremmin miten ketterät menetelmät eroavat niistä, kun ketteriin menetelmiin syvennytään tarkemmin alaluvussa 2.2.2.

2.2.1 Perinteiset menetelmät

Perinteiset ohjelmakehityksen menetelmät seuraavat tarkemmin ennalta määriteltyä polkua, jossa edetään ennakoitavasti ja suunnitellusti. Perinteisissä menetelmissä tavoitteet määritellään yleensä projektin alkuvaiheessa, jolloin niiden saavuttaminen perustuu vahvasti huolelliseen vaatimusmäärittelyyn ja suunnitteluun, ja kesken prosessin ei sallita muutoksia. Tämä perinteinen ohjelmistokehityksen vaiheittain etenevä prosessi on esitetty kuviossa 3 tämän kappaleen jälkeen. Teoreettisesti tämän lähestymistavan pitäisi tuottaa toimiva ohjelmisto nopeammin, mutta ohjelmistokehitys ei ole kuitenkaan ihan niin yksinkertaista ja suoraviivaista, sillä esimerkiksi yksi havaitsematon pieni virhe kehitysprosessin alussa voi saada koko projektin epäonnistumaan, kun sitä ei olekaan huomattu ajoissa. (Kannan, ym. 2014; Sweeney, 2022.)



Kuvio 3. Perinteinen ohjelmistokehityksen prosessi (Sweeney, 2022).

Kuten ylläolevasta kuvasta kolme on huomattavissa, niin ohjelmistokehitysprosessi käyttäen vesiputousmallia etenee vaatimusmäärittelystä, aina jokaisen vaiheen kautta käyttöönottovaiheeseen asti lineaarisesti. Tämä perinteinen ohjelmistokehityksen portaittainen malli on saanut alkunsa teollisuudesta, sillä siinä tuote aluksi suunnitellaan erittäin huolellisesti ja vasta tämän jälkeen siirrytään tuotteen valmistamiseen, kuten esimerkiksi auton valmistuksessa. Vesiputousmalli on näistä portaittaisista malleista suosituin ohjelmistokehityksen menetelmä ja sitä käytettäänkin usein esimerkkinä portaittaisista malleista. Larman ja Basil (2003, s.47-56) ovat pitäneet vaiheittain etenevää ohjelmistokehityksen mallia ongelmallisena jo 1970-luvulta lähtien koska:

1. Loppukäyttäjä ei useinkaan tiedä tarkalleen, mitä hän haluaa tuotteelta.
2. Kehitystyön aikana paljastuu yleensä monia yksityiskohtia, joita ei voitu tietää ennen toteutusvaihetta, vaikka kaikki vaatimukset olisi tiedostettu.
3. Ihminen ei kykene hallitsemaan suuren järjestelmän monimutkaisuutta, vaikka kaikki kehityksen aikana paljastuneet yksityiskohdat olisi tiedossa.

4. Vaikka monimutkaisuus pystyttäisiin hallitsemaan, ulkoiset vaatimukset voivat johtaa muutostarpeisiin ohjelmistossa, mikä voi edellyttää aiempien päätösten peruuttamista.

Ratkaisuna näihin ongelmiin Larman ja Basil (2003, s.47-56) esittelivät iteratiivisen ja inkrementaalisen ohjelmistokehityksen mallin jo 1970-luvulla. Valitettavasti näitä vaihtoehtoisia lähestymistapoja ei otettu tarpeeksi vakavasti vielä silloin, ja siksi kestikin 30 vuotta ymmärtää, että tämä on varsin tehokas tapa kehittää ohjelmistoja. Vaikka perinteisiä ohjelmistokehitysmenetelmiä onkin kritisoitu jo pitkään ja monet parannusehdotukset ovat olleet samankaltaisia kuin nykyiset ketterät menetelmät, niin edelleen ohjelmistokehityksessä käytetään myös perinteisiä portaittaisia malleja. (Abbas ym., 2002).

2.2.2 Ketterät menetelmät

Ketteriä menetelmiä voidaan yleisesti kuvata mainitsemalla peruskäytännöt, joita eri menetelmät jakavat. Craig Larmanin (2004) mukaan ketteriä menetelmiä ei ole mahdollista määritellä tarkasti, koska erilaiset käytännön vaihtelevat. Kuitenkin lyhyet ajallisesti rajatut iteraatiot, joissa suunnitelmia ja tavoitteita sopeutetaan kehityksen aikana ovat peruskäytäntöjä, joita ketterät menetelmät jakavat. Boehm ja Turner (2005) puolestaan antaa käytännönläheisemmän määritelmän, jonka mukaan ketterät menetelmät ovat erittäin kevyitä prosesseja, jotka käyttävät lyhyitä iterointisyklejä, osallistavat käyttäjiä vaatimusten määrittämiseen, priorisointiin ja varmistamiseen sekä nojaavat tiimin sisäiseen hiljaiseen tietoon täydellisen dokumentaation sijaan. Lisäksi kaikkien ketterien menetelmien tulee noudattaa Agile Manifeston neljää arvoa ja kahtatoista periaatetta, jotka käytiin läpi aiemmin luvussa 2.1.

Kannanin ja ym. (2014) mukaan ketterät menetelmät ovat luonteeltaan joustavia ja mukautuvia. Niissä ei ole välttämättä valmista suunnitelmaa alusta asti, vaan niitä kehitetään ja sovelletaan dynaamisesti projektin edetessä. Ketterät menetelmät mahdollistavat muutosten tekemisen projektin vaatimuksiin ja tavoitteisiin nopeasti ja joustavasti ilman, että ne haittaavat prosessin loppuun

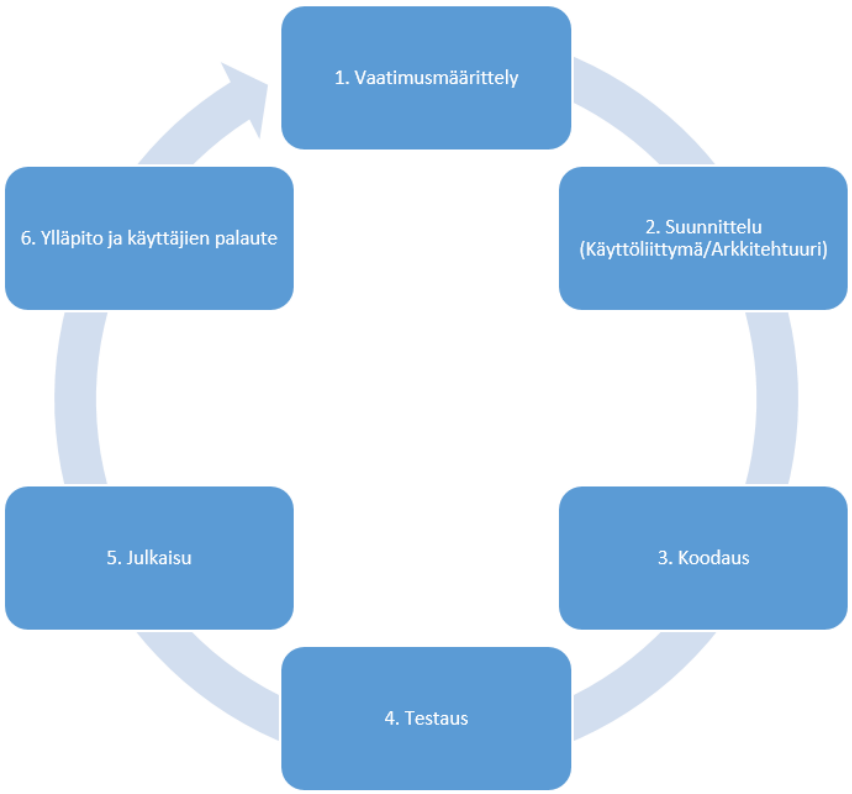
viemistä. Tämä tekee ketteristä menetelmistä erittäin hyödyllisiä projekteissa, joissa vaatimukset ja olosuhteet voivat muuttua nopeasti

Ketterä ohjelmistokehitys perustuu iteraatioihin eli aikaväleihin, joissa suunnitellaan, määritellään, toteutetaan ja testataan ohjelmistoa. Jokaisen iteraation lopussa ohjelmisto on valmistunut osittain ja seuraava iteraatio aloitetaan uusilla tavoitteilla. Ketterät menetelmät mahdollistavat ohjelmiston kehittämisen askel kerrallaan, jolloin kehittäjät voivat oppia ja sopeutua muutoksiin matkan varrella. Tärkeä osa ketterää menetelmää on myös asiakaspalaute, jota kerätään jokaisen iteraation vaiheessa. Asiakaspalautteen avulla ohjelmistoa voidaan kehittää ja muuttaa seuraavissa iteraatioissa vastaamaan paremmin asiakkaan tarpeita. Tämä mahdollistaa ohjelmiston kehittämisen asiakkaan tarpeiden ja vaatimusten mukaisesti inkrementaalisesti ja joustavasti. (Larman & Basil, 2003.)

Aiemmin esitettyjen määritelmien pohjalta voidaankin mielestäni todeta ketterien menetelmien olevan sarja kehitysmenetelmiä, jotka ovat syntyneet ketterien periaatteiden pohjalta ja ne ovat kevyitä prosesseja sekä sisältävät lyhyitä iteratiivisia syklejä. Boehmin ja Turnerin (2005) mukaan, jotta menetelmä on aidosti ketterä, niin sen tulee täyttää kaikki seuraavat ominaisuudet:

1. Joustava: pystyy reagoimaan ohjelmiston muuttuviin vaatimuksiin sekä prosessit ja periaatteet syntyvät mieluummin projektin aikana kuin etukäteen.
2. Iteratiivinen: ohjelmisto kehitetään useissa lyhyissä ajanjaksoissa, joissa jokaisessa kehitetään osa järjestelmää aina suunnittelusta toimitukseen.
3. Inkrementaalinen: Uusia toiminallisuuksia kehitetään järjestelmään jokaisessa julkaisussa ja julkaisu toimitetaan asiakkaalle palautteen saamiseksi.
4. Itseorganisoituva: Tiimit päättävät itse parhaan tavan toteuttaa tuotos.

Ketterässä ohjelmistokehityksen prosessissa, joka esitetty kuviossa 4, on samat vaiheet kuin perinteisessä ohjelmistokehityksen prosessissa (kuva 3), mutta suurin ero on siinä, että vaiheet on kytketty yhteen, eikä niissä siirrytä vaiheesta aina seuraavaan. Lisäksi ketterissä menetelmissä hyödynnetään käyttäjien antamaan palautetta, jonka avulla ohjelmistoa voidaan kehittää haluttuun suuntaan seuraavissa iteraatioissa. Ketterien menetelmien yhteen kytketyt kehitysvaiheet mahdollistavat kehittäjien ja asiakkaiden pääsyn testaamaan ja arvioimaan sovelluksen jokaista osaa aikaisemmin ja tekemään päätöksiä siitä, miten sovelluksen kehityksen tulisi edetä. Tämä tuo valtavia hyötyjä molemmille osapuolille, sillä lopputulos sisältää kaikki toimivaan sovellukseen tarvittavat suunnitteluelementit, ominaisuudet ja toiminnot. (Kannan, ym. 2014; Sweeney, 2022.)



Kuvio 4. Ketterän ohjelmistokehityksen prosessi (Sweeney, 2022).

Ketterien ohjelmistokehitysmenetelmien suosio on kasvanut huomattavasti vuoden 2001 jälkeen, kun ketterän ohjelmistokehityksen julistus julkaistiin. Mo-

nia erilaisia ketteriä ohjelmistokehitysmenetelmiä on kehitetty vastaamaan erilaisten organisaatioiden tarpeisiin ja toimintatapoihin. Vaikka nämä menetelmät eroavat toisistaan, niin niiden yhteinen tavoite on kyky sopeutua nopeasti ohjelmistokehityksen muuttuviin vaatimuksiin, jotka voivat ilmetä kehitysprosessin aikana. Agilen Alliancen mukaan (2001) ohjelmistokehityksessä ketteristä menetelmistä tällä hetkellä suosituimpia ovat mm. Scrum, Extreme Programming ja Feature-Driven Development.

Vuonna 2018 Stack Overflow -sivustolla toteutettu kysely paljasti, että yli 85% ammattimaisista sovelluskehittäjistä käyttää ketteriä ohjelmistokehitysmenetelmiä, joista suosituin oli Scrum 63 prosentin osuudella. Perinteisiä ohjelmistokehityksen menetelmiä käytti vain hieman alle 15% (Stack Overflow, 2018). Scrum menetelmään perehdytään tarkemmin luvussa 2.3.

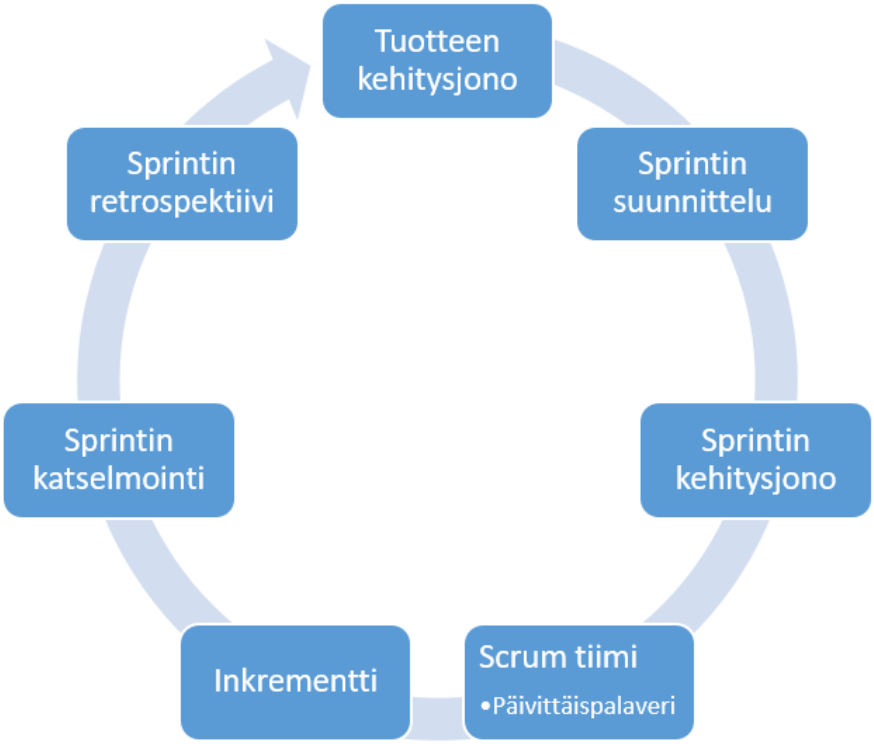
Kun huomattiin, että ketterät menetelmät toimivat hyvin ohjelmistokehityksessä pienissä tiimeissä, alettiin tutkia näiden menetelmien soveltuvuutta myös suurempiin ja hajautettuihin organisaatioihin. Leffingwellin (2007, s. 87-88) mukaan, useiden ketterän kehityksen pääperiaatteiden on havaittu skaalautuvan hyvin myös näihin suuriin ja hajautettuihin organisaatioihin. Hänen mukaansa suurimpana haasteena on ollut, että olemassa olevat ketterät ohjelmistokehitysmenetelmät ovat keskittyneet lähinnä pienten tiimien tarpeisiin, eivätkä ole välttämättä soveltuvia suurempien organisaatioiden tarpeisiin. Kun organisaatio kasvaa, niin sen prosessit, käytännöt ja mallit monimutkaistuvat, mikä voi tehdä ketterän kehittämisen filosofian toteuttamisesta haastavaa. Tähän ongelmaan on kehitetty ratkaisuksi ketterän ohjelmistokehityksen viitekehyksiä, kuten Scaled Agile Framework eli SAFe, joka on suunniteltu erityisesti suurille organisaatioille. SAFe viitekehystä tarkastellaan tarkemmin luvussa 2.4.

2.3 Scrum

Scrum on ketterän projektinhallinnan viitekehys, joka mahdollistaa monimutkaisten ongelmien hallinnan ja korkeimman mahdollisen lisäarvon tuottavien

tuotteiden toimittamisen asiakkaalle. Scrum viitekehyksen luojien Ken Schwaberin ja Jeff Sutherlandin (2020, s.3) mukaan Scrum ei ole pelkästään prosessi tai tekniikka tuotteiden rakentamiseen, vaan enemmänkin viitekehys, jonka avulla kehitystiimi voi jatkuvasti arvioida tuotetta, sen kehitystapoja ja työn tehokkuutta.

Nykyään Scrum on suosituin ketterän ohjelmistokehityksen menetelmä, kuten edellisessä luvussa jo Stack Overflown (2018) tekemän laajan tutkimuksen pohjalta jo todettiinkin. Se tarjoaa kehyksen iteratiiviselle ja inkrementaaliselle ohjelmistokehitykselle sekä korostaa erityisesti jatkuvaa parantamista ja vuorovaikutusta kehitystiimin ja asiakkaan välillä. Scrum-viitekehys perustuu empirismiin, joka tarkoittaa kokemukseen perustuvaa prosessinhallintaa ja faktoihin perustuvaa päätöksentekoa. Tärkeimmät empirismin tekijät ovat läpinäkyvyys, tarkastelu ja sopeuttaminen. Läpinäkyvyys tarkoittaa sitä, että kaikki merkittävät tekijät ovat näkyvissä projektin osapuolille, ja se edellyttää yhteistä ymmärrystä tärkeistä käsitteistä, kuten "valmiin" määritelmästä. Säännöllinen tarkastelu auttaa projektin jäseniä seuraamaan edistymistä ja havaitsemaan mahdollisia poikkeamia, jotka ovat syytä selvittää mahdollisimman nopeasti. Scrum-viitekehykseen kuuluu neljä muodollista tapahtumaa: Sprintin suunnittelu, päivittäispalaveri, sprintin katselmointi ja retrospektiivi. (Schwaber & Sutherland 2020, s. 3–5.) Kuviossa 5 esitetään Scrumin-viitekehyksen prosessin kulku, tapahtumat ja tuotokset.



Kuvio 5: Scrum-viitekehyksen prosessin kulku, tapahtumat ja tuotokset (Schwaber & Sutherland, 2020).

Ensimmäisenä vaiheena on tuoteomistajan (Product Owner) luoma tuotteen kehitysjono, joka sisältää vaatimukset tuotteen kehittämiseksi. Tämän jälkeen kehitystiimi valitsee sprintin suunnittelussa, mitkä vaatimukset otetaan mukaan seuraavaan sprinttiin ja ne siirretään sprintin kehitysjonoon. Sprintin aikana tiimi toteuttaa valitut vaatimukset ja pitää päivittäispalaverin, jossa käydään läpi sprintin edistyminen ja mahdolliset ongelmat. Sprintin lopussa tiimi pitää sprintin katselmoinnin, jossa esitellään kehitystiimin rakentama tuote eli inkrementti jonka tulisi täyttää "valmiin" määritelmä. Lopuksi tiimi pitää retrospektiivin, jossa arvioidaan sprintin onnistumista ja etsitään tapoja parantaa prosessia tulevissa sprinteissä. Tärkeät tuotokset prosessissa ovat tuotteen kehitysjono, sprintin kehitysjono ja inkrementit (Schwaber & Sutherland, 2020, s. 3-5.) Scrumin roolit esitellään tarkemmin luvussa 2.3.1, Scrumin tapahtumat luvussa 2.3.2 sekä Scrumin tuotokset luvussa 2.3.4.

2.3.1 Scrumin roolit

Scrum-viitekehys sisältää kolme erilaista roolia, jotka ovat kaikki tärkeä osa Scrum-tiimiä: tuoteomistaja (Product Owner), Scrum Master ja kehitystiimi (Development Team). Scrum:issa jokaisella roolilla on tarkkaan määritelty vastuualue ja velvollisuudet projektin onnistumiseksi, ja jokainen jäsen, riippumatta omasta roolistaan tai taidoistaan, kuuluu Scrum-tiimiin. Kehitystiimin kokoonpano tulee muuttua projektin vaatimusten ja tarvittavan osaamisen mukaan, mikä tarkoittaa, että Scrum ei tarjoa yhtä ainoaa tiimirakennemallia kaikille projekteille. (Schwaber & Sutherland, 2020, s. 5.)

Kehitysprojektin onnistuminen riippuu suuresti tuoteomistajan roolista, joka vastaa kehitettävän tuotteen arvon maksimoinnista, kehitystiimin työn järjestämisestä sekä vastaa myös kehitysjonon (Product Backlog) hallinnasta. Schwaberin ja Sutherlandin (2020) mukaan on erittäin tärkeää, että koko organisaatio kunnioittaa tuoteomistajan päätöksiä, jotta hän voi menestyksekkäästi hoitaa tehtävänsä. Lisäksi he painottavat myös, että tuoteomistajan tulee olla yksi henkilö eikä ryhmä ihmisiä, jotta vastuu ja päätöksenteko ovat selkeitä ja tehokkaita. (Schwaber & Sutherland, 2020, s. 5-6.)

Scrum Masterin vastuulla on ohjata ja opettaa Scrum-tiimin jäseniä Scrumin periaatteiden, käytäntöjen ja sääntöjen noudattamisessa. Tämä sisältää myös ulkopuolisten auttamisen ymmärtämään, miten he voivat edistää Scrum-tiimin toimintaa. Scrum Master ei kuitenkaan ole mikään valvoja, vaan pikemminkin valmentaja, joka auttaa tiimiä saavuttamaan parhaan mahdollisen suorituskyvyn. Scrum Masterin tehtävänä on myös auttaa tiimiä tunnistamaan ja poistamaan esteitä, jotka voivat hidastaa tai estää projektin etenemistä. Lisäksi Scrum Master varmistaa, että Scrum-tiimin kaikki jäsenet ymmärtävät tuoteomistajan tavoitteet ja tarpeet kehitysjonon kohtien takana, sekä julkaisujen laajuuden. Scrum Master auttaa myös tuoteomistajaa kehittämään ja ylläpitämään tuotejonoa sekä kommunikoimaan tiimille selkeästi ja tehokkaasti. Tavoitteena on varmistaa, että tiimi on täysin tietoinen tuotteen tavoitteista ja tarpeista, jotta se voi suunnitella ja toteuttaa tarvittavat tehtävät onnistuneesti. (Schwaber & Sutherland, 2020, s. 6-7.)

Kehitystiimi on Scrum-rooli, jonka jäsenet vastaavat tuoteversion kehityksestä eli muuttavat kehitysjonon sisällön tuotteeksi asiakkaalle. Kehitystiimissä voi esimerkiksi työskennellä koodareita, IT-arkkitehteja sekä testaajia. Kehitystiimiä kannustetaan työskentelemään autonomisesti ja organisoimaan omaa työtään, jotta tiimin synergiaedut voivat parantaa suorituskykyä ja tuottavuutta. Kehitystiimille on tärkeää olla itseohjautuva ja monitaitoinen, ja jokainen jäsen on kehittäjä, riippumatta vastuualueestaan. Kehitystiimi jakaa yhteisen vastuun kehitystyöstä, eikä siihen kuulu alatiimejä. (Schwaber & Sutherland, 2020, s. 5.)

2.3.2 Scrumin tapahtumat

Scrumissa on ennalta sovitut ja aikarajatut tapahtumat, jotka on suunniteltu lisäämään projektin läpinäkyvyyttä ja mahdollistamaan tarkastelua sen edistymisen aikana. Scrum-viitekehyksen ydin on sprintti, joka on ennalta sovittu ajanjakso, jonka aikana tiimi keskittyy yhden tuotteen ominaisuuden, tavoitteen tai julkaisun kehittämiseen. Sprintin kesto on enintään yksi kuukausi, ja sen aikana tiimin tavoitteena on tuottaa valmiin määritelmän täyttävä inkrementti eli tuotos. Tämä tuotos voi koostua yhdestä tai useammasta sprintin aikana tehdystä julkaisusta. Schwaberin ja Sutherlandin (2020) mukaan sprintin keston tulee pysyä samana koko tuotteen kehityksen ajan. Laceyn (2015) mukaan sprintin kestolle ei ole yleispätevää suositusta, vaan sen määrittelee aina kehitystiimi. Nykyään monet kehitystiimit suosivat sprinttien keston lyhentämistä kahteen viikkoon tai jopa viikkoon, koska se vähentää riskiä, että työ ei valmistu ajoissa tai ei vastaa asiakkaan tarpeita. Kuitenkin sprintin keston määrittelyssä tärkeintä on varmistaa, että se on suhteutettu projektin kokoon ja sen vaatimuksiin. Sprintit ovat jatkuvasti käynnissä ja uusi sprintti seuraa aina edellistä ja sisältää suunnittelupalaverin, päivittäiset palaverit, kehitystyön, sprintin katselmoinnin ja sprintin retrospektiivin. (Schwaber & Sutherland, 2020, s. 7-8.)

Jokainen sprintti aloitetaan aina sprintin suunnittelupalaverilla (Sprint Planning), johon kaikki tiimin jäsenet osallistuvat. Suunnittelupalaverissa määritellään, mitä sprintin aikana tehdään sekä ja miten tämä valittu työ saadaan toteutettua valmiiksi inkrementiksi. Tavoitteena on määrittää realistiset tavoitteet sprintille ja suunnitella työ niin, että se voidaan saattaa valmiiksi sprintin loppuun mennessä. Päivittäispalaverissa (Daily Scrum Meeting) koko Scrum-tiimi kokoontuu jokaisena sprintin päivänä korkeintaan 15 minuutin ajaksi ja käy läpi edellisen päivän työn edistymisen, sekä ennustaa mitä tulee tapahtumaan seuraavaan päivään mennessä. Jokainen kehitystiimin jäsen vastaa vuorollaan kolmeen kysymykseen: mitä tein eilen auttaakseni kehitystiimiä saavuttamaan sprintin tavoitteen, mitä aion tehdä tänään auttaakseni kehitystiimiä saavuttamaan sprintin tavoitteen, ja onko mitään esteitä, jotka estävät minua tai kehitystiimiä saavuttamasta sprintin tavoitetta. Päiväpalaveri auttaa kehitystiimiä pysymään ajan tasalla työn edistymisestä, vähentää riskejä ja auttaa ratkaisemaan mahdollisia ongelmia ajoissa. (Schwaber & Sutherland, 2020, s. 8-9.)

Sprintin katselmointi (Sprint Review) tapahtuu sprintin lopussa ja sen tarkoitus on arvioida, onko sprintin aikana kehitetty inkrementti saavuttanut sprintin tavoitteen ja Scrum-tiimin määrittelemän valmiin-määritelmän. Katselmoinnissa käydään läpi kehitetty inkrementti ja annetaan palautetta sen laadusta ja toimivuudesta. Tämän tiedon avulla katselmoinnin osallistujat pystyvät antamaan palautetta ja yhdessä pohtimaan, miten tuotetta voidaan kehittää edelleen ja optimoida sen arvoa. Katselmoinnin tarkoituksena on arvioida sprintin onnistumista ja edistää yhteistyötä ja kommunikaatiota Scrum-tiimin ja sidosryhmien välillä. (Schwaber & Sutherland, 2020, s. 9.)

Sprintin retrospektiivissä (Sprint Retrospective) Scrum-tiimi arvioi sprintin aikana käytettyjä prosesseja, työkaluja ja kommunikointia, ja pyrkii löytämään keinoja työn parantamiseksi seuraavaa sprinttiä varten. Retrospektiivissä keskitytään erityisesti prosessien parantamiseen ja kehittämiseen, jotta tulevissa sprinteissä tiimi pystyy työskentelemään entistä tehokkaammin ja tuottavammin. Tavoitteena on tunnistaa ongelmakohdat ja kehittämistarpeet sekä luoda toimintasuunnitelma niiden korjaamiseksi. Retrospektiiviin osallistuu koko

Scrum-tiimi ja se pidetään yleensä heti sprintin katselmoinnin jälkeen ennen seuraavan sprintin suunnittelupalaveria. (Schwaber & Sutherland, 2020, s. 10.)

2.3.3 Scrumin tuotokset

Scrum-viitekehyksessä tuotoksia on kolme: tuotteen kehitysjono, sprintin kehitysjono ja inkrementti. Nämä tuotokset edustavat tuotteen kehityksen työmäärää tai arvoa ja niiden tarkoituksena on lisätä läpinäkyvyyttä kehitystyön edistymisestä. Läpinäkyvyys mahdollistaa Scrumin tuotosten arvioinnin ja sopeuttamisen tarvittaessa, mikä auttaa tiimiä saavuttamaan parempia tuloksia. Tuotosten avulla kaikki tiimin jäsenet voivat ymmärtää yhteisen vision ja päämäärän ja työskennellä yhdessä kohti yhteistä tavoitetta. Tuotteen kehitysjono (Product Backlog) on priorisoitu lista, joka kuvaa kehitettävän tuotteen vaatimuksia ja sisältää kaikki ominaisuudet, toiminnot, vaatimukset, parannukset ja korjaukset, jotka ovat tarpeellisia tuotteen kehityksen kannalta. On tärkeää huomata, että tuotteen kehitysjono ei ole koskaan täysin valmis, vaan se kehittyy jatkuvasti tuotteen kehittyessä sekä palautteen ja ympäristön muutosten perusteella. Tuotteen kehitysjonossa olevat kohteet, jotka voidaan toteuttaa yhden sprintin aikana, tunnetaan nimellä "valmiit toteutukseen" (ready for implementation). Tiimi valitsee nämä toteutukseen valmiit kohteet sprintin suunnittelupalaverissa ja siirtää ne sprintin kehitysjonoon. Tuotteen kehitysjonoa tulee jatkuvasti jalostaa, jotta Scrum-tiimi voi saavuttaa halutun läpinäkyvyystason. Jalostamisen tarkoituksena on pilkkoa ja tarkentaa kehitysjonon kohteita, jolloin ne muuttuvat tarkemmiksi ja pienemmiksi. Tuotteen kehitysjono on sidoksissa tuotteen tavoitteeseen, joka varmistaa sen läpinäkyvyyden ja mitattavissa olevan edistyksen. (Schwaber & Sutherland, 2020, s. 10-11.)

Sprintin kehitysjono (Spring Backlog) koostuu sprintin tavoitteesta, valmiista tuotteen kehitysjonon kohteista ja inkrementin toteutussuunnitelmasta. Kehitystiimi luo sprintin kehitysjonon ja se toimii kehitystiimin keskeisenä suunnitelmana. Sprintin kehitysjonossa esitettävien tehtävien yksityiskohtaisuuden tarkoituksena on mahdollistaa riittävän tarkat arviot työmäärästä, joka tarvitaan sprintin aikana toteutettavien tehtävien suorittamiseen. Tämä tarkoittaa sitä,

että sprintin kehitysjonossa kuvataan konkreettisia toimenpiteitä, jotka tarvitaan sprintin tavoitteen saavuttamiseksi. Kehitystiimin jäsenet vastaavat sprintin kehitysjonon tehtävien suorittamisesta, ja heidän on voitava ymmärtää, mitä kukin tehtävä edellyttää ja miten se vaikuttaa sprintin kokonaistavoitteen saavuttamiseen. Sprintin kehitysjonoa tarkistetaan ja päivitetään jatkuvasti sprintin aikana, jotta se pysyy ajan tasalla ja auttaa kehitystiimiä pysymään suunnitellussa aikataulussa. (Schwaber & Sutherland, 2020, s. 11.)

Inkrementti (Increment) eli tuoteversio on sprintin ja sitä edeltävien sprinttien valmistuva työ, joka vastaa määriteltyä valmiuden tasoa ja edistää tuotteen kehitystä kohti sen lopullista tavoitetta. Inkrementit tulee testata ja integroida osaksi tuotetta jokaisen sprintin jälkeen, jotta varmistetaan niiden yhteensopivuus ja käyttökelpoisuus. Inkrementit esitellään sprintin katselmoinnissa kaikille sidosryhmille, jossa sitä tarkastellaan myös aiempien inkrementtien muodostamassa kokonaisuudessa. (Schwaber & Sutherland, 2020, s. 11-12.)

2.4 Scaled Agile Framework (SAFe)

2.4.1 Arvot ja Periaatteet

Scaled Agile Framework (SAFe) on Dean Leffingwellin ja Scaled Agile Incorporationin vuonna 2011 esittelemä ketterän ohjelmistokehityksen viitekehys, joka tarkoittaa ketterien menetelmien skaalausta läpi organisaation läpi hyödyntäen ketteriä menetelmiä. SAFe sisältää roolit, vastuualueet ja toimintamallit, jotka tukevat ketterää ohjelmistokehitystä. Sen perusajatus on tarjota skaalautuva ja toimiva malli ketterän ohjelmistokehityksen toteuttamiseen erityisesti suurissa projekteissa ja suurissa organisaatioissa. Scaled Agile Incorporationin (2021) mukaan SAFe-viitekehys on sovellettavissa yli 50 hengen organisaatioista aina useiden tuhansien henkilöiden organisaatioihin.

SAFe -viitekehyksen arvot ja periaatteet ovat keskeinen osa sen toimintaa ja niiden välillä on vahva yhteys. Arvot ovat SAFe -viitekehyksen kivijalka, joka ohjaa organisaation kulttuuria ja päivittäistä toimintaa. Arvojen kautta pyritään

varmistamaan, että organisaatio toimii tavoitteellisesti ja tehokkaasti, sekä että kaikki työskentelevät yhteisten päämäärien saavuttamiseksi. Periaatteet puolestaan ovat käytännön ohjeita siitä, miten SAFe -viitekehyksen tavoitteet saavutetaan. Ne auttavat organisaatiota tekemään parempia päätöksiä, parantamaan kommunikointia, vähentämään riskejä ja lisäämään toiminnan läpinäkyvyyttä. Periaatteet toimivat käytännön työkaluina, jotka auttavat organisaatiota soveltamaan arvoja käytäntöön. (Scaled Agile Inc., 2021.)

SAFe -viitekehyksen neljä ydinarvoa ovat:

1. Sisäänrakennettu laatu (Built-in Quality): Laatu tulee huomioida koko ohjelmistokehitysprosessin ajan, eikä se ole pelkästään testauksen tai tuotannon vastuulla.

2. Tulosten aikaansaaminen (Program Execution): Kehityksen tulee edetä suunnitellulla tavalla hyödyntäen erilaisia ketteriä toimintatapoja. Tavoitteena on saada ohjelman eri osa-alueet toimimaan saumattomasti yhdessä ja varmistaa, että ohjelma saadaan vietyä loppuun suunnitellussa aikataulussa ja budjetissa.

3. Läpinäkyvyys (Transparency): Organisaation toiminnan tulee olla avointa. Läpinäkyvyys saavutetaan kommunikoimalla selkeästi organisaation tavoitteista, strategioista ja toimintamalleista.

4. Tekemisen kohdistaminen liiketoiminnan tavoitteisiin (Alignment): Organisaatiolla tulee olla yhteinen näkemys ja tavoitteet, jotka ohjaavat toimintaa. Strategisten tavoitteiden tulee jakautuva aina yksittäisiin tiimeihin saakka.

(Scaled Agile Inc., 2021.)

Näistä arvoista sisäänrakennettua laatua sekä sen kehittämiseen liittyviin menetelmiä ja käytäntöjä käydään yksityiskohtaisemmin läpi luvussa kolme. Sisäänrakennettu laatu ja nämä kolme muuta arvoa toimivat perustana SAFe – viitekehyksen 10 periaatteelle, jotka ovat seuraavat:

1. Taloudellinen näkökulma
2. Systeemiajattelu
3. Oleta vaihtelevuutta – Säilytä vaihtoehtoja
4. Kehitä nopeilla, integroiduilla oppimissykleillä
5. Tee päätökset perustuen toimivaan järjestelmään
6. Visualisoi työ, rajaa yhdenaikaisen työn määrää, pienennä eräkokoja ja hallitse jonoja
7. Noudata kadenssia, synkronoi eri tahojen suunnittelu
8. Vapauta tietotyöläisten sisäinen motivaatio
9. Hajauta päätöksenteko
10. Organisoidu arvon ympärille

(Scaled Agile Inc., 2021.)

Esimerkiksi periaate "Hajauta päätöksenteko" tukee ydinarvoa läpinäkyvyydestä, koska se edistää avointa kommunikointia organisaation sisällä ja auttaa kaikkia ymmärtämään, mitkä päätökset täytyy keskittää ja mitkä voidaan puolestaan hajauttaa alemmas. Toisaalta periaate "Kehitä nopeilla, integroiduilla oppimissykleillä" tukee ydinarvoa sisäänrakennetusta laadusta, koska se auttaa organisaatiota varmistamaan, että laatu on mukana kaikessa tekemisessä ja integraatiot auttavat paljastamaan systeemitason ongelmat aiemmin.

2.4.2 SAFe:n versiot

Scaled Agile Framework (SAFe) tarjoaa neljä erilaista versiota organisaatioiden eri tarpeisiin. Nämä versiot ovat Essential SAFe, Portfolio SAFe, Large Solution SAFe ja Full SAFe. Kaikki versiot perustuvat Essential SAFe -versioon, joka on SAFe:n perusversio. Essential SAFe sisältää tiimitason (team level) ja hanketason (program level). Portfolio SAFe laajentaa SAFe:n perusversiota sisältämällä portfoliotason (portfolio level). Portfoliotaso tarjoaa erilaisia

menetelmiä ja työkaluja, kuten budjetointiin ja hallinnointiin, liiketoiminta-arvojen ymmärtämiseen. Large Solution SAFe puolestaan tarjoaa laajojen ratkaisujen tason perusversioon päälle. Tämä taso on suunniteltu tukemaan monimutkaisten ja suurten järjestelmien kehitystyötä. Laajojen ratkaisujen taso tarjoaa muun muassa rooleja ja prosesseja, jotka helpottavat useiden toimitusjunien koordinointia ja yhteistyötä. Full SAFe on SAFe:n versio, joka yhdistää Large Solution SAFe- ja Portfolio SAFe -versiot. Tämä malli on suunniteltu erityisesti suurille organisaatioille, jotka tarvitsevat useita rinnakkaisia SAFe-versioita. Full SAFe tarjoaa laajan valikoiman toimintamalleja, rooleja ja prosesseja, jotka auttavat organisaatiota hallitsemaan ja koordinoimaan monimutkaisten ja suurten järjestelmien kehitystä. Organisaatiot voivat valita haluamansa version tarpeidensa mukaan. (Scaled Agile Inc., 2021.)

SAFe:n kaikissa versioissa pyritään asiakaskeskeisyyteen ja asiakas asetetaan keskeiseen asemaan organisaation kaikessa tekemisessä. Asiakaskeskeisyys tarkoittaa, että organisaation kaikessa toiminnassa keskitytään asiakkaan tarpeisiin ja vaatimuksiin. Tämä tarkoittaa, että organisaatio pyrkii ymmärtämään asiakkaiden tarpeita ja haasteita sekä varmistamaan, että sen tuotteet ja palvelut vastaavat näitä tarpeita ja haasteita mahdollisimman hyvin. Lisäksi ratkaisun suunnittelu on yksi SAFen perusperiaatteista, joka auttaa organisaatioita kehittämään parempia ja asiakaskeskeisempiä tuotteita ja palveluita. Ratkaisun suunnittelun avulla organisaatiot pyrkivät ymmärtämään paremmin asiakkaiden tarpeita ja ongelmia esimerkiksi prototyyppien luomisen ja testauksen avulla sekä ongelman määrittelyn ja ratkaisuideoiden keräämisen avulla. Ratkaisun suunnittelu avulla organisaatiot pyrkivät myös kehittämään parempia käyttöliittymiä ja käyttäjäkokemuksia, joiden avulla asiakkaat voivat käyttää tuotteita ja palveluita helpommin ja tehokkaammin. Näin organisaatiot voivat luoda lisäarvoa asiakkailleen ja erottua kilpailijoistaan. Kun asiakaskeskeisyys ja ratkaisun suunnittelu yhdistyvät, tuote voidaan kehittää parhaalla mahdollisella tavalla ratkaisemaan asiakkaan ongelma. (Scaled Agile Inc., 2021.)

2.4.3 Toimitusjuna ja sen roolit

Ketterän ohjelmistokehityksen perustana ovat noin 5-10 hengen monitaitoiset itseorganisoituvat tiimit, joita käytetään myös SAFe:ssa. SAFe:ssa ketterä tiimi koostuu tuoteomistajasta (Product Owner) Scrummasterista (Scrum Master) ja kehitystiimistä (Develepment team). Tiimi vastaa koko ratkaisun määrittelystä, rakentamisesta, testaamisesta ja mahdollisesti myös tuotantoon viennistä. Tiimin vastuulla on myös valita itselleen sopiva ketterän kehityksen malli, joka voi olla esimerkiksi Scrum, Extreme Programming (XP) tai jopa niiden osittainen yhdistelmä. (Scaled Agile Inc., 2021.)

Suurimpana erona muihin suurille organisaatioille tarkoitettuihin ketterän kehityksen malleihin verrattuna, kuten luvussa 2.3 esiteltyyn Scrum-viitekehykseen on se, että SAFe-viitekehyksen avulla toteutettujen projektien ohjelmistokehityksestä vastaa useampi kuin yksi ketterä tiimi, ja SAFe-mallissa nämä tiimit työskentelevät synkronoidusti toimitusjunassa (Agile Release Train), joka on yksi SAFe:n keskeisimmistä periaatteista. Toimitusjunaan kuuluu useita tiimejä, jotka työskentelevät yhdessä tuotteen kehittämiseksi ja julkaisemiseksi. Tyypillisesti yhdessä toimitusjunassa työskentelee 50-125 henkilöä. On tärkeää, että kaikki sidosryhmät, ohjelmat ja tiimit ovat tietoisia siitä, miten toimitusjunajuna etenee. Tämä tarkoittaa sitä, että kommunikoinnin projektin etenemisestä on oltava avointa ja läpinäkyvää, jotta kaikki sidosryhmät voivat seurata projektin tilaa ja edistymistä. Tämä voidaan saavuttaa esimerkiksi visualisoimalla projketin eteneminen ja käyttämällä portfoliotasoista Kanban-taulua, joka näyttää projektin tilanteen avoimesti kaikille sidosryhmille. Näin varmistetaan, että kaikki tietävät, mitä on tehty, mitä on tulossa ja mitä mahdollisia esteitä tai haasteita on ilmennyt. (Scaled Agile Inc., 2021.)

Tiimit suunnittelevat työnsä suunnittelupalavereissa (Program Increment Planning), joissa päätetään tiimin tekemä työ seuraavalle hankkeen työjaksolle (Program Increment). Nämä hankkeen työjaksot kestävät yleensä 8-12 viikkoa, ja sen aikana tiimit työskentelevät yhteistyössä kohti tavoitettaan. Nämä työjaksot jaetaan vielä pienempiin kokonaisuuksiin eli iteraatiohin (Iteration),

joiden kesto vaihtelee 1-4 viikon välillä. Jokaisen työjakson aikana tiimin tulisi saada valmiiksi tuotteen versio (Product Inkrement).

Tuotepäällikkö (Product Manager) vastaa yksittäisten tuotteiden kehittämisestä ja johtamisesta. He ovat vastuussa asiakastarpeiden selvittämisestä ja muuntamisesta vaatimuksiksi eli määrittelyiksi. Tuotepäällikkö määrittää tuotteen ominaisuudet (Feature) ja priorisoi ne, jotka puolestaan kehitystiimit toteuttavat, ja varmistavat, että tuote täyttää asiakkaiden tarpeet ja liiketoiminnan tavoitteet. Tuoteomistajat (Product Owner) ovat vastuussa tuotteen kehittämisestä ja sen toimittamisesta tiimille. He työskentelevät yhdessä tuotepäällikön kanssa varmistaakseen, että asiakastarpeet on määritelty ja tuotteen ominaisuudet on priorisoitu oikein. Tuoteomistajat ovat myös vastuussa käyttäjätarinoiden (Story) luomisesta ja niiden tarkemmasta määrittelystä, jotta tiimi ymmärtää tarkalleen, mitä heidän pitää toteuttaa. He varmistavat, että tiimi keskittyy työskentelemään oikeissa asioissa oikeaan aikaan, ja että tiimi tekee jatkuvaa edistystä. Tuoteomistajat toimivat myös yhteistyössä muiden tiimien tuoteomistajien kanssa yhdenmukaistaakseen priorisointia ja aikatauluja, jotta kokonaisuus pysyy hallinnassa ja läpinäkyvyys paranee. Nämä käyttäjätarinat tiimi toteuttaa itse valitsemansa ketterän kehityksen viitekehyksen mukaisesti, kuten esimerkiksi Scrum-viitekehystä käyttäen. Esimerkiksi tällöin nämä hankkeen työjaksot pilkotaan kaksi viikkoa kestäviksi sprinteiksi, joissa noudatetaan Scrum-viitekehyksen rooleja, tapahtumia ja tuotoksia. Hankkeen työjakso päättyy aina tutki ja sopeudu - tapahtumaan (Inspect & Adapt), jossa tiimit arvioivat työnsä tuloksia ja suunnittelevat seuraavaa työjaksoa. (Scaled Agile Inc., 2021.)

2.4.4 SAFe:n ongelmat

SAFe viitekehys on ollut käytössä jo reilu kymmenen vuotta, alkaen vuodesta 2011 kun se esiteltiin. Vaikka se onkin erittäin suosittu ja hyväksi todettu ketterän ohjelmistokehityksen viitekehys, niin siitä löytyy niin hyviä kuin huonojakin puolia. Esimerkiksi Dumitrou ja muut (2019) nostivat tekemässään tutkimuksessa esiin, että SAFe –viitekehys ei aina sovi kaikille organisaatiolle, sillä

sen käyttöönotto voi vaatia merkittäviä muutoksia organisaation rakenteeseen ja kulttuuriin. Lisäksi SAFe-viitekehyksen käyttöönottaminen ei ole heidän mukaansa ihan helppoa, sillä usein siinä kohdataan haasteita tiedonkulussa, uuden kulttuurin vastustamista sekä tarvittavaa osaamista on hankala löytää. Toisaalta he toteavat, että SAFe –viitekehys voi oikein käytettynä tuoda organisaatiolle monia etuja kuten nopeutettu tuotekehitys, parempi yhteistyö tiimien välillä, parempi suunnittelu- ja seurantakyky sekä syvällisempi ymmärrys organisaation kokonaisuudesta.

Evans (2019) puolestaan kirjoittaa blogissaan, että SAFe-viitekehys keskittyy enemmän prosesseihin ja konkreettisiin tuotoksiin kuin arvontuottoon. Sen keskeisin tavoite on saada ominaisuuksia toteutettua uusiin tai olemassa oleviin tuotteisiin ennalta määrätyssä ajassa. Tämä johtaa hänen mukaansa usein siihen, että ketterän ohjelmistokehityksen ydinarvo - arvon tuottaminen asiakkaalle, unohdetaan vaikka uusia ominaisuuksia kehitettäisiin nopeasti ja tehokkaasti asiakkaalle. Hän myös kritisoi uusimpaan SAFe 5.0 lisättyjä perusperiaatteita: muotoiluajattelu ja asiakaskeskeisyys sekä toteaa niiden enemmänkin vain sekoittavan kuin luovan selkeyttä. Dexter (2020) puolestaan toteaa SAFe –viitekehyksen olevan hyvin laaja ja monimutkainen malli, että harva organisaatio pystyy todella ymmärtämään sen toimintatavat ja sitä, miten sen mukaan tulisi toimia. Mallissa on paljon erilaisia tapaamisia, arvoja ja menetelmiä, joten on käytännössä haastavaa saada kaikki organisaatiossa toimivat henkilöt ymmärtämään tavoitteet samalla tavalla. Lisäksi hän nostaa ongelmaksi sen, miten tuoteomistaja on usein vain käyttäjätarinoita kehitysjonoon tuottava henkilö sekä vastaa niiden hyväksynnästä, vaikka SAFe –viitekehyksen mukaan hänen tulisi vastata toteutuksen tuottamasta arvosta.

3 SISÄÄNRAKENNETTU LAATU JA SEN KEHITTÄMINEN

Sanalla ”laatu” (Quality) on monia merkityksiä, mutta Juranin (1999, kappale 2) mukaan kaksi niistä ovat kriittisen tärkeitä, kun puhutaan laadun hallinnasta.

Hänen mukaansa laatu tarkoittaa niitä tuotteen ominaisuuksia, jotka vastaavat asiakkaan tarpeisiin sekä odotuksiin ja siten takaavat asiakastyytyväisyyden. Korkeamman laadun on tarkoitus tarjota korkeampaa asiakastyytyväisyyttä ja lisätä organisaation tuloja. Korkeamman laadun nähdään vaativan yleensä investointeja ja siten lisäävän kustannuksia, tästä johtuen se nähdään yleensä olevan kallista. Toisaalta hänen mukaansa ”laatu” tarkoittaa puutteiden ja virheiden puuttumista, jotka edellyttävät saman työn tekemisen uudelleen näiden havaittujen virheiden korjaamiseksi. Nämä virheet voivat hänen mukaansa johtaa esimerkiksi asiakastyytymättömyyteen ja korvausvaatimuksiin. Tässä mielessä laadun merkitys suuntautuu kustannuksiin, ja korkeampi laatu maksaakin yleensä vähemmän, sillä se vähentää muun muassa virheiden määrää, työn uudelleen tekemistä, vähentää testaamiseen käytettävää aikaa, nopeuttaa uusien tuotteiden kehittämistä ja parantaa kapasiteettia. Hänen mukaansa onkin tärkeä erottaa, että hyvän laadun kustannukset syntyvät etukäteen investointeina laatuun, mutta ne taas vähentävät tai jopa kokonaan poistavat huonosta laadusta johtuvat kustannukset, jotka ovat korkeampia kuin organisaatiot yleensä ymmärtävät. Nämä Huonon laadun kustannukset koostuvat kaikista kustannuksista, jotka katoaisivat, jos puutteita ei olisi - ei virheitä, ei uudelleentyötä, ei reklamaatioita.

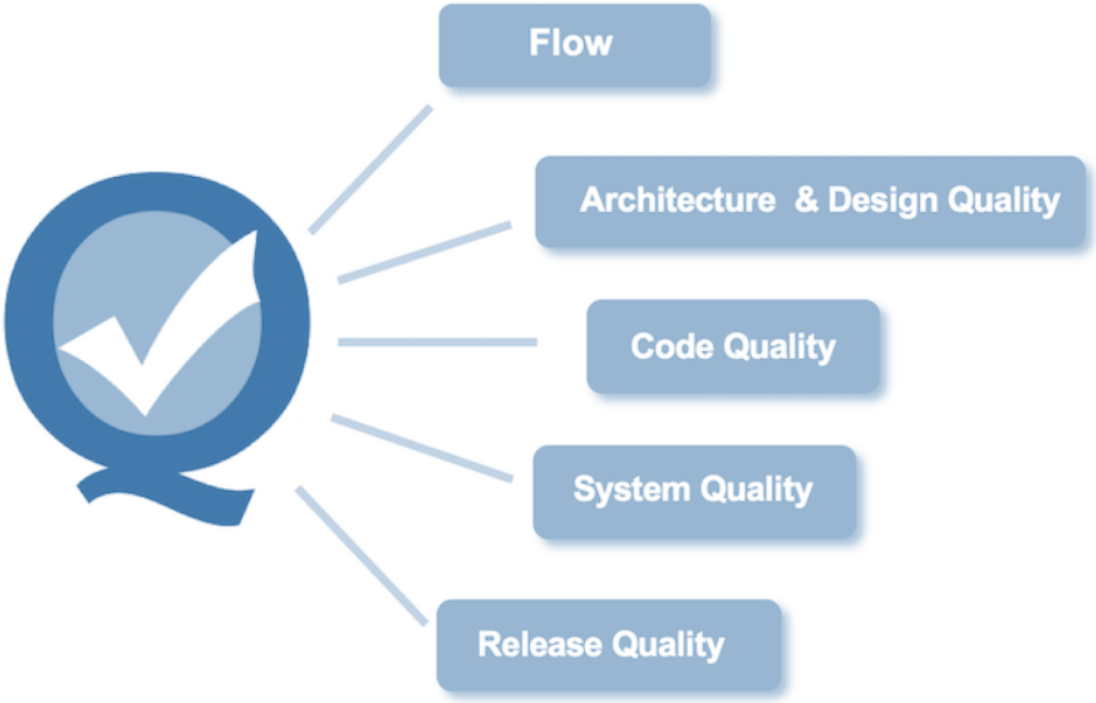
William Edwards Deming (1982) totesi laadunhallinnan periaatteita ja menetelmiä käsittelevässä Out of the Crisis –teoksessaan seuraavasti: “Inspection does not improve the quality, nor guarantee quality. Inspection is too late. The quality, good or bad, is already in the product. Quality cannot be inspected into a product or service; it must be built into it.” Tämä kuuluisa toteamus perustuukin hänen filosofiaansa, jonka mukaan laadunvarmistuksen tulisi olla ennalta ehkäisevää, eikä korjaavaa. Hän halusi korostaa, että tarkastus ei paranna laatua, vaan sen sijaan laatu on jo tuotteessa, kun tarkastus tehdään – oli laatu sitten hyvä tai huono. Siksi hänen mukaansa laatu on rakennettava tuotteeseen tai palveluun alusta alkaen sen sijaan, että sitä yritetään lisätä jälkikäteen tarkastuksen avulla.

Tutkimukset määrittelevät sisäänrakennetun laadun (Built-in Quality) hieman eri tavoin. Esimerkiksi Liu & Bai (2009, s. 236) määrittelivät sisäänrakennetun

laadun organisaation kyvyksi suunnitella, kehittää ja toteuttaa tuotteita ja palveluita, jotka täyttävät asiakkaiden tarpeet ja vaatimukset. Kun taas toisessa tutkimuksessa sisäänrakennettu laatu määriteltiin "tuotteen tai palvelun kyvyksi vastata asiakkaiden tarpeita ensimmäisellä kerralla, ilman tarvetta korjauksiin tai muutoksiin" (Shahin, ym. ,2002, s. 425).

Näiden määritelmien perusteella voidaan sanoa, että sisäänrakennettu laatu tarkoittaa sitä, että laatu on rakennettu osaksi tuotetta tai palvelua sen suunnittelusta alkaen, eikä sitä yritetä lisätä jälkikäteen tarkastuksen avulla. Tämä määritelmä perustuu ajatukseen, että laadunvarmistuksen tulisi olla osa tuotteen tai palvelun suunnittelua ja valmistusprosessia, eikä pelkästään sen lopullista tarkastusta. Lisäksi laatu tulisi rakentaa tuotteeseen ja palveluun kaikkien vaiheiden aikana, jotta lopputuloksena olisi korkealaatuinen tuote tai palvelu, ilman tarvetta korjata sitä.

Kuten aiemmin jo todettiin, niin tuotteen tai palvelun valmistuessa se on joko hyvälaatuinen tai huonolaatuinen. Yleensä suurimpana ongelmana on se, että laadun tarkka arvioiminen on usein mahdollista vasta lopputuotteen, esimerkiksi ohjelmiston valmistumisen jälkeen. SAFe-viitekehyksessä tätä ongelmaa pyritään ratkaisemaan siten, että sisäänrakennettu laatu varmistetaan jokaisessa projektivaiheessa koko tuotteen elinkaaren ajan. Jokainen kehityskierros sisältää elementtejä, jotka varmistavat, että laatu toteutuu laatustandardien mukaisesti. Laatu ei siis ole jälkikäteen lisätty ominaisuus, vaan se on sisäänrakennettu osa tuotetta tai palvelua. SAFe -viitekehyksessä sisäänrakennettu laatu jaetaan viiteen elementtiin: jatkuva kehityksen sykli (Flow), arkkitehtuurin ja suunnittelun laatu (Architecture & Design Quality), koodin laatu (Code Quality), järjestelmän laatu (System Quality) sekä julkaisujen laatu (Release Quality), jotka on esitelty tarkemmin kuviossa 7. (Scaled Agile Inc., 2021.)



Kuvio 6. Sisäänrakennetun laadun viisi elementtiä (Scaled Agile Inc., 2021).

Kuten jo aiemminkin on todettu, niin sisäänrakennetun laadun parantaminen on yksi ketterän ohjelmistokehityksen tärkeimmistä tavoitteista. Sen laadun varmistaminen ketterässä ohjelmistokehitysprojektissa vaatii jatkuvaa huomiota ja sitoutumista laadun varmistamiseen koko projektin elinkaaren ajan. Seuraavissa alaluvuissa tutustutaan erilaisiin käytäntöihin ja menetelmiin, jotka on havaittu tehokkaiksi sisäänrakennetun laadun parantamisessa ketterissä ohjelmistokehitysprojekteissa. Nämä käytännöt ja menetelmät perustuvat aikaisempiin tutkimuksiin ja niiden avulla voidaan edistää laadukkaan ohjelmiston kehittämistä ketterästi.

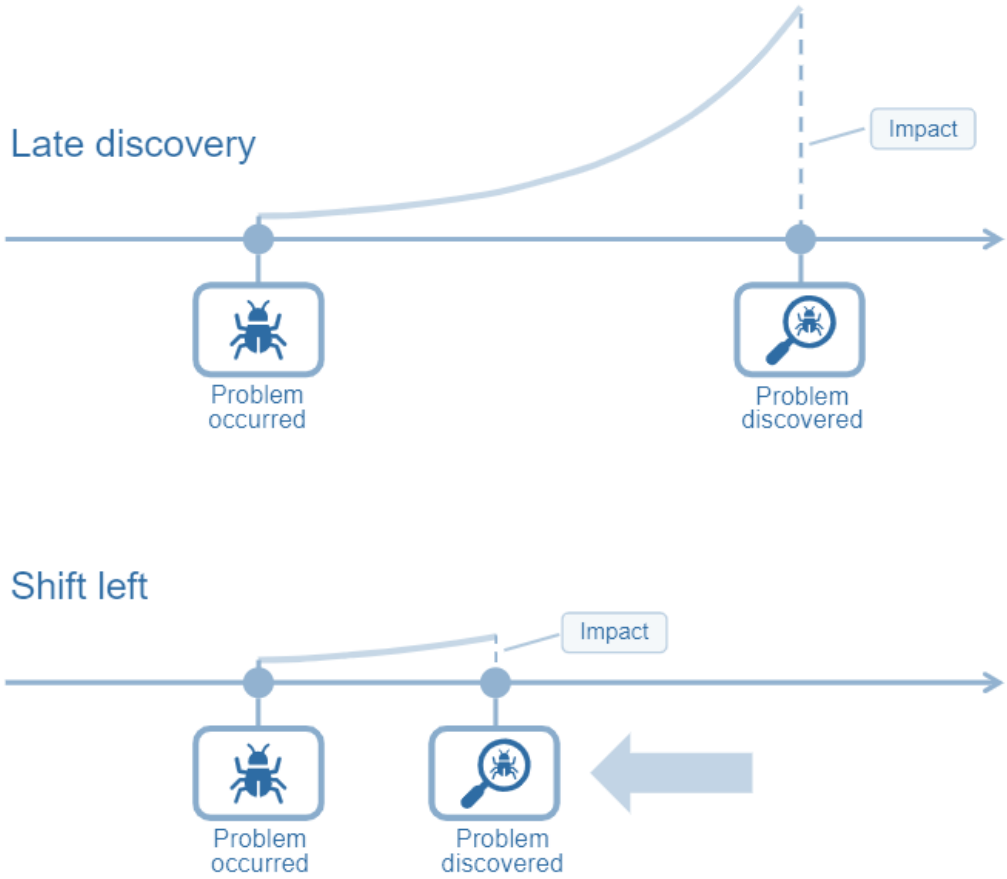
3.1 Jatkuva kehityksen sykli

Jatkuva kehityksen sykli (Flow) tarkoittaa sitä, että kehitystiimi keskittyy jatkuvan virtauksen varmistamiseen ohjelmistokehitysprosessissa. Tällä tarkoitetaan sitä, että tiimi pyrkii varmistamaan, että kaikki toiminta etenee sujuvasti ja esteettömästi projektin alusta loppuun asti. Sisäänrakennettu laatu liittyy kiinteästi jatkuvaan arvovirtaan, joka kuvataan SAFe-viitekehyksen jatkuvan ar-

vonvirtaukseen liittyvässä periaatteessa kuusi: ”visualisoi työ, rajaa yhdenaikaisen työn määrää, pienennä eräkokoja ja hallitse jonoja.” Yksi tämän jatkuvan arvonvirtauksen keskeisimpiä mittareita on virtaustehokkuus (Flow Efficiency), jolla mitataan ketterässä ohjelmistokehityksessä sitä, kuinka paljon aikaa kehitystiimi käyttää varsinaiseen toteutukseen verrattuna muihin tehtäviin, kuten odotteluun, suunnitteluun ja testaukseen. Tämä mittari auttaa arvioimaan kehitystiimin tehokkuutta ja paljastamaan pullonkauloja kehitysprosessissa. Kun virtaustehokkuus on korkea, kehittäjät voivat keskittyä olennaiseen ja työ etenee sujuvasti. Jos virtaustehokkuus on matala, se voi johtaa ajanhukkaan, viivästyksiin ja tehottomaan työskentelyyn. (Scaled Agile Inc, 2021.)

Mikkosen ja Taipaleen (2017) tutkimuksen mukaan virtaustehokkuus on tärkeä mittari ohjelmistokehityksen prosessin arvioinnissa, sillä se kertoo kuinka paljon aikaa tiimi käyttää varsinaiseen kehitystyöhön verrattuna kokonaisaikaan. Heidän tutkimuksensa osoitti, että virtaustehokkuus vaikuttaa merkittävästi kehitystiimin tuottavuuteen, sillä tehokkaasti virtaava kehitysprosessi parantaa tiimin työn sujuvuutta ja keskittymistä olennaiseen, esimerkiksi vähentämällä hukkaa prosessista, virheitä lopputuotteessa, tunnistamalla ja poistamalla pullonkauloja.

Shift left -periaate SAFe-viitekehyksessä tarkoittaa kehityksen toimintojen varhentamista tai aikaisemmaksi siirtämistä, jotta mahdollisia ongelmia voidaan havaita ja korjata mahdollisimman varhaisessa vaiheessa. Tämän periaatteen avulla ongelmien havaitseminen ja korjaavien toimenpiteiden tekeminen olisi mahdollista ennakoivasti ja jatkuvasti, mikä mahdollistaa oppimisen ja kehittymisen ajan myötä. Tämä auttaa parantamaan jatkuvan kehityksen sykliä ja välttämään viivästyksiä, joita voisi esiintyä, jos huono laatu havaitaan vasta myöhemmin kehitysprosessin aikana. Shif left –periaate visualisoitu kuviossa 7.



Kuvio 7. Shift left –periaate (Scaled Agile, Inc, 2021).

Shift left – periaatteen mukaisesti kehityksen eri vaiheita tulee toteuttaa ja testata, kuten ominaisuuksia (Feature), tarinoita (Story) ja koodia (Code), jotta mahdolliset virheet voidaan havaita ja korjata välittömästi. Tavoitteena on integroida laatu kehitysprosessiin ja varmistaa, että jokainen kehitysvaihe tuottaa laadukasta ja toimivaa koodia. Tämä auttaa välttämään virheitä ja palautteita myöhemmissä kehitysvaiheissa, jolloin kehitysprosessi on tehokkaampi ja nopeampi. Myöskin koko järjestelmän testausta kerralla kannattaa välttää, sillä se hidastaa kehitysprosessia verrattuna jatkuvaan pienempien kokonaisuuksien, esim. ominaisuuksien ja tarinoiden testaukseen. (Scaled Agile Inc., 2021.)

DoR (Definition of Ready) on SAFe-viitekehyksen käsite, joka liittyy ohjelmistokehityksen laatuun ja sen parantamiseen. DoR määrittelee kriteerit, jotka on täytettävä ennen kuin tuotekehityksen jonoon (backlog) otetaan uusi ominaisuus (Feature) tai tarina (Story). Kun DoR-kriteereitä noudatetaan, se voi aut-

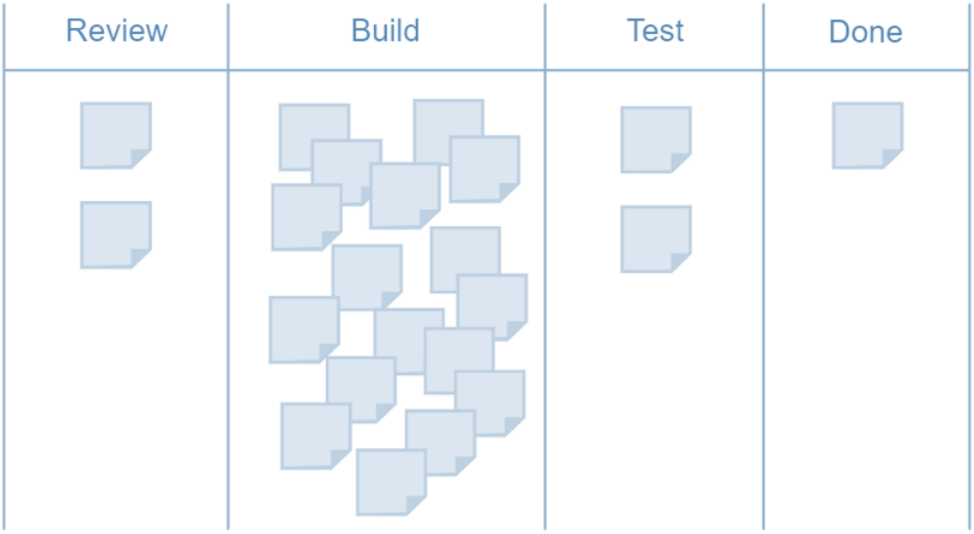
taa vähentämään virheitä ja epäselvyyksiä, jotka saattavat johtaa huonolaatuiseen ohjelmistoon. Esimerkiksi Wijaya & Raharjo (2018) havaitsivat tutkimuksessaan, että DoR-kriteerien käyttö parantaa tiimin tuottavuutta ja auttaa vähentämään hukkaa, kuten ylimääräistä odottelua ja turhaa työtä sekä parantaa tiimin tuottavuutta. Joskin he huomauttavat, että DoR-kriteerien määrittely vaihtelee organisaatioittain ja ehdottavat organisaatioita määrittelemään DoR-kriteerit yhtenäisesti ja varmistamaan niiden oikean käytön ja hyväksynnän ohjelmistokehitysprojekteissaan.

Kanban-menetelmän avulla työn virtausta voidaan seurata visuaalisesti Kanban-taululla. Tämä auttaa tunnistamaan, mitkä toiminnallisuudet (Feature) ja tarinat (Story) ovat työn alla, missä vaiheissa työ viipyy liian pitkään ja mikä hidastaa työn virtausta. Kanban-sana viittaa korttiin ja korttien määrä heijastaa keskeneräisen työn määrää. Työn määrä tulee arvioida ja rajoittaa ottaen huomioon tekijöiden taidot ja käytettävissä oleva aika. Työvaiheet esitetään visuaalisesti rinnakkain, ja työtä hallitaan jonoperiaatteella, jossa vähemmän kiireellinen työ priorisoidaan ja asetetaan jonoon odottamaan toteuttamista. Ohjelmistokehityksessä Kanban-taulua käytetään yleisesti digitaalisessa muodossa ja se on tärkeä osa kehitystiimin päivittäistä viestintää, jotta organisaatio voi ymmärtää paremmin ketterän kehittämisen menetelmiä ja nähdä työn edistyvän läpinäkyvästi. Käyttämällä elektronista Kanban-taulua, kuten esimerkiksi Jira, voidaan tehokkaasti hallita nopeasti syklistä kehitystyötä ja saada visuaalinen tilannekuva työn edistymisestä. Lisäksi projektipäällikkö voi hyödyntää Kanban-taulua erinomaisena työkaluna raportointiin, dokumentointiin ja työn ohjaamiseen. (Torkkola, 2015, s. 49-50: Agile Alliance, 2023.)

Ketterässä ohjelmistokehityksessä keskeneräisten työtehtävien määrää on tärkeä seurata, tämä seurantaa auttaa kehitystiimiä arvioimaan, kuinka paljon työtä on aloitettu, mutta ei vielä saatu valmiiksi tietyllä hetkellä. Kanban-menetelmässä tämä on yksi tärkeimmistä mittareista, ja sitä seurataan työpöydällä visuaalisesti esimerkiksi rajoittamalla keskeneräisten työtehtävien määrää. Tämä auttaa vähentämään hukkaa ja keskittymään priorisoituihin työtehtäviin. Keskeneräisen töiden hallinta auttaa parantamaan kehitystiimin tehokkuutta ja

vähentämään keskeneräisten tehtävien aiheuttamaa kaaosta. Liian suuri keskeneräisten päällekkäisten työtehtävien määrä voi hidastaa kehitysprosessia, kun taas liian pieni määrä voi johtaa resurssien alikäyttöön. (Reinertsen, 2009, s. 143-166.)

Kanban-taulu ja keskeneräisen työn rajoittaminen perustuvat jo aiemmin mainittuihin SAF:en periaatteisiin ja erityisesti periaatteeseen kuusi: ”visualisoi työ, rajaa yhdenaikaisen työn määrää, pienennä eräkokoja ja hallitse jonoja.” Seuraavassa kuvassa kahdeksan havainnollistetaan miten visualisointi eli Kanban-taulun käyttö auttaa hallitsemaan työn määrää ja näin auttavan varmistamaan optimaalisen virtaustehokkuuden. Esimerkki Kanban-taulusta esitetty kuviossa 8.



Kuvio 8 Kanban-taulu (Scaled Agile Inc, 2021).

Ylläolevasta kuviosta kahdeksan voidaan helposti huomata, että pullonkaulana tässä esimerkissä on ohjelmiston rakentaminen eli kehitystiimin koodaajat. Heidän tehtävänään on rakentaa ominaisuuksia sekä tarinoita ja siirtää niitä eteenpäin testaajille testattaviksi, ennen tuoteomistajan tai tuotepäällikön hyväksymistä valmiiksi. Ratkaisuna tähän onkin työmäärän rajoittaminen kehitysjaksoissa eli sprinteissä ja inkrementeissä, ettei kehitystiimi tee liikaa asioita päällekkäin, joka Reinertsenin (2009) mukaan voi hidastaa ohjelmistokehitystä.

3.2 Järjestelmän arkkitehtuuri ja suunnittelun laatu

Järjestelmän arkkitehtuuri ja suunnittelun laatu (Architecture & Design Quality) ovat ratkaisevan tärkeitä tekijöitä sen varmistamiseksi, että järjestelmä vastaa tämänhetkisiä ja tulevaisuuden tarpeita. Järjestelmän arkkitehtuurin ja suunnittelun laadun on oltava joustavaa ja sopeutuvaa liiketoiminnan muutosten kanssa, joten niiden hyvä suunnittelu etukäteen on ensiarvoisen tärkeää. Vaikka tulevaisuuden tarpeiden ennustaminen rakennusvaiheessa on vaikeaa, SAFe -viitekehyksen mukaan paras lähestymistapa on perustaa arkkitehtuurisuunnittelu aikaisempaan kokemukseen, mallinnukseen, simulaatioon ja kokeiluun. Näin voidaan luoda muutosvalmis arkkitehtuuri, joka on kestävä ja skaalautuva tulevaisuuden tarpeisiin. (Scaled Agile Inc., 2021.)

Ketterän kehityksen periaatteiden mukaisesti arkkitehtuurin suunnittelua kannattaa lähestyä iteratiivisesti. Tämä tarkoittaa sitä, että arkkitehtuuri suunnitellaan pienissä osissa ja sitä kehitetään jatkuvasti ohjelmiston kehityksen edetessä. Tämä mahdollistaa joustavan lähestymistavan ja mahdollistaa arkkitehtuurin sopeuttamisen nopeasti muuttuviin vaatimuksiin. Bassin ym. (2015, s. 759-780) tekemä tutkimus tarkasteli teknisen velan hallintaa ja iteratiivisen arkkitehtuurin suunnittelun vaikutusta teknisen velan vähentämiseen ja ohjelmiston laadun parantamiseen. Tutkimuksen tulokset osoittivat, että iteratiivinen arkkitehtuurin suunnittelu auttaa hallitsemaan teknistä velkaa ja parantamaan ohjelmiston suorituskykyä, luotettavuutta ja ylläpidettävyyttä. Tutkimus korosti myös, että iteratiivinen lähestymistapa auttaa tunnistamaan teknisen velan varhaisessa vaiheessa ja parantaa siten ohjelmiston laatua ja ylläpidettävyyttä.

3.3 Koodin laatu ja sen parantaminen

Koodi on järjestelmässä se, joka lopulta ratkaisee järjestelmän toimivuuden tai toimimattomuuden. Koodin muutettavuus luotettavasti ja nopeasti kertoo sen laadusta. Koodin laatua (Code Quality) voidaan varmistaa useilla eri tavoilla, joita käydään seuraavaksi läpi. Yksikkötestauksessa tavoitteena on automati-

soida koodin osia, joita pystytään testaamaan automaattisesti aina, kun muutoksia tulee. Näin voidaan olla varmoja, että lisätty koodi ei riko mitään olemassa olevaa, vaan uudet ominaisuudet toimivat vanhan päällä. Lisäksi yhteisomistajuutta pidetään myös hyvänä keinona vähentää riippuvuuksia tiimien välillä ja varmistaa, että yksittäinen kehittäjä tai tiimi ei hidasta arvon toimituksen nopeaa virtausta. Yksittäinen henkilö voi lisätä toiminallisuuksia, korjata virheitä, parantaa suunnittelua tai refaktoroida, koska koodia ei omista yksi tiimi tai yksilö. Käyttämällä hyviä koodausstandardeja kannustetaan yhtenäisyyteen, jotta jokainen voi ymmärtää ja ylläpitää jokaisen komponentin laatua. Koodin tarkastukset (Code Reviews) ovat prosessi, jossa ohjelmiston kehittäjät tarkastavat toistensa koodia virheiden ja puutteiden varalta. Tarkoituksena on parantaa koodin laatua, löytää mahdollisia virheitä aikaisessa vaiheessa ja parantaa tiimityöskentelyä. Tarkastuksia voidaan tehdä eri tavoin, kuten pariohjelmointina, ryhmäarvioina tai koodin lukemisina yksilötasolla. Koodin tarkastukset auttavat varmistamaan, että tuotettu koodi on laadukasta ja täyttää sille asetetut vaatimukset. (Scaled Agile Inc, 2021.)

Useat tutkimukset ovat osoittaneet, että koodin tarkastukset voivat merkittävästi parantaa sisäänrakennettua laatua ohjelmistokehityksessä. Esimerkiksi Nagappan ym. (2006) toteuttamassa tutkimuksessa havaittiin, että koodin tarkastukset voivat vähentää ohjelmiston virheitä jopa 60-90 prosentilla. Myös Johnsonin ja Tjahjadin (2008) tutkimuksessa havaittiin, että koodin tarkastukset voivat vähentää ohjelmiston virheitä huomattavasti, ja että nämä virheet voidaan havaita ja korjata paljon aikaisemmin kehitysprosessin aikana. Toisaalta, Rigby ym. (2014) toteuttamassa tutkimuksessa havaittiin, että koodin tarkastukset eivät välttämättä aina johda parempaan sisäänrakennettuun laatuun, vaan että niiden vaikutus riippuu monista tekijöistä, kuten tarkastusten laadusta, kehittäjien kokemuksesta ja tarkastusten ajankohdasta. Yhteenvetona voidaan todeta, että koodin tarkastukset voivat olla tehokas tapa parantaa ohjelmiston sisäänrakennettua laatua, mutta niiden vaikutus riippuu monista tekijöistä, ja niitä on toteutettava oikein ja systemaattisesti.

3.4 Jatkuva testaaminen ja automaattinen testaus

Järjestelmälaatu (System Quality) tarkoittaa sitä, kuinka varmistetaan, että järjestelmä toimii oikein ja että kehityksessä keskitytään oikeisiin asioihin. Vaikka koodin ja suunnittelun laadun avulla voidaan varmistaa, että järjestelmän osia voidaan helposti tunnistaa ja muokata, järjestelmän laadun avulla voidaan osoittaa, että järjestelmät toimivat odotetulla tavalla ja että kaikki ovat yhtä mieltä siitä, millaisia muutoksia tulee tehdä. Kun järjestelmään tehdään useita pieniä muutoksia, on tärkeää, että jokaisen muutoksen vaikutus voidaan testata nopeasti ja palautetta voidaan saada nopeasti, jotta tarvittavat korjaukset voidaan tehdä. (Scaled Agile Inc., 2021.)

Perinteisissä ohjelmistokehitysmenetelmissä odotetaan yleensä siihen asti, kunnes koko kehitettävän järjestelmän koodi on kirjoitettu ja tämän jälkeen testataan kerralla. Tämä ei sovi yhteen Scrum-viitekehyksen perusajatuksen kanssa, sillä sprintin aikana testattavaksi vietyjä kohteita ei tulisi testata vasta sprintin lopussa. Tämä aiheuttaa turhaa viivettä ja hidastaa kehitystyötä. Siksi Scrum-viitekehyksen mukaan tulisi käyttää jatkuvaa toteutuksen, testauksen, integroinnin ketjua, jossa koodin asennetaan heti valmistuttuaan testiympäristöön ja testataan mahdollisimman nopeasti valmistumisen jälkeen. Parhaat käytänteet tähän ovat jatkuva integraatio (Continuous Integration) ja jatkuva toimitus (Continuous Delivery) sekä automaattinen testaus. (Linz, 2014, kappale 5.5.)

Jatkuva integrointi tarkoittaa käytäntöä, jossa kehittäjien koodaama uusi koodi integroidaan jatkuvasti muiden kehittäjien koodiin. Tämä mahdollistaa nopean palautteen koodin laadusta, mikä parantaa ohjelmiston kokonaislaatua ja auttaa havaitsemaan virheitä ja puutteita aikaisessa vaiheessa, jolloin niiden korjaaminen on helpompaa ja halvempaa. Fowlerin (2010) mukaan hyviä työkaluja joita voidaan käyttää jatkuvassa integraatiossa ovat muun muassa automatisoidut testit.

Jatkuva toimitus on puolestaan ohjelmistokehityksen käytäntö, jossa ohjelmiston toimitusprosessi on automatisoitu ja standardisoitu siten, että ohjelmiston

toimitus voidaan suorittaa nopeasti ja luotettavasti, milloin tahansa. Tämä tarkoittaa sitä, että kehitystiimi voi toimittaa ohjelmistoa asiakkaille nopeasti ja usein pienissä erissä, mikä mahdollistaa nopean palautteen ja reagoinnin asiakastarpeisiin. Jatkuva toimitus sisältää jatkuvan integroinnin käytännön sekä muun muassa automaattiset testit. Tämä edellyttää automaattista testausta, joka tekee ohjelmistosta turvallisemman ja vähentää julkaisuprosessin riskejä. (Humble & Farley, 2010.)

Automaattinen testaus ketterässä ohjelmistokehityksessä tarkoittaa automatisoitujen testien käyttöä ohjelmiston toiminallisuuksien ja laadun varmistamiseen. Ketterässä kehityksessä korostetaan nopeaa palautetta, joten automaattinen testaus on tärkeä osa kehitysprosessia. Testit pyritään tekemään mahdollisimman aikaisin kehitysprosessin aikana, jotta mahdolliset virheet voidaan havaita ja korjata nopeasti. Automaattisten testien avulla varmistetaan myös kehityksen jatkuvuus ja vähennetään manuaalisen testauksen aiheuttamaa työmäärää. Automaattinen testaus ketterässä ohjelmistokehityksessä voi sisältää yksikkötestejä, integraatiotestejä, hyväksymistestejä ja regressiotestejä. Yksikkötestit testaavat yksittäisiä komponentteja, kuten metodeja ja funktioita. Integraatiotestit testaavat komponenttien yhteistoimintaa ja hyväksymistestit varmistavat ohjelmiston toimivuuden asiakkaan vaatimusten mukaisesti. Regressiotestit puolestaan varmistavat, että ohjelmiston muutokset eivät vaikuta aiemmin testattuihin toiminallisuuksiin. (Fowler, 2010.)

Jatkuvan integroinnin, jatkuvan toimituksen sekä automaattisen testaamisen menetelmät parantavat sisäänrakennettua laatua ketterässä ohjelmistokehitysprojektissa monella tavalla. Niiden avulla kehittäjät voivat nopeasti havaita virheitä ja puutteita, jolloin ne voidaan korjata ennen kuin ne aiheuttavat suurempia ongelmia. Lisäksi niiden avulla voidaan varmistaa ohjelmiston jatkuva toimivuus ja yhteensopivuus eri ympäristöissä. Tämä vähentää riskiä siitä, että ohjelmisto ei toimi oikein tai että sen käyttöönotto vie liikaa aikaa tai resursseja. (Scaled Agile Inc, 2021.)

Valmiin määritelmä (Definition of Done) on SAFe-viitekehyksen käsite, joka liittyy ohjelmistokehityksen laadukkaaseen toteuttamiseen ja sen varmistamiseen. DoD-kriteeri määrittelee kriteerit, jotka on täytettävä ennen kuin ominaisuutta (Feature) tai tarinaa (Story) voidaan pitää valmiina. DoD-kriteerit sisältävät erilaisia kriteereitä, kuten automaattisten testien suorittamisen, dokumentaation päivittämisen ja koodin tarkastamisen. Kun tiimi käyttää DoD-kriteereitä niin samalla varmistetaan, että tuote on laadukas ja täyttää asiakkaan tarpeet. Jokainen ominaisuus (Feature) tai tarina (Story) tulee hyväksyä valmiiksi hyväksymiskriteerien mukaisesti juuri kyseisestä kokonaisuudesta vastuussa olevan roolin toimesta. Jokaisella SAFetiimin tasolla on omat hyväksyntäkriteerinsä, esimerkiksi tuotepäällikko (Product Manager) hyväksyy Featuren, kun taas puolestaan Story-tason hyväksymisestä vastaa tuoteomistaja (Product Owner). Nämä pienemmät kokonaisuudet yhdistyvät lopulta yhdeksi toimivaksi järjestelmäksi. Ennen tuotantoon vientiä on varmistettava, että dokumentaatio on valmis ja toteutus ei ole puutteellinen. Vaikka järjestelmässä olisikin joitakin virheitä tai toimimattomuuksia, niistä on oltava yhteenveto ja suunnitelma niiden korjaamiseksi ja testaamiseksi tulevaisuudessa. (Schwaber & Sutherland, 2017: Scaled Agile Inc., 2021.)

DoD-kriteerin käytöstä on tehty useita tutkimuksia, joissa tarkastellaan sen vaikutusta ohjelmiston sisäänrakennetun laadun parantamiseen ketterässä ohjelmistokehityksessä. Esimerkiksi Zulkarnain ja Ali (2017), käsittelivät artikkelissään tapaustutkimusta, jossa tarkastellaan DoD-kriteerien vaikutusta ohjelmiston laatuun ketterässä ohjelmistokehityksessä. Tutkimuksessa havaittiin, että DoD-kriteerien määrittely ja käyttö vaikuttavat merkittävästi ohjelmiston laatuun sekä DoD-kriteerien käyttämättömyys tai heikko määrittely johtaa usein laatuongelmiin. Lisäksi tutkimus ehdottaa, että organisaatiot kehittävät yhtenäisen määritelmän DoD-kriteeteille ja varmistavat sen käytön prosessissaan.

3.5 Laadunvarmistus

Laadunvarmistus ketterässä ohjelmistokehityksessä tarkoittaa jatkuvaa prosessia, jossa kehitys- ja laatutiimi varmistavat ohjelmiston laadun kaikissa kehitysvaiheissa. Tämä sisältää testaamista, arviointia ja jatkuvaa palautteen keräämistä. Laadunvarmistus on integroitava osaksi ketterän kehityksen prosessia, jotta se olisi jatkuvaa ja tehokasta. Tuotantoon viennin laatu (Release Quality) tarkoittaa varmistusta siitä, että kaikki halutut ominaisuudet on laadukkaasti toteutettu, testattu ja ne vastaavat vaatimuksia. Tämä onnistuu vain kehitys- ja laatutiimin välisellä yhteistyöllä. (Scaled Agile Inc, 2021.)

Yhteistyö kehitys- ja laatutiimien välillä ketterässä ohjelmistokehityksessä tarkoittaa läheistä yhteistyötä kehitys- ja laatutiimien välillä ohjelmiston laadun varmistamiseksi ja kehittämiseksi. Tämä yhteistyö on erittäin tärkeää, jotta voidaan varmistaa, että ohjelmisto on korkealaatuista ja täyttää asiakkaan tarpeet. Ketterässä ohjelmistokehityksessä kehitys- ja laatutiimit ovat tiiviisti yhteydessä toisiinsa läpi kehitysprosessin. Laatutiimi osallistuu kehitysprosessiin jo varhaisessa vaiheessa auttamalla määrittelemään testattavat toiminnallisuudet ja varmistamaan, että testausprosessit ovat tehokkaita. Kehitystiimi taas on vastuussa ohjelmiston toteutuksesta ja kehittämisestä. (Scaled Agile Inc, 2021.)

Yhteistyön tarkoituksena on varmistaa, että ohjelmisto on testattu perusteellisesti ja että mahdolliset ongelmat on korjattu ennen julkaisua. Laatutiimi antaa palautetta kehitystiimille testaustuloksista ja auttaa heitä ymmärtämään, miten heidän toimintansa vaikuttaa ohjelmiston laatuun. Tämä edistää tiedonkulkua ja oppimista sekä auttaa kehitystiimiä parantamaan kehitysprosessiaan ja välttämään samoja virheitä tulevaisuudessa. Yhteistyön merkitys korostuu erityisesti ketterissä menetelmissä, joissa kehitysprosessi on iteratiivinen ja nopea. Ketterissä menetelmissä kehitystiimi voi nopeasti tuottaa uusia toiminallisuuksia, ja laatutiimin on tärkeää varmistaa, että uusi toiminnallisuus on testattu perusteellisesti ja että se toimii odotetulla tavalla ennen sen julkaisua. (Scaled Agile Inc, 2021.)

Jo aiemminkin mainitun Mustafeen ja Taylorin (2016) tutkimuksen mukaan laadunvarmistuksen onnistunut toteuttaminen ketterässä ohjelmistokehityksessä vaatii tiimityötä, avointa kommunikaatiota, jatkuvaa palautetta ja iteratiivista kehitystä, joustavuutta, sopeutumiskykyä, automatisoitua testausta ja jatkuvaa integraatiota. Erityisesti he korostivat automatisoidun testauksen ja jatkuvan integraation merkitystä laadunvarmistuksen parantamisessa, joita käsiteltiinkin jo tarkemmin edellisessä alaluvussa. Lisäksi heidän mukaansa jatkuvan palautteen ja arvioinnin avulla varmistetaan, että ohjelmisto vastaa asiakkaan tarpeita ja kehitys etenee oikeaan suuntaan, kun taas joustavuus ja sopeutumiskyky mahdollistavat laadunvarmistuksen mukautumisen muuttuviin vaatimuksiin ja tarpeisiin.

3.6 Kehitystiimin itsearviointi ja jatkuvaparantaminen

Kehitystiimin itsearviointi ja jatkuva parantaminen ovat tärkeitä käytäntöjä ketterässä ohjelmistokehityksessä, jossa tavoitteena on jatkuva kehitys ja parantaminen tiimin työskentelyssä. Kehitystiimin itsearviointi tarkoittaa tiimin kykyä arvioida omaa suorituskykyään ja tunnistaa kehittymiskohteita. Itsearviointi auttaa tiimiä tunnistamaan vahvuudet ja heikkoudet sekä suunnittelemaan toimenpiteitä, jotka auttavat parantamaan työskentelyä ja tuottavuutta. Itsearvioinnin avulla tiimi voi myös tunnistaa riskejä ja mahdollisuuksia, jotka voivat vaikuttaa tiimin suorituskykyyn. (Scaled Agile Inc., 2021.)

Jatkuva parantaminen tarkoittaa puolestaan jatkuvaa pyrkimystä parantaa tiimin työskentelyä ja tuottavuutta. Jatkuva parantuminen voi näkyä esimerkiksi siten, että ajan myötä tiimi pystyy tekemään samassa ajassa yhä enemmän ominaisuuksia ja tarinoita kuin aiemmin ja näin tuottamaan enemmän arvoa asiakkaalle. Tiimi voi käyttää erilaisia menetelmiä ja työkaluja, kuten retrospektiivikokouksia, joiden avulla tiimi voi arvioida, miten he ovat edistyneet ja miten he voivat parantaa tulevaisuudessa. Jatkuvan parantamisen tavoitteena on varmistaa, että tiimi pystyy toimittamaan korkealaatuisia ohjelmistotuotteita nopeasti ja tehokkaasti. Retrospektiivien avulla voidaan löytää esimerkiksi pullonkauloja ja ehdottaa parannuksia kehitysprosessiin. Retrospektiivikokouksia

onkin käsitelty jo aiemmin luvussa 2.3.2 Scrum-viitekehykseen liittyen. (Scaled Agile Inc, 2021: Schwaber & Sutherland, 2020, s. 10.)

Lisäksi SAFe-viitekehyksen mukaan, vaikka tiimit pitävätkin säännöllisiä tiimitason retrospektiivejä jokaisessa sprintissä, niin lisäksi kaikkien ketterän kehitysjunan (Agile Release Train) tiimien tulisi osallistua yhteiseen inkrementin lopuksi järjestettävään tutki ja sopeudu -tapahtumaan (Inspect&Adapt) yhdessä kaikki sidosryhmien kanssa, jotta asiakkaan palautetta ja vaatimuksia saadaan käytettyä seuraavien kehitystöiden ohjaamiseen. Tämän avulla voidaan varmistaa toteutettujen ominaisuuksien ja toiminallisuuksien vastaavan asiakkaan tarpeita ja parantaa sisäänrakennettua laatua. (Scaled Agile Inc., 2021.)

Balan ym. (2019) tutkivat kehitystiimien itsearviointia Scrum-viitekehystä käyttävissä ohjelmistokehitysprojekteissa ja sen vaikutusta ohjelmiston laatuun. Tutkimuksen tarkoituksena oli tutkia, kuinka kehitystiimin itsearviointi vaikuttaa tiimin yhteistyöhön ja kommunikointiin sekä sitä kautta ohjelmiston laatuun. Tutkimus toteutettiin kahdessa eri Scrum-projektissa, joissa molemmissa oli neljä tiimiä. Yhdessä projektissa tiimit suorittivat itsearvioinnin jokaisen sprintin lopussa, kun taas toisessa projektissa itsearviointia ei suoritettu. Tutkimuksen aikana kerättiin dataa tiimien yhteistyöstä, kommunikoinnista ja ohjelmiston laadusta. Tutkimuksen tulokset osoittivat, että tiimien itsearviointi paransi merkittävästi tiimin yhteistyötä ja kommunikointia, mikä vaikutti positiivisesti ohjelmiston laatuun. Tiimit, jotka suorittivat itsearvioinnin, kommunikoivat enemmän ja avoimemmin keskenään, mikä auttoi heitä tunnistamaan ja ratkaisemaan ongelmia nopeammin. Lisäksi itsearvioinnin avulla tiimit pystyivät tunnistamaan ja parantamaan omia heikkouksiaan ja kehittämään tiimityötään pidemmällä aikavälillä.

3.7 Yhteenveto ketteristä menetelmistä ja käytännöistä sisäänrakennetun laadun kehittämiseksi

Yang ym. (2016, s. 157-184) tekivät yhteenvedon suosituimmista ketteristä käytännöistä ja löysivät yli 100 ketterää käytännettä, jotka he listasivat perustuen siihen, kuinka usein niihin on viitattu tutkituissa artikkeleissa. Kirjoittajien mukaan heidän yhteenvetonsa on ensimmäinen laatuaan tieteellisessä kirjallisuudessa, sillä vastaavaa kattavaa yleiskatsausta ei ole aikaisemmin tehty. On tärkeää huomioida, että nämä seuraavaksi listatut käytännöt ovat luokiteltu suosituimmiksi, koska ne ovat esiintyneet eniten kirjallisuudessa, mutta tämä ei välttämättä tarkoita, että ne ovat kaikkein käytetyimpiä ketteriä käytäntöjä.

1. Testivetoinen kehitys
2. Pariohjelmointi
3. Jatkuva integraatio
4. Päivittäispalaveri (Daily)
5. Refaktorointi
6. Läsnä oleva asiakas
7. Tuotteen kehitysjono (Backlog)
8. Koodin yhteisomistajuus
9. Retrospektiivit
10. Ohjelmointistandardit
11. Käyttäjätarinat (Story)
12. Suunnittelu
13. Sprintin suunnittelu (Sprint planning)
14. Iteratiivinen kehittäminen
15. Avoin työtila
16. Yksinkertainen suunnittelu
17. Yksikkötestaus
18. Sprintin katselmointi (Sprint review)
19. Järjestelmän vertauskuva
20. Pienet julkaisut

Kyseistä luetteloa, johon on koottu suosituimpia ketteriä käytänteitä, tarkastelemalla voidaankin huomata sen sisältävän suurimman osan aiemmissa luvuissa esitellyistä ja mainituista ketteristä käytänteistä, jotka liittyvät ketteriin periaatteisiin sekä Scrum- ja SAFe-viitekehyksiin. Tämän tutkimuksen tietoperustassa on mainittu tai käsitelty listalla esiintyvästä 20 käytänteestä 16:ta ketterää käytäntöä, jotka ovat osoittautuneet tehokkaiksi ohjelmistokehitysprojektien sisäänrakennetun laadun parantamisessa.

Ohjelmistokehityksen sisäänrakennettua laatua voidaan parantaa monilla erilaisilla menetelmillä tai käytännöillä, joita tässä kehittämistyössä on esitelty aiemmissa luvuissa. Organisaatiot ja ohjelmistokehitysprojektit voivatkin tavoitella sisäänrakennetun laadun parantamista toimimalla luvussa 2.1 esiteltyjen ketterien periaatteiden mukaisesti, jotka tarjoavat ohjeita siitä, miten ohjelmistokehitystä voidaan tehdä ketterästi. Nämä periaatteet jättävät kuitenkin käytännön toteutuksen organisaation vastuulle, jolloin organisaatio voi hyödyntää ketteriä periaatteita sille parhaiten soveltuvalla tavalla sisäänrakennetun laadun kehittämiseksi. Lisäksi organisaatiot ja ohjelmistokehitysprojektit voivat hyödyntää toiminnassaan ketteriä menetelmiä kuten esimerkiksi tässä kehittämistyössä luvuissa 2.3 ja 2.4 esiteltyjä Scrum- ja SAFe- viitekehyksiä. Nämä viitekehykset sisältävät monia hyviä toimintamalleja, rooleja, tapahtumia ja tuotoksia sisäänrakennetun laadun parantamiseksi, kuten esimerkiksi iteratiivinen ja inkrementaalinen kehittäminen, tuoteomistajan-rooli, tuotteen priorisoitu kehitysjono sekä iteraation katselmoinnit. Vaikka yritys ei haluaisi käyttää ketterää ohjelmistokehitystä tai sillä ei ole tarpeeksi osaamista käyttää esimerkiksi koko SAFe-viitekehystä, niin se voi silti ottaa käyttöön ketteriä käytäntöjä, joita se pitää tarpeellisina omassa toiminnassaan ohjelmiston sisäänrakennetun laadun kehittämiseksi. Näistä hyviä esimerkkejä ovat luvussa kolme läpikäydyt käytänteet ja periaatteet kuten pienet julkaisut, kanban-taulun käyttö visualisoimaan työnkulkua, jatkuva integrointi ja toimitus, automaattinen testaus, DoR- ja DoD-laatuporttien käyttö ennen seuraavia työvaiheita sekä kehitystiimin itsearviointi ja jatkuva parantaminen.

4 TUTKIMUKSEN TOTEUTUS

Tämä luku käsittelee tutkimuksen toteuttamisen vaiheita ja perustelee valittuja tutkimus- ja analyysimenetelmiä. Seuraavissa alaluvuissa tutkimuksen kohde esitellään ja lisäksi kerrotaan, miten tutkimusaineisto on kerätty, käsitelty ja analysoitu. Lisäksi kuvataan tutkimuksen toteutusvaiheet, jotta lukijalle muodostuu kokonaiskäsitys tämän tutkimusprosessin vaiheista.

4.1 Tutkimuksen kohde

Tutkimuksen kohteena olevassa ketterässä ohjelmistokehitysprojektissa käytetään SAFe-viitekehystä ja kehitystiimi puolestaan kehittää ohjelmistoa Scrum-viitekehyksen mukaisesti. Inkrementit kestävät projektissa 10 viikkoa ja jokaiseen inkrementtiin sisältyy viisi kaksi viikkoa kestävää kehitysjaksoa eli sprinttiä. Projektissa työskentelee kaksi erillistä Scrum-tiimiä, joilla on molemmilla oma tuoteomista vastaamassa story-tason kehittämisestä sekä niiden priorisoinnista. Haastateltavien henkilöiden lisäksi projektissa työskentelee useita määrittelijöitä ja testaajia.

Tutkimuksen aineisto kerättiin eräässä finanssialan yrityksen ketterässä ohjelmistokehitys projektissa työskenteleviä henkilöitä haastattelemalla. Tutkimukseen valittiin kahdeksan haastateltavaa henkilöä, jotka työskentelevät avainrooleissa tutkimuksen kohteena olevassa projektissa. Tähän päädyttiin sen takia, että Hirsjärven ja Hurmeen (2022, luku 3) mukaan empiirisessä tutkimuksessa on tärkeää saada eri näkökulmia tutkittavaan aiheeseen ja lisäksi sisäänrakennetun laadun kehittäminen ketterässä ohjelmistokehitysprojektissa on monitahoinen aihe, johon liittyy useita eri rooleja ja vastuita. Lisäksi Saaranen-Kauppinen & Puusniekka (2006) korostavat, että haastattelun kohteena olevat informantit on valittavat huolellisesti, koska tutkijan tulee arvioida tarkasti, keneltä saadaan mielekästä aineistoa tutkimusta varten. Huolellisen tutkimuksen suunnittelun perusteella tutkimuksessa päädyttiin haastattelemaan molempia tuoteomistajia, tuotepäällikköä, projektipäällikköä, testauspäällik-

köä, liiketoiminta omistajaa, ratkaisuarkkitehtia ja agile valmentajaa eli ketterien menetelmien ja periaatteiden valmentajaa, jotta saadaan mahdollisimman laajasti tietoa eri näkökulmista ja eri roolien vastuualueista.

SAFe-viitekehyksen mukaan tuoteomistaja on vastuussa tuotteen kehittämisestä ja priorisoinnista, joten hänellä on tärkeä rooli sisäänrakennetun laadun kehittämisessä. Tuotepäällikkö vastaa tuotteen suunnittelusta ja kehityksestä, ja hänen näkemyksensä auttavat ymmärtämään, miten sisäänrakennetun laadun kehittäminen voi olla osa tuotteen kehittämistä. Liiketoiminta omistajalla on puolestaan tärkeä rooli yrityksen strategian määrittämisessä ja hänen näkemyksensä auttavat ymmärtämään, miten sisäänrakennetun laadun kehittäminen voi tukea liiketoimintastrategiaa ja miten liiketoiminta kokee tuotteen laadun. (Scaled Agile Inc., 2021)

Projektipäällikkö johtaa kehitysprojektia ja on vastuussa projektin onnistumisesta. Hänen näkemyksensä auttavat ymmärtämään, miten sisäänrakennettu laatu vaikuttaa projektin aikatauluun ja projektin alussa määriteltyihin tavoitteisiin. Testauspäällikkö vastaa puolestaan laadunvarmistuksesta testauksesta ja varmistaa, että tuote täyttää sille asetetut vaatimukset. Hänen näkemyksensä auttavat ymmärtämään, miten sisäänrakennetun laadun kehittäminen voi auttaa varmistamaan tuotteen laadun. Haastattelemalla ratkaisuarkkitehtia, joka vastaa ohjelmiston arkkitehtuurista ja teknisestä suunnittelusta saadaan ymmärrystä, miten laadukkaasti ja hyviä kehityskäytäntöjä noudattaen kehitettävä ohjelmisto on toteutettu sekä miten ohjelmiston arkkitehtuuri on hänen näkemyksen mukaan vaikuttanut sisäänrakennettuun laatuun. Vaikka agile valmentaja ei sinänsä kuulu projektin tuote- tai kehitystiimiin, niin myöskin hänen haastattelemisensa on erittäin tärkeää sisäänrakennetun laadun kehittämistä koskevassa empiirisessä tutkimuksessa, koska agile valmentajalla on erityisosaamista ketterästä ohjelmistokehityksestä ja sen käytännöistä. Agile valmentaja auttaa kehitystiimiä ymmärtämään ja soveltamaan ketterän kehityksen periaatteita ja menetelmiä, ja hänellä on tärkeä rooli kehitysprosessin jatkuvassa parantamisessa. Hänen näkemyksensä auttavat ymmärtämään, miten sisäänrakennetun laadun kehittäminen sopii ketterään kehitysprosessiin ja miten näitä käytäntöjä voidaan parhaiten soveltaa. (Scaled Agile Inc., 2021)

Näiden eri rooleissa työskentelevien henkilöiden haastatteluista saadaan erilaisia näkökulmia siitä, miten sisäänrakennettua laatua voitaisiin kyseisessä ohjelmistokehitysprojektissa parantaa. Tämä auttaa tutkijaa ymmärtämään, miten sisäänrakennetun laadun kehittäminen voidaan parhaiten integroida ketterään ohjelmistokehitysprosessiin sekä auttaa vastaamaan tämän tutkimuksen tutkimuskysymykseen.

Hirsjärven ja Halmeen (2022, luku 3) mukaan haastattelun onnistumisen kannalta on suositeltavaa, että informantit eli haastateltavat voivat tutustua jo ennalta haastattelun teemoihin tai käsiteltävään aiheeseen. Kun haastateltaville lähetettiin kutsu haastatteluun, niin siihen liitettiin myös haastattelun teemat, jotta he pystyivät tutustumaan niihin etukäteen. Tutkimussuunnitelmassa (Liite 1) esiteltyjä apukysymyksiä ei kuitenkaan annettu haastateltaville ennakkoon, sillä tutkimuksen tekijä ei nähnyt sitä tarpeelliseksi, kun käsiteltävät teemat olivat tiedossa. Hirsjärven ja Hurmeen (2022, luku 3) mukaan ei ole olemassa oikeaa vastausta tutkittavan joukon riittävyyteen vaan se määräytyy aina kyseessä olevan yksilöllinen tutkimuksen mukaisesti. Kun kaikki kahdeksan haastatteluun valittua ja suostunutta henkilöä oli haastateltu, niin tutkija analysoi aineiston ja totesi aineiston määrän riittäväksi. Jos aineiston määrä ei olisi ollut riittävä, niin olisi ollut mahdollista toteuttaa vielä muutama lisähaastattelu tutkimusaineiston täydentämiseksi.

4.2 Tutkimusaineiston kerääminen ja haastattelujen kulku

Tässä tutkimuksessa aineistonkeruumenetelmänä käytettiin haastattelua. Haastattelu valikoitui aineistonkeruumenetelmäksi, sillä se on Hirsjärven ja Hurmeen (2022, luku 3.) mukaan aineistonkeruumenetelmänä melko yksinkertainen ja tehokas menetelmä syvällisemmän tiedon keräämiseen. Lisäksi Schultzen ja Avitalin (2011, s.1-16) mukaan haastattelu on paras aineistonkeruumenetelmä, kun halutaan ymmärtää tutkimuksen kohteena olevaa ilmiötä

kokonaisvaltaisesti ja kerätä haastateltavien näkemyksiä siihen liittyen. Haastattelussa on mahdollista suunnata kysymyksiä vastaajan vastausten perusteella ja päästä syvemmälle vastaajan ajatuksiin ja taustalla oleviin motiiveihin. Tämä johtuu siitä, että haastattelutilanne antaa mahdollisuuden vuorovaikutukseen ja keskusteluun, jossa tutkija voi esittää tarkentavia kysymyksiä ja saada lisätietoa vastaajan vastauksista.

Erilaisia haastatteluja käytetään usein tiedonkeruun menetelmänä tapaustutkimuksessa, koska tapaustutkimus liittyy tyypillisesti toimintaan eri tilanteissa, jolloin itse toimijat eli kehitettävän ilmiön asiantuntijat voivat kuvata ja selittää ilmiötä. Haastattelua pidetään yksinkertaisena tutkimusmenetelmä, jota voidaan käyttää esimerkiksi silloin, kun halutaan selvittää, mitä henkilö ajattelee tai miksi hän toimii tietynlaisella tavalla. Haastattelu on myös varsin joustava menetelmä, sillä siinä voidaan toistaa esitetty kysymys, oikoa väärinkäsityksiä tai selventää kysymyksen tarkoitusta. Haastattelukysymykset voidaan esittää siinä järjestyksessä, jossa tutkija katsoo ne aiheelliseksi, ja tarvittaessa voidaan myös esittää lisäkysymyksiä. (Ojasalo ym. 2020, s. 55.)

Haastattelu voi olla joko strukturoitu, jossa kaikille haastateltaville esitetään samat kysymykset, tai puolistrukturoitu, jossa haastattelija voi joustavasti soveltaa kysymyksiä ja syventyä aiheisiin tarkemmin. Tässä tutkimuksessa tutkimusaineisto kerättiin teemahaastattelulla, joka on puolistrukturoitu menetelmä. Teemahaastattelussa tutkija valitsee haastattelun aihepiirit etukäteen ja laatii kysymykset niiden ympärille. (Kananen 2008, s. 73-75; Hirsjärvi & Hurme 2002, luku 3.) Teemahaastattelu valittiin aineistonkeruumenetelmäksi, koska sen avulla on mahdollista saada syventää tutkijan tietoa tietystä aiheesta ja se voi tuoda aiheesta esiin uusia näkökulmia, joita on hankala löytää muilla tiedonkeruumenetelmillä.

Teemahaastattelu eli puolistrukturoitu haastattelu on aineistonkeruumenetelmänä lomakehaastattelun ja avoimen haastattelun välimuoto. Myersin ja Newmanin (2007, s. 2-26) mukaan tälläistä haastattelua voidaankin kuvata haastatteluna, jossa on suunnitelma haastattelun kulusta sekä mahdollisuus tutki-

jan improvisaation. Teemahaastattelussa kysymykset eivät ole tarkassa sanamuodossa, eikä kysymyksiä tarvitse esittää ennalta määrätyssä järjestyksessä. Tärkeintä tässä haastattelumallissa on se, että luonne on keskusteleva, mutta tutkijan kysymyksiin kuitenkin vastataan haastattelun aikana. Teemahaastattelu etenee tarkkaan suunniteltuja kysymyksiä väljemmin keskittyen ennalta tutkijan ennalta suunnittelemiin aihepiireihin. Tästä johtuen haastattelu etenee omalla painollaan haastattelijan ohjaten keskustelua pysymään ennalta määritellyissä teemoissa. Teemahaastattelussa tutkija käy keskustellen läpi haastateltavan kanssa erilaisia teemoja ja pyrkii varmistumaan, että kaikki ilmiön osa-alueet tulevat huomioiduksi. Haastattelutilanteessa teemoja voidaan tarkentaa kysymyksillä, jotka usein syntyvät vasta haastattelun aikana. Teemahaastattelu valittiin toteutustavaksi, sillä etukäteen suunnitellut teemat ohjaavat haastattelun kulkua, mutta jättävät tilaa myös vapaammalle keskustelulle. (Kananen 2008, s. 73-75; Hirsjärvi & Hurme 2022, luku 3.)

Teemahaastattelut voidaan suorittaa yksilö- tai ryhmähaastatteluina. Yksilöhaastattelussa haastattelun kohteena on yksi haastateltava, kun taas ryhmähaastattelussa haastateltavia on samaan aikaan useita. Yksilöhaastattelun etu verrattuna ryhmähaastatteluun on, että siinä voidaan kerätä jokaiselta haastateltavalta tietoa rauhallisessa ympäristössä ilman keskeytyksiä, jonka takia tässä tutkimuksessa käytettiin yksilöhaastattelua (Kananen 2008, s. 73-75; Hirsjärvi & Hurme 2022, luku 3). Lisäksi Saaranen-Kauppinen & Puusniekka (2006) korostavat, että teemahaastattelussa on erittäin tärkeää, että haastattelija tuntee aiheen hyvin ja on perehtynyt haastateltavien tilanteeseen, jotta voidaan käsitellä tarkasti asetettuja teemoja. Siksi haastattelu toteutettiin vasta sen jälkeen, kun tutkija oli tutustunut laajasti aiheeseen liittyvään kirjallisuuteen sekä aiempiin tutkimuksiin.

Empiirinen aineisto kerättiin yksilöhaastatteluina kohdeyrityksessä huhtikuussa 2023 puolistrukturoidulla teemahaastattelulla. Haastattelu toteutettiin online-haastatteluina käyttäen Microsoft Teams –sovellusta, joka mahdollisti häiriötekijöiden minimoimisen. Kaikki haastattelut nauhoitettiin dokumentointia ja myöhempää analysointia varten, ja niiden aikana kirjattiin muistiinpanoihin

keskeisiä huomioita. Haastatteluun varattiin riittävästi aikaa, jotta kaikki tarvittavat teemat olisi mahdollista käydä huolellisesti läpi ilman kiireen tunnetta. Haastattelujen kesto vaihteli 45-65 minuutin välillä

Tutkimuksen teemahaastattelurunko rakennettiin tutkimusongelman ja teoreettisen viitekehyksen ympärille. Jokaisessa haastattelussa käsiteltiin samoja teemoja, mutta kysymykset saattoivat hieman vaihdella haastateltavien välillä. Jos haastattelun aikana tuli ilmi jokin teema, joka oli jo käsitelty aikaisemmin, kyseistä teemaa ei enään käsitelty uudelleen. Teemahaastattelurunko on saatavilla opinnäytetyön liitteenä (Liite 1). Haastattelun teemoja olivat ketterät menetelmät ja SAFe-viitekehys, sisäänrakennettu laatu, jatkuva arvovirta, järjestelmän arkkitehtuuri ja koodin laatu, laadunvarmistus ja testaus sekä jatkuva parantaminen ja itsearviointi. Teemoja käsiteltiin ensin yleisellä tasolla ja sen jälkeen syvennyttiin niihin tarkemmin.

4.3 Aineiston analysointi

Hirsijärven ja Hurmeen (2022, luku 7) mukaan aineiston analysoinnin tarkoitus on parantaa sen sisältämän tiedon arvoa, jotta se saadaan muutettua selkeäksi ja yhtenäiseksi informaatioksi. Laadullisen aineiston käsittely perustuu loogiseen päättelyyn sekä aineiston tulkintaan, ja sitä tulee tehdä koko tutkimusprosessin ajan. Tutkimuksen validiteettia voidaan kasvattaa vertaamalla haastattelujen tuloksia aiempaan kirjallisuuteen. Tämä ei kuitenkaan tarkoita sitä, että teorian ja empiirisen tutkimuksen löydösten tulisi aina tukea toisiaan, koska se voi jopa estää tutkimuksen tekemisen. Laadullisissa tutkimuksissa voidaan käyttää teorialähtöistä eli abduktiivista tai aineistolähtöistä eli induktiivistä päättelyä, sekä näiden välimallia. Aineistolähtöinen päättely perustuu havaintoihin ja yksittäistapauksiin ja tutkimusaineistosta pyritään johtamaan yleisiä sääntöjä ja periaatteita. Teorialähtöinen päättely puolestaan perustuu ilmiöön liittyviin aiempiin tutkimuksiin. Tällöin tutkija käyttää olemassa olevaa teoriaa tai aiempaa tutkimusta lähtökohtanaan ja etsii tutkimusaineistosta näiden käsitteiden ja teemojen esiintymistä.

Tässä tutkimuksessa hyödynnettiin aineistolähtöistä sisällönanalyysiä, sillä se mahdollistaa suuren määrän tietoa sisältävän aineiston järjestelmällisen analyysin ja tulkinnan. Hirsjärven ja Hurmeen (2022, luku 7.) mukaan aineistolähtöinen sisällönanalyysi mahdollistaa aineiston luokittelun, jonka avulla on helpompi löytää tiettyjä teemoja ja aiheita, jotka esiintyvät toistuvasti aineistossa. Tähän ratkaisuun on päädytty sen takia, että näiden teemojen tunnistaminen auttaa tutkijaa ymmärtämään paremmin aineiston merkityksen ja näin vastaamaan varsinaiseen tutkimuskysymykseen. Lisäksi haastatteluissa esiin nousseiden ilmiöiden ja teemojen korrelaatiota on verrattu aiempiin tutkimuksiin, joita on käyty läpi aiemmissa luvuissa.

Kanasen (2008, s. 88-91) mukaan laadullisen tutkimuksen yleisimmät analyysimenetelmät ovat teemoittelu, tyypittely, sisällönerittely, diskurssianalyysi ja keskustelunanalyysi. Teemoittelussa keskitytään siihen, mitä kukin teema merkitsee aineistossa. Teemoittelun avulla voidaan pilkkoa ja ryhmitellä tutkimusaineisto teemojen mukaan, jotta voidaan verrata eri teemojen esiintymistä aineistossa. Teemoittelu on yksi sisällönanalyysin muoto ja yksi laadullisen tutkimuksen analyysimenetelmistä. Aineistonkeruumenetelmän ollessa teemahaastattelu, niin tässä tapauksessa myös aineiston teemoittelu on suhteellisen yksinkertaista. Teemoittelun avulla pyritään havaitsemaan tutkimusongelman kannalta olennaiset teemat kerätystä aineistosta. Teemoittelun onnistumiseksi on erittäin tärkeää, että teemat syntyvät aineiston pohjalta eivätkä ole ennalta määrättyjä haastattelun teemoja tutkijan toimesta. Tämän tutkimuksen analyysissä päätettiin käyttää teemoittelua, sillä se mahdollistaa kerätyn aineiston ryhmittelyn ja jaottelun keskeisiin teemoihin ja aihepiireihin, jotka olivat relevantteja tutkimusongelman kannalta sekä auttavat vastamaan tutkimuskysymyksiin sekä tuottamaan johtopäätöksiä teemoittain. Tässä tutkimuksessa aineiston teemoittelussa ei korosteta teemojen lukumäärää vaan tärkeintä on, että aineisto jaotellaan johdonmukaisesti ja systemaattisesti niiden teemojen mukaan, joita tutkimusaineistosta nousee esille. Tutkimusaineistosta havaittujen olennaisimpien teemojen avulla pyritään vastamaan tutkimuskysymyksiin sekä esittämään vastauksista johdettuja kehitysehdotuksia tutkimuksen kohteena olevan ohjelmistokehitysprojektiin.

Tämän tutkimuksen analysoitava aineisto muodostui kahdeksasta litteroidusta haastattelusta ja haastattelujen aikana tehdyistä muistiinpanoista. Aineiston litterointi suoritettiin Teams-ohjelmiston avulla samalla kun haastattelut nauhoitettiin. Aineiston analysointi alkoi jo haastatteluiden aikana, kun tutkija kiinnitti huomiota aineiston ja teorian yhtäläisyyksiin ja eroavaisuuksiin sekä teki niistä muistiinpanoja. Samalla kun aineistoa kuunneltiin uudelleen ja muistiinpanoja tarkennettiin haastattelun jälkeen, keskeisiä teemoja ja yhtäläisyyksiä nostettiin esiin ja niistä tehtiin muistiinpanoja. Näitä muistiinpanoja käytettiin myöhemmin hyödyksi varsinaisessa aineiston analysoinnissa.

Tässä tutkimuksessa tutkimusaineisto analysoitiin seuraavan Tuomen ja Sarajärven (2009, s.92) kuvaaman laadullisen tutkimuksen etenemisen mallin mukaisesti:

1. Päätä mikä aineostossa kiinnostaa
2. Aineiston läpikäynti ja kiinnostuksenkohteiden merkintä ja erottelu (aineiston litterointi ja koodaus)
3. Merkittyjen asioiden kerääminen yhteen ja erittely muusta aineistosta
4. Aineiston teemoittelu
5. Yhteenvedon kirjoittaminen

Tutkimusaineistosta eli litteroiduista haastatteluista sekä muistiinpanoista suodatettiin ensin tutkimuksen kannalta oleellinen aineisto. Tuomen ja Sarajärven mukaan (2009, s. 92) vaikka laadullisesta tutkimusaineistosta löytyy yleensä useita kiinnostavia ilmiöitä ja asioita, niin yksittäisessä tutkimuksessa ei ole mahdollista tutkia kaikkia näitä asioita kerrallaan. Siksi heidän mukaansa onkin erittäin tärkeää, että valitsee tarkkaan rajatun ilmiön, josta esitetään kaikki tutkimuksessa löydetty tieto ja se käydään läpi perusteellisesti.

Keskeisten teemojen tunnistaminen tapahtui valitun teorialähtöisen sisällönanalyysin keinoin. Tutkija käytti apunaan tietoperustassa esiteltyjä aiempia tutkimuksia ja pyrki tunnistamaan myös aiemmissa tutkimuksissa mainittu käsitteitä ja teemoja tutkimusaineistosta. Alustavat teemat muodostettiin aineis-

ton perusteella ja aiemmin suodatettu aineistoa lajiteltiin tunnistettujen teemojen alle. Tämän jälkeen teemoiteltu aineisto analysoitiin ja pyrittiin tunnistamaan samankaltaisuudet ja eroavaisuudet aiempiin tutkimuksiin liittyen. Tutkimuksen ja tutkimusaineiston analyysin tuloksia käsitellään luvussa viisi. Hirsjärven ja Hurmeen mukaan (2022, luku 7) tutkimus ei ole vielä kuitenkaan valmis silloin, kun tutkimusaineisto on vasta analysoitu, vaan tuloksia tulee myös tulkita ja selittää. Tuloksien tulkitsemisella ja selittämisellä tarkoitetaan, että tutkija pohtii tutkimusaineiston analyysin tuloksia ja tekee niistä johtopäätöksiä. Tämän tutkimuksen luvussa kuusi esitetään tutkijan analysoidusta tutkimusaineistosta tekemiä johtopäätöksiä.

5 TUTKIMUKSEN TULOKSET

Tässä luvussa esitellään tutkimuksen tulokset, jotka perustuvat kerättyyn aineistoon ja sen analyysiin. Haastattelun teemat oli suunniteltu huolellisesti ennen tutkimuksen toteutusta sekä haastattelu kysymykset oli laadittu vastaamaan tutkimusongelmaan mahdollisimman hyvin. Haastattelu antoikin hyvin vastauksia tutkimuskysymykseen sekä alatutkimuskysymykseen, jotka olivat seuraavat:

- Miten ketterän ohjelmistokehitysprojektin sisäänrakennettua laatua voidaan parantaa?
- Mitkä ovat keskeisimpiä ketteriä menetelmiä ja käytäntöjä sisäänrakennetun laadun parantamisen näkökulmasta?

Haastattelujen perusteella tässä tutkimuksen kohteena olevassa ketterässä ohjelmistokehityksessä sisäänrakennettu laatu koetaan erittäin tärkeäksi projektin onnistumiselle. Haastatelluiden henkilöiden vastauksissa korostui laadun olevan yhtä kuin asiakkaan kokema laatu, eli laatu syntyy siitä, miten asiakas kokee lopputuloksen. He olivat myös yhtä mieltä siitä että, varmistaakseen loppukäyttäjälle mahdollisimman hyvän käyttökokemuksen on erittäin tärkeää,

että laatu rakennetaan ohjelmistoon jokaisessa vaiheessa, kuten valmistelussa, määrittelyssä ja koodauksessa, kuten myös SAFe-viitekehyksessä (Scaled Agile Inc., 2021) on tavoitteena. Haastateltavien mielestä onnistuneen ohjelmistokehitys projektin perustana ovat laadukkaat toimintamallit ja prosessit, jotka kaikki ymmärtävät samalla tavalla heti projektin alusta lähtien. Ilman yhteisiä toimintamalleja ja prosesseja on mahdotonta rakentaa laadukas ohjelmisto. Haastateltavien vastausten mukaan tärkeimpinä tekijöinä ketterin menetelmin toteutettavan ohjelmistokehitysprojektin sisäänrakennettuun laatuun nähtiin seuraavat teemat:

1. SAFe-viitekehys ja ketterät periaatteet
2. Sujuva arvonvirtaus ja sen läpinäkyvyys
3. Laadunvarmistus ja testaus
4. Jatkuva parantaminen
5. Kommunikaatio

5.1 SAFe-viitekehys ja ketterät periaatteet

Haastateltavien vastausten perusteella voidaan todeta, että vaikka tutkimuksen kohteena olevaa ohjelmistokehitysprojektia toteutetaan ketterän SAFe-viitekehyksen mukaisesti, niin he eivät koe projektin kaikkien toimintatapojen ja käytäntöjen olevan ketterien periaatteiden mukaisia. Useimmat haastateltavat mainitsivat, että ketteryyteen pyrkiminen ja projektin toteuttaminen SAFe-viitekehyksen mukaisesti on hyödyllisiä, mutta viitekehyksen ydinarvojen käytännön soveltaminen ja aidon ketteryyden saavuttaminen on ollut haasteellista.
Kaikki haastateltavat olivat sitä mieltä, että käytössä oleva SAFe-viitekehys tuo selkeyttä projektin rooleihin ja selkeyttää kuka päättää mistäkin asiasta. Haastattelujen perusteella voidaankin todeta projektin käyttävän ketteriä käytäntöjä kuten sprinttejä, sprintin suunnitteluja ja demoja. Toisaalta haastateltavat myös totesivat, että nämä käytänteet eivät vielä tee itse projektista ketterää.

Haastateltavien vastauksissa korostui se, että jokaisessa projektissa tulisi ottaa huomioon projektin tärkeimmät resurssit eli ihmiset ja oikeat roolitukset

hyödyntäen heidän ammattitaitoaan. Haastateltavien mielestä projektissa käytössä olevat roolit ja niiden vastuut ovat hyvin tiedossa, mutta parempi kommunikaatio eri roolien välillä olisi tarpeellista. Lisäksi muutamat haastateltavat mainitsivat, että erilaisten palaverien tavoitteet tulisi määritellä tarkemmin, jotta projektipalaverien tavoitteet ja tarkoitus ovat selkeät kaikille projektissa työskenteleville henkilöille. Kaikilla haastatelluilla oli yhtenäinen näkemys siitä, että koko projektin laajuisesti järjestetään liikaa palavereja. Lisäksi isona haasteena koettiin, että jatkuvien palaverien välissä, ei ehdi keskittymään muuhun työhön. Moni haastateltavista nostikin esille, että oli tärkeää priorisoida ja sopia yhdessä mihin palavereihin kukin osallistuu. Tästä hyvänä esimerkkinä nostettiin, että pahimmillaan yhdessä palaverissa voi olla mukana tuotepäällikkö, tuoteomistaja, liiketoimintaomistaja ja määrittelijä, vaikka varsin hyvin riittäisi, kun yksi näistä rooleista olisi mukana. Luvussa 2.4.3 esitellyissä SAFe-viitekehyksen rooleissa onkin määritelty hyvin tarkasti kaikkien toimitusjunassa työskentelevien roolit ja niiden tarkoituksena onkin välttää juuri tätä päällekkäistä työtä roolien välillä (Scaled Agile Inc., 2021).

Seitsemän haastateltavan mielestä ketterät menetelmät ja vesiputousmalli ovat sekoittuneet kyseissä projektissa jonkinlaiseksi hybridiksi. Tähän nähtiin syynä esimerkiksi se, että projektissa on ollut haasteita toteuttaa ohjelmiston toiminallisuuksia ketterien periaatteiden mukaisesti – inkrementaalisesti ja iteroiden sekä vaaditun dokumentaation määrä on huomattavan suuri, ennen kuin itse toiminallisuuksien kehittäminen voidaan aloittaa. Nämä toimintatavat ovat ristiriidassa ketterän kehityksen periaatteiden kanssa, jotka ovat mainittuna luvussa 2.1. Tähän ovat päätyneet myös Misra ym. (2010, s. 451–474) sillä he mainitsevat tutkimuksessaan, että eräs keskeinen filosofinen ero ketterän kehityksen mallien ja perinteisten mallien välillä on, että ketterässä kehityksessä pyritään pois dokumentaatiokeskeisestä ohjelmistokehityksestä. Heidän mukaansa toimiva ohjelmisto tulisi olla etusijalla dokumentaation yli ja kehittämistä tulisi suorittaa iteroiden ja inkrementaalisesti, jotta mahdolliset epäselvyydet vaatimuksissa sekä virheet havaitaan mahdollisimman aikaisessa vaiheessa ja suuntaa on mahdollista korjata.

Yhteenvetona voidaankin todeta, että haastateltavien vastauksissa on sekä positiivisia että negatiivisia näkemyksiä SAFe-viitekehyksen ja ketterien käytäntöjen hyödyntämisestä projektissa. Useimpien informanttien mielestä SAFe-viitekehyksen roolit ja vastuut tuovat selkeyttä vastuisiin ja päätöksentekoon projektissa. Lisäksi projektissa käytössä olevan SAFe-viitekehyksen ja toimittajan käyttämän SCRUM-viitekehyksen tapahtumien kuten sprintin ja inkrementtien suunnittelujen, demojen ja retrospektiivien koetaan olevan hyödyllisiä, kun niitä hyödynnetään oikein. Puolet haastateltavista korosti myös projektin alkuvaiheessa ollutta resurssipulaa ja näkivät sen mahdollisena syynä siihen, että aidon ketteryyden saavuttaminen on ollut projektissa hankalaa, kun asioita on jouduttu tekemään ”kädestä suuhun”. Haastateltavien vastauksissa nousi myös esille, että he kokevat toimittajien kanssa tehtyjen sopimusten hankaloittavan ketterien toimintamalleja, sillä sopimuksissa ei ole heidän mielestään otettu tarpeeksi huomioon SAFe-viitekehyksen ydinarvoja.

5.2 Sujuva ja jatkuva arvonvirtaus

Kaikki haastateltavat ovat kokeneet sujuvan ja jatkuvan arvovirran tuottamisen onnistuneen ketterässä ohjelmistokehitysprojektissa vaihtelevasti. Useimmat ongelmat liittyvät kireään aikatauluun, vähäisiin resursseihin sekä liiketoiminnallisten vaatimusten ja UI-designien valmistumiseen myöhäisessä vaiheessa ennen suunnittelupalavereja tai välillä jopa suunnittelupalaverien jälkeen. Useimmat haastateltavista olivat sitä mieltä, että toteutukseen valmiita määriteltyjä toiminallisuuksia tulisi olla vähintään kahdeksi seuraavaksi inkrementiksi eli 10 viikon kehitysjaksoksi. Heidän mukaan tässä projektissa näitä toteutukseen valmiita toiminallisuuksia on ollut hädin tuskin juuri alkavaan inkrementtiin.

Kuusi informanttia oli sitä mieltä, että inkrementtien suunnittelut toimittajan puolelta tehdään hätäisesti ja otetaan töitä työn alle, vaikka niissä on vielä avoimia asioita, kuten esimerkiksi työmääräarviot ja tekniset ratkaisut ennekuin niitä voidaan vaatimusten mukaisesti toteuttaa. Lisäksi monet haastatel-

vista kokivat, että inkrementtien suunnittelupalavereissa tulisi ratkaista paremmin muun muassa riippuvuudet muihin hankkeisiin ja projektin sisäiset riippuvuudet eri toimittajien välillä. Tästä hyvänä esimerkkinä seuraava kommentti:

> ”Suunnittelupalavereissa tulisi olla mukana kaikkien projektin avainroolien lisäksi myös tarpeellisesti henkilöt toimittajilta sekä sisäisistä hankkeista, joiden kanssa projektilla on riippuvaisuuksia.”

Suurin osa haastateltavista koki työmääräarvioiden olevan ennemminkin arvauksia kuin huolella tehty arvioita ja tämä on heidän mielestään johtanut siihen, että kesken toteutuksen huomataankin, että tämä kokonaisuus vaatii enemmän työtä ja sitä ei saadakaan aikataulussa valmiiksi. Tästä hyvänä esimerkkinä toimii seuraavat kommentti:

> ”Työmääräarviot eivät olleet luotettavia, ja spill overia tuli paljon eli suunniteltiin enemmän kuin saatiin tehtyä.”

Toisaalta taas moni informantti oli sitä mieltä, että vaikka työmääräarviot eivät ole olleet kovin tarkkoja, niin projektin edetessä featureja on onnistuttu pilkkomaan järkevämmiksi kokonaisuuksiksi kuin projektin alkuvaiheessa. Tämä on mahdollistanut sen, että niitä saadaan toteutettua yhdessä inkrementissa johon SAFe-viitekehyksen mukaan tulisi pyrkiäkin.

Lähes kaikki informantit totesivat, että päällekkäisen työn määrää ei ole osattu rajoittaa oikealla tavalla vaan on tehty hyvin paljon erilaisia toiminallisuuksia päällekkäin samassa inkrementissa. Tämä on johtanut siihen, että featureita ei ole saatu valmiiksi suunnitellussa aikataulussa. Monet haastateltavat kokivat tämän erityisen haastavaksi, sillä sen koettiin hankaloittavan jatkuvan arvovirran tuottamista. Esimerkiksi yhden informantin mielestä tämä hankaloittaa testaamista, sillä testattavia featureita ei tule testaukseen tasaisella tahdilla vaan hyvin epäsäännöllisesti. Haastatteluissa nousi esille tähän mahdollisena ratkaisuna päällekkäisen työn määrän rajoittaminen, jotta voidaan keskittyä saattamaan kesken olevat työt valmiiksi ennen uusien töiden aloittamista.

Tämä voitaisiin toteuttaa esimerkiksi siten, että tehdään maksimissaan vain kuutta featurea tai storya yhtä aikaa ja seuraava voidaan aloittaa vasta kun edellinen näistä on valmistunut. Nämä haastatteluissa esille nousseet asiat korreloivat hyvin Reinhertsenin (2009, s. 143-166) esittämien tulosten kanssa, sillä hänen mukaansa päällekkäisen työn rajoittaminen auttaa parantamaan kehitystiimin tehokkuutta ja vähentämään keskeneräisten työtehtävien aiheuttamaa kaaosta. Toisaalta hän myös muistuttaa, että liian pieni päällekkäisen työn määrä voi johtaa resurssien alikäyttöön.

Useampi haastateltava toi myös esille, että tekniset riippuvuudet eivät ole aina olleet selvillä ennen toteutusta, mikä on osoittautunut ongelmaksi myöhemmin. Tämä on johtanut siihen, että suunniteltuja töitä ei ole pystytty toteuttamaan sprinttien ja inkrementtien aikana, kun on jo aloitettu toteuttamaan toiminallisuuksia joiden DoR-laatuportin (Definition of Ready) tekniset kriteerit eivät ole olleet valmiina. Toisaalta viisi informanttia, nosti esille, että DoR-laatuporttien käyttö on parantunut projektin edetessä, kun siihen on kiinnitetty aiempaa enemmän huomiota ja se onkin tällä hetkellä hyvällä tasolla. Kaksi informanttia nostivat kuitenkin esille olisiko DoR-laatuportin kriteereitä järkevä käydä yhdessä läpi, jotta kaikki ymmärtäisivät ne samalla tavalla. Tästä hyvänä esimerkkinä toimii kommentti:

> ”Esimerkiksi DoR-laatuportti ”alustavat käyttäjätarinat on tunnistettu” ymmärretään usein, että käyttäjätarinat tulee olla jäädytetty ja muutoksia siihen ei enää sallita, vaikka tätä se ei suinkaan tarkoita.”

Haastattelun tulosten perusteella voidaan nähdä, että teknisten riippuvuuksien ja DoR-laatuporttien käytön vaikutus projektin onnistumiseen on merkittävä. Useampi haastateltava mainitsi, että teknisten riippuvuuksien selvittäminen ennen toteutusta on ollut haasteellista ja johtanut suunniteltujen töiden toteutuksen viivästymiseen. Tässä kohtaa Wijaya & Raharjon (2018) tutkimus on relevantti, sillä se osoittaa, että DoR-kriteerien yhtenäinen määrittely ja niiden oikeanlainen käyttö auttavat parantamaan tiimin tuottavuutta ja vähentämään hukkaa. Kolme informanttia esitti, että DoR-laatuportit tulisi käydä läpi yhdessä, jotta kaikki ymmärtäisivät ne samalla tavalla, mikä olisi myös hyvä käy-

täntö Wijayan ja Raharajon (2018) mukaan. Organisaatioiden tulisi siis panostaa DoR-kriteerien yhtenäiseen määrittelyyn ja varmistaa niiden oikea käyttö ja hyväksyntä ohjelmistokehitysprojekteissaan, jotta voidaan välttää viivästymisiä ja varmistaa projektin onnistuminen.

Yhteenvetona voidaan todeta, että jatkuvan arvovirran tuottamisessa on ollut haasteita ketterän ohjelmistokehitysprojektin eri vaiheissa. Suurimpana syynä tähän nähtiin se, että projektissa on koettu selkeästi pulaa resursseista, jonka nähdään vaikuttaneen negatiivisesti sisäänrakennettuun laatuun. Useat informantit mainitsivat ongelmia, jotka johtuvat riittämättömästä suunnittelusta ja kommunikaatiosta, sekä liiallisesta työmäärästä ja epäselvistä riippuvuuksista eri toimittajien välillä. Myös DoR-kriteerien puutteellinen noudattaminen mainittiin useaan kertaan ongelmana, joka on johtanut heikentyneeseen laatuun projektin alkuvaiheessa. Toisaalta myös tämän nähtiin parantuneen projektin edetessä. Lisäksi työmääräarviot ovat osoittautuneet hieman epäluotettaviksi, mikä on johtanut suunnitellun työmäärän ylittymiseen.

5.3 Laadunvarmistus ja testaus

Projektissa havaitut laatuongelmat tulivat esille jokaisessa haastattelussa, vaikka vaadittu laatu on määritelty toimittajasopimuksissa. Useat haastateltavat nostivat esille, että näkevät suurimpana syynä projektin aikataulun myöhästymiselle koodin heikon laadun. Erityisesti huolissaan oltiin miten koodin havaittu heikko laatu ja virheiden suuri määrä voi näkyä loppukäyttäjälle esimerkiksi epävakautena ja ohjelmiston hitautena. Kolme informanteista nosti esille, että vaikka vaadittu laatu on määritelty sopimuksissa, niin projektissa työskentelevillä henkilöillä ei ole yhteneväistä käsitys vaaditusta laadusta. Heidän mukaansa tämä johtaa siihen, että ei pystytä rakentamaan ohjelmistoa joka vastaa näitä laatuvaatimuksia. Haastateltavat toivatkin tähän liittyen esille, että varsinkin kun laatuongelmia on havaittu, niin olisi tärkeä puuttua koodin laatuongelmiin käyttämällä esimerkiksi koodikatselmuksia ongelmien mahdollisimman aikaiseksi havaitsemiseksi.

Myös aiempien tutkimustulosten perusteella projektissa havaitut laatuongelmat ovat yleisiä ohjelmistokehityksessä ja ne voivat johtaa aikataulun viivästymiseen ja heikentyneeseen lopputuotteen laatuun. Esimerkiksi Nagappan ym. (2006) mukaan koodin tarkastukset ovat erittäin tehokas keino parantaa ohjelmiston laatua ja vähentää virheitä jo kehitysprosessin aikana. Toisaalta Rigsby ym. (2014) kuitenkin nostavat esille että, koodin tarkastusten vaikutus voi vaihdella ja niiden käyttö ei aina johda automaattisesti parempaan sisäänrakennettuun laatuun vaan tämä riippuu muun muassa tarkastusten laadusta, kehittäjien kokemuksesta ja tarkastusten ajankohdasta.

Suurin osa haastatelluista toi esille, että projektin aikana on ollut epäselvyyksiä, milloin storyt ovat valmiita esiteltäväksi ja näin hyväksyttäviksi odottamaan feature-tason hyväksymistestausta. Useiden haastateltavien mielestä storyja ei pitäisi tuoda esiteltäviksi ja hyväksyttäviksi, jos niissä on edelleen auki kriittisiä tai korkean tason bugeja. Tämä on johtanut muun muassa siihen, että näitä kriittisiä ja korkean tason bugeja on auki vielä hyväksymistestauksessa, jonka koettiin vaikeuttavan hyväksymistestausta ja aiheuttavan paljon selvittelyä. Tämän arveltiin johtuneen siitä, että ei ole sovittu yhteisiä vaatimuksia sille, milloin esimerkiksi story on valmis ja voidaan siirtää odottamaan muiden samaan toiminnallisuuteen liittyen storyjen valmistumista. Lisäksi haastatteluissa nousi esille että, DoD-laatuportin (Definiton of Done) kriteerien ei koettu olevan määritelty tarpeeksi hyvin tai ainakaan käyty yhdessä läpi, jotta kaikilla olisi niistä sama ymmärrys. Haastatteluissa ilmeni, että kun tähän DoD-laatuportin käyttöön on kiinnitetty enemmän huomiota ja niin se on vähentänyt näitä epäselvyyksiä projektin edetessä. Osa haastateltavista kuitenkin koki, että nämä kriteerit ja hyväksymisen edellytykset olisi hyvä käydä yhdessä läpi, jotta kaikki ovat samalla sivulla niiden suhteen. Tästä esimerkkinä seuraavat kommentti:

> ”Vaikka itselleni DoD-kriteerit tuntuvat selkeiltä, niin on hyvin mahdollista, että niistä ei ole yhteistä ymmärrystä tuotetiimin ja toteutustiimin suhteen.”

> ”DoD:ien käytössä pitäisi olla tiukempi ja high- ja critical- bugit tulisi korjata ennen hyväksymistä”

Nämä haastatteluissa ilmenneet yhteisten käytäntöjen puutteet sekä käytännön ongelmat tukevat aiempia tutkimuksia DoD-kriteerien käytön merkityksestä ja sen vaikutuksesta ohjelmiston sisäänrakennettuun laatuun. Esimerkiksi Zulkarnain ja Ali (2017) päätyivät siihen, että DoD-kriteerien käyttämättömyys tai niiden huono määrittely johtaa usein ongelmiin ohjelmiston laadussa. He myös suosittelivat DoD-kriteerien yhteistä määrittelyä näiden laatuongelmien vähentämiseksi.

Useissa haastatteluissa nousi myös esille, että integraatiotestauksen ja hyväksymistestauksen työnjako ei ole täysin selvillä. Lähes kaikki haastateltavista nosti esille, että hyväksymistestauksessa löydetään varsin paljon virheitä, jotka olisi pitänyt löytyä jo aiemmin integraatiotestauksessa. Tämän on havaittu aiheuttaneen paljon lisätyötä hyväksymistestaajille. Toisaalta taas ketterän metriikan mittari UAT Defect Leakage, joka indikoi, kuinka suuri osa bugeista löydetään vasta hyväksymistestausvaiheessa verrattuna aiempiin vaiheisiin että, projektissa löydetään yli 75% kriittisen ja korkean tason bugeista ennen hyväksymistestausta. Tähän haastateltavat kokivat mahdollisesti syynä olevan sen, että bugeja löytyy paljon myös hyväksymistestauksessa, vaikkakin niitä havaitaan paljon myös aiemmin. Lisäksi lähes kaikki haastateltavat nostivat esille hyväksymistestaus-tiimin suoriutuneen erittäin hyvin ja olevan erittäin tärkeässä roolissa, jotta rakennettava ohjelmisto näkyy loppukäyttäjille laadukkaana kokonaisuutena. Tästä esimerkkinä seuraavat haastateltavien kommentit:

> ”UAT-tiimi on suoriutunut todella hyvin ja testannut myös paljon asioita, jotka tulisi testata jo aiemmin.”

> ”Olen UAT-testauksen laatuun todella tyytyväinen ja yhteistyö heidän kanssaan sujunut todella hyvin, vaikka siellä on ollut vaihtuvuutta.”

5.4 Jatkuva parantaminen

Jatkuva parantaminen on yksi ketterän ohjelmistokehityksen periaatteista. Sillä tarkoitetaan jatkuvaa pyrkimystä parantaa tiimin työskentelyä ja tuottavuutta. Tähän voidaan Scaled Agile Incorporationin (2021) mukaan käyttää

erilaisia menetelmiä kuten retrospektiivejä. Niiden tarkoituksena on arvioida sitä, miten tiimi on suoriutunut ja mitä asioita tulisi parantaa. Kaikkien kahdeksan haastateltavien vastauksista käy ilmi, että jatkuva parantaminen ei ole toteutunut kovinkaan hyvin kyseisessä ketterässä ohjelmistokehitysprojektissa. Toisaalta kaikki haastateltavat pitivät jatkuvaa parantumista erittäin tärkeänä, jotta ohjelmiston sisäänrakennettu laatu myöskin paranisi. Haastatteluissa kävi ilmi, että retrospektiiveita on pidetty projektin aluksi, mutta ne ovat jääneet kokonaan pois käytöstä, sillä niiden hyödyllisyys oli koettu heikoksi. Tästä kertovatkin seuraavat kommentit:

> ”Retroja järjestettiin vain koska ne ovat käytössä Scrum:issa, mutta niissä ei mietitty konkreettisia asioita, joita voitaisiin jatkossa parantaa”

> ”Retroissa tulisi keskittyä asioihin joiden nähdään onnistuneen huonosti ja miettiä miten niitä voitaisiin kehittää eikä vain todeta kaiken olevan hyvin, jos näin ei ole”

Esille nousi myös mielipiteitä, että toteutustiimi ei paranna itse toimintatapojaan vaan ainoastaan parantaa asiakkaan esille nostamia epäkohtia. Tämän hän näki johtavan siihen, että esille nostetut ongelmat kyllä korjataan, mutta niiden juurisyitä ei. Tämän nähtiin olevan jatkuvan parantamisen periaatteen vastaista, ja vaikuttavan myös negatiivisesti sisäänrakennettuun laatuun. Toisaalta suurin osa haastateltavista oli sitä mieltä, että myös tiimin tekeminen on kehittynyt projektin aikana, ja tämä näkyy esimerkiksi parempana kommunikaationa sekä paremmin valmisteltuina demo-tilaisuuksina, vaikka retrospektiiveita ei ole pidettykään.

Haastatteluissa kävi myös selväksi, että projektissa ei haastateltavien mielestä hyödynnetä erilaisia metriikoita kovinkaan hyvin. Tämä on näkynyt esimerkiksi siten, että iteraatiohin suunnitellaan jatkuvasti liikaa storyja, vaikka suunniteltuja töitä ei saada valmiiksi. Lähes kaikki haastateltavat näkivät erittäin tärkeänä näiden metriikoiden paremman hyödyntämisen ja päätösten tekemisen metriikoiden tarjoaman datan pohjalta, jotta sisäänrakennettua laatua voidaan parantaa. Tästä hyvänä esimerkkinä toimii seuraava kommentti:

> ”Jos omaa tekemistä oikeasti reflektoitaisiin, niin tiimin tulisi suunnitella vähemmän töitä seuraaviin iteraation, kun edellisissä iteraatiossa ei ole saatu suunniteltuja töitä valmiiksi.”

Nämä tutkimuksessa esiin tulleet asiat liittyen retrospektiivien hyödyntämiseen korreloivat hyvin aiempiin tutkimuksiin jatkuvan parantamisen merkityksestä ohjelmistokehitysprojektin sisäänrakennettuun laatuun. Esimerkiksi jo aiemmin luvussa 3.6 esitelty Balanin ym. (2019) tutkimus retrospektiivien hyödyllisyydestä osoitti, että tiimien itsearviointi paransi merkittävästi tiimin yhteistyötä ja kommunikointia, mikä vaikutti positiivisesti ohjelmiston laatuun. Tutkimuksen mukaan tiimit, jotka suorittivat itsearvioinnin ja käyttivät retrospektiiveita, kommunikoivat enemmän ja avoimemmin keskenään, mikä auttoi heitä tunnistamaan ja ratkaisemaan ongelmia nopeammin. Lisäksi itsearvioinnin avulla tiimit pystyivät tunnistamaan ja parantamaan omia heikkouksiaan ja parantamaan toimintaansa jatkuvan parantamisen periaatteen mukaisesti.

5.5 Kommunikaatio

Ketterä ohjelmistokehitys perustuu tiiviiseen yhteistyöhön tiimin jäsenten välillä sekä toisaalta myös jatkuvaan kommunikointiin asiakkaan kanssa. Tällainen kommunikointi mahdollistaa nopean reagoinnin muutoksiin ja tarpeisiin, sekä vähentää virheiden syntymistä ja parantaa projektin laatua (Scaled Agile Inc, 2021). Kaikkien haastateltavien mielestä tehokas ja avoin kommunikointi tilaajan ja toimittajan välillä on erittäin kriittinen asia ohjelmiston laadukkaan toteutuksen kannalta. He toivat myös esille, että jos halutaan saada hyötyjä iteratiivisesta ja inkrementaalisesta kehittämisestä, niin tulee varmistaa, että kommunikaatio on aktiivista. Tällä aktiivisella kommunikaatiolla tarkoitettiin, että toteutustiimin tulee ymmärtää mitä asiakas haluaa ja näin he voivat rakentaa ohjelmiston asiakkaan vaatimusten ja tarpeiden mukaisesti.

Haastateltavien mukaan projektissa on pyritty korostamaan mahdollisimman välitöntä ja matalan kynnyksen kommunikaatiota. Toisaalta kommunikaatiotapoihin liittyen haastateltavien vastaukset hieman jakautuivat. Osan mielestä

kommunikaatio toteutuu parhaiten kasvotusten, osan mielestä puolestaan digitaaliset kommunikaatiovälineet, kuten Teams ja Jira toimivat lähes yhtä hyvin kuin kasvotusten tapahtuva viestintä. Kaksi haastateltavaa korosti, että psykologinen turvallisuus ja tiimin yhteinen ymmärrys siitä, että välitön kommunikointi on tärkeää, jotta kaikki uskaltavat tuoda omat näkemyksensä esille ja kysyä asioista, jos kokevat niiden oleva epäselviä. Samaan teemaan liittyen yksi haastateltavista nosti esiin myös näkökulman, että niin sanottujen pienien ja nopeiden kysymysten esittäminen digitaalisesti on mahdollisesti isomman kynnyksen takana kuin kasvotusten kommunikoidessa.

> ”Ei uskalleta kysyä ja varmistaa matalalla kynnyksellä, miten tämän halutaan toimivan ja siksi vaatimuksia ei välttämättä täysin ymmärretä.”

Tässä ohjelmistokehitysprojektissa toimittaja on Offshore-toimittaja, joka tarkoittaa, että toimittajayritys sijaitsee eri mantereella kuin asiakas. Kuusi informanttia koki, että projektissa käytössä oleva Offshore-malli vaikeuttaa kommunikaatiota ja näin vaikuttaa negatiivisesti ohjelmiston sisäänrakennettuun laatuun. Informantit toivat esille, että tähän voisi mahdollisesti olla syynä kulttuurilliset erot toimintatavoissa. Informanttien kommentit toivat hyvin esille, että kommunikaation laadun koetaan olevan selvästi parempi sellaisten henkilöiden kanssa, jotka ovat saapuneet fyysisesti Suomeen ja heidän kanssaan on rakentunut sitä kautta parempi luottamussuhde:

> ”Suomeen tulleiden kanssa yhteistyö sujuu paremmin, kun on tavattu kasvokkain, niin on varmaankin pienempi kynnys kysyä asioista”

> ”Ollaan tutustuttu myös henkilökohtaisesti, niin tämän on parantanut luottamusta ja yhteistyötä molempiin suuntiin”

Toisaalta kaksi informanttia ei näe Offshore-mallia automaattisesti huonona asiana. Heistä toinen oli sitä mieltä, että riittää kun toimittajan avainhenkilöt kuten määrittelijät, ratkaisuarkkitehdit ja projektipäällikkö ovat Suomessa. Toinen taas puolestaan oli sitä mieltä, että digitaalisten palveluiden, kuten ohjelmistojen kehityksessä, Offshore-malli ei ole ongelma, sillä koronan aiheuttaman etätyöboomi on parantanut digitaalista kommunikointia viime vuosina.

Hänen mielestään tärkeämpi kriteeri toimittajan valinnalle tulisi olla mahdollisimman korkea laatu, eikä se onko toimittaja Offshore- vai perinteinen Onsite-toimittaja.

Nämä haastattelussa esiin nousseet tulokset liittyen kommunikaatioon ovat osin ristiriidassa Misran ym. (2009, s.1869–1890) tutkimuksen tulosten kanssa. Heidän tutkimuksen mukaan kommunikaatio ei ole yksi kriittisimpiä ketterän ohjelmistokehityksen menestystekijöitä. Samaan lopputulokseen on päätynyt myös Stankovic ym. (2013, s. 1663–1678) sillä heidän tutkimuksensa mukaan keskusteleva kulttuuri, jossa arvostetaan kasvokkain tapahtuvaa kommunikaatiota ei ole kovinkaan merkittävä tekijä ketterän ohjelmistokehitys projektin onnistumiselle.

6 JOHTOPÄÄTÖKSET JA POHDINTA

Tässä luvussa esitetään luvussa viisi tutkimuksen tulokset esiteltyjen tulosten perusteella muodostuneita johtopäätöksiä sisäänrakennetun laadun kehittämiseen liittyvistä ilmiöistä. Lisäksi niiden perusteella esitetään tutkimuksen tavoitteena olleita kehitysehdotuksia, joita tutkimuksen kohteena oleva projekti voi hyödyntää sisäänrakennetun laadun parantamiseksi. Lisäksi alaluvussa 6.2 arvioidaan tutkimuksen luotettavuutta ja vaikuttavuutta sekä alaluvussa 6.3 pohditaan tutkimusprosessia sekä esitetään mahdollisia aiheita jatkotutkimukselle.

6.1 Tulosten tarkastelu ja johtopäätökset

Tutkimuksessa tarkasteltiin miten projektissa eri rooleissa työskentelevät henkilöt näkevät sisäänrakennetun laadun sekä millaisin ketteriin menetelmiin ja erityisesti SAFe-viitekehyksen liittyvin menetelmin ja käytännöin sisäänrakennettua laatua voidaan kehittää. Seuraavaksi tarkastellaan ilmiöitä, joiden tutkimusten tulosten perusteella voidaan todeta vaikuttavan sisäänrakennettuun

laatuun sekä esitetään tutkijan kehittämisehdotuksia näihin ilmiöihin liittyen. Läpikäynti aloitetaan yleisesti SAFe-viitekehykseen liittyvistä ilmiöistä, jonka jälkeen käydään läpi kommunikaatioon sekä jatkuvaan parantamiseen liittyviä tuloksia. Lopuksi käydään läpi ilmiöt, jotka nähdään vaikuttavan SAFe-viitekehyksen sisäänrakennettuun laatuun jo aiemmin luvussa 3.1 läpikäytyjen viiden elementin kautta: jatkuva kehityksen sykli, arkkitehtuuri ja suunnittelun laatu, koodin laatu, järjestelmän laatu sekä julkaisujen laatu (Scaled Agile Inc. 2021).

Tämän tutkimuksen tarkoituksena oli selvittää, miten ketterän ohjelmistokehitysprojektin sisäänrakennettua laatua voidaan parantaa. Haastattelujen perusteella olisi ehdottoman tärkeää, että SAFe-viitekehyksen roolit ja periaatteet käydään läpi tarpeeksi tarkalla tasolla jo ennen projektin aloittamista, jotta kaikilla on sama käsitys siitä, miten viitekehystä hyödynnytetään kyseisessä ohjelmistokehitysprojektissa. SAFe-viitekehyksen osoittamat roolit ja tapahtumat koettiin hyväksi selkänojaksi toimia ketterästi, mutta tutkimuksen tulosten perusteella on erittäin tärkeää varmistaa, että kaikilla on yhteinen ymmärrys eri roolien vastuista sekä siitä kuka päättää ja vastaa mistäkin asioista. Tällä voidaan esimerkiksi vaikuttaa haastatteluissa esiin nousseeseen ongelmaan, että erilaisia palavereja koetaan olevan liian paljon ja niiden vievän paljon aikaa muulta varsinaiselta työltä. Tämän tutkimuksen mukaan olisikin ehdottaman tärkeä varmistaa, että ihmiset eivät ole mukana sellaisissa palavereissa, joissa heitä ei välttämättä tarvita. Toisaalta tulee myös varmistaa, että tarvittavat henkilöt ovat mukana niissä palavereissa joissa heidän on ehdottomasti oltava. Tutkimustulosten perusteella suosittelenkin käymään läpi projektin palaverikäytännöt yhdessä sekä määrittelemään selkeät tavoitteet ja osallistujat eri palavereille kuten suunnittelupalavereille, erilaisille läpikäynneille sekä demoille.

Kommunikoinnin nähtiin olevan erittäin kriittisessä roolissa tutkimuksen kohteena olevan ohjelmistokehitysprojektin onnistumisen kannalta, sillä kyseisessä projektissa toimittajana on Offshore-toimittaja. Haastatteluissa selvisi, että haastateltavien mielestä toimittajalla ja sen työntekijöillä ei välttämättä ole kokonaisvaltaista ymmärrystä Suomen finanssitoimialasta ja sitä koskevasta tiukasta regulaatiosta. Lisäksi kaikki kommunikointi tapahtuu englanniksi, joka

ei ole toimittajan tai asiakkaan työntekijöiden äidinkieli. Tämän lisäksi kulttuurillisten erojen nähtiin tuovan haasteita kommunikaatioon. Haastatteluissa nousikin esiin, että kommunikaation koetaan olevan parempaa niiden henkilöiden kanssa ketkä ovat tulleet Suomeen työskentelemään. Tämän koettiin mahdollisesti johtuvan siitä, että he ovat omaksuneet suomalaisen suoran palautteen kulttuurin ja on myöskin rakentunut syvempi luottamus nostaa asioita esille, kun ollaan tavattu ja työskennellään kasvotusten. Toisaalta tämä on osin ristiriidassa aiempien tutkimusten kanssa, jotka esiteltiin luvussa 5.5. Tutkijan mielestä tämä on hyvin erikoista, sillä ketterät periaatteet ja SAFe-viitekehys painottavat kasvokkain käytävää välitöntä kommunikaatiota. Tämän tutkimuksen mukaan on erittäin tärkeää sisäänrakennetun laadun parantamisen näkökulmasta, että kommunikaatio on mahdollisimman avointa ja projektissa vallitsee niin sanottu psykologisen turvallisuuden tunne, jolloin kaikki uskaltavat nostaa esille havaitsemiaan asioita ja ongelmia. Tutkimuksen tulosten perusteella on mahdotonta rakentaa asiakkaan liiketoiminnan tarpeisiin soveltuvaa ohjelmistoa, jos ei ole yhteistä ymmärrystä millaista ohjelmistoa ollaan rakentamassa, mikä asia sillä halutaan ratkaista ja millaisia ominaisuuksia ohjelmistolta vaaditaan. Tämän takia onkin erittäin tärkeää varmistaa, että kommunikointi on avointa ja läpinäkyvää, jotta kaikki tiimin jäsenet ovat tietoisia projektin tilanteesta ja voivat tarvittaessa tarjota apuaan.

Aiempien luvussa 3.6 esiteltyjen tutkimusten mukaan jatkuva retrospektiivinen arviointi ja toimintatapojen jatkuva parantaminen ovat tärkeitä palasia ketterässä ohjelmistokehitysprojektissa. Haastattelujen perusteella projektissa ei ole hyödynnetty ollenkaan retrospektiivejä eikä näin myöskään objektiivista itsearviointia ole tehty. Tämä on johtunut haastateltavien mukaan siitä että, projektin alkuvaiheessa, kun retrospektiivejä pidettiin, niin niiden ei koettu olevan hyödyllisiä eikä omaa tekemistä arvioitu objektiivisesti. Retrospektiivit kuitenkin koettiin haastatteluissa erittäin tärkeäksi osaksi ketterään ohjelmistokehitystä ja sisäänrakennetun laadun parantamista, sillä niiden avulla tiimi voi arvioida omaa toimintaansa ja tunnistaa kehityskohteita. Erittäin tärkeäksi nähtiin se, että niissä tulisi keskittyä konkreettisiin toimenpiteisiin, joilla tiimin toimintaan voidaan kehittää ja näin sisäänrakennettua laatua parantaa. Edellä mainittujen tutkimuksen tulosten sekä muiden tässä tutkimuksessa käsiteltyjen

tutkimusten perusteella liittyen jatkuvaan parantamiseen ja itsearviointiin retrospektiivit tulisi ottaa jatkuvaksi käytänteeksi SAFe- ja Scrum-viitekehyksien mallin mukaisesti. Tutkimustulosten perusteella niitä tulisi pitää säännöllisesti jokaisen iteraation jälkeen yhdessä tuote- ja kehitystiimin kesken. Lisäksi haastattelun perusteella nousi esille tarve järjestää myös jokaisen 10 viikon kehitysjakson jälkeen eli inkrementin jälkeen luvussa 2.4.3 esitelty Inspect&Adapt-tapahtuma vain tuotetiimin jäsenten kesken. Tämän koettiin olevan hyödyllistä, jotta myös tuotetiimi voisi arvioida mitkä asiat ovat menneet hyvin ja mitkä puolestaan vaativat kehittämistä ja kenties uusien käytänteiden kokeilua. Sisäänrakennetun laadun kehittämisen näkökulmasta Inspect &Adapt-tapahtuma olisi erittäin hyvä käytäntö tutkimuksen kohteena olevalle projektille jatkuvaan parantamiseen liittyen, koska sen avulla voidaan tunnistaa konkreettisia kehityskohteita ja niiden perusteella kehittää projektissa olevia toimintatapoja, joilla varmistetaan laadukkaamman ohjelmiston rakentaminen.

Haastatteluissa nousi myös esille että, konkreettisen kehittämiskohteiden tunnistamiseksi tulisi käyttää erilaisia ketteriä metriikoita kuten esimerkiksi historiallinen velositeetti sekä arvovirta-analyysi. Ketteristä metriikoista saatavan datan perusteella on helpompi katsoa mikä on mennyt hyvin ja mitä asioita tulisi parantaa ja tällöin päätökset perustuvat faktaan eikä mahdollisesti vääriin tuntemuksiin. Lisäksi samojen metriikkojen perusteella on helppo jälkikäteen arvioida sitä, onko valittujen kehityskohteiden parantaminen oikeasti onnistunut, kun voidaan verrata uusia tuloksia aiempiin tuloksiin. Tässä tutkimuksessa nousikin esille, että projektin kannalta merkittävien metriikoiden tunnistamiseen olisi hyvä ottaa mukaan agile valmentaja, joka osaa katsoa projektia hieman kauempaa ja näin tunnistaa objektiivisesti kehittämistä vaativia asioita. Lisäksi agile valmentajan apu retrospektiivi käytäntöjen muodostamisessa ja fasilitoinnissa olisi tutkimuksen perusteella erittäin hyödyllistä ja auttaisi parantamaan rakennettavan ohjelmiston sisäänrakennettua laatua tulevaisuudessa.

Kuten aiemminkin on jo mainittu, niin yksi SAFe-viitekehyksen sisäänrakennetun laadun elementistä on jatkuva kehityksen sykli ja tähän liittyvä jatkuvan

arvovirran tuottaminen. Jatkuvan arvovirran optimoinnin voidaan katsoa parantavan ohjelmiston sisäänrakennettua laatua. Haastattelujen perusteella jatkuvaa arvovirtaa voidaan optimoida tutkimuksen kohteena olevassa projektissa keskittymällä jakamaan kehitettävät toiminnallisuudet eli featuret tarpeeksi pieniksi kokonaisuuksia, jotta ne ehditään tekemään ja testaamaan yhden inkrementin aikana. Myöskin aiemmat tässä työssä luvussa 3.1 esitellyt tutkimukset ovat osoittaneet, että toiminallisuuksien ja vaatimusten jakaminen pieniksi kehityskokonaisuuksiksi vähentää riskiä virheisiin ja vaatimusten ymmärtämättömyyteen. Lisäksi tässä tutkimuksessa nousi esille, että projektissa vaatimusmäärittelyt ja käyttöliittymän suunnittelut ovat valmistuneet varsin myöhään ja tämän koetaan vaikeuttaneen jatkuvan arvovirran tuottamista ja suunnitelmallista tekemistä. Haastattelujen perusteella toteutukseen valmiita töitä olisi hyvä olla kehitysjonossa kahdeksi seuraavaksi 10 viikon kehitysjaksoksi, jotta myöskin vaatimusmäärittelyä ja käyttöliittymä suunnittelua voidaan tehdä laadukkaasti. Tämän tutkimuksen mukaan seuraavassa vaiheessa olisikin hyvä keskittyä tähän, jotta saadaan muodostettua priorisoitu työjono, josta toteutustiimin on helppo suunnitella työnsä aina seuraavalle inkrementille ja näin pystytään varmistamaan jatkuva kehityksen sykli ja laadukas arvovirta asiakkaalle.

Tutkimuksessa tuloksissa sekä aiemmissa tässä työssä esitellyissä tutkimuksissa luvuissa 3.1 ja 3.4 nousi esille jatkuvaan kehityksen sykliin ja julkaisun laatuun liittyviin sisäänrakennetun laadun elementteihin läheisesti kytköksissä olevat DoR- ja DoD-laatuportit. Ne ovat keskeisiä tarkastuspisteitä ketterässä ohjelmistokehityksessä ja niiden avulla voidaan varmistaa yhteinen ymmärrys siitä, mitä on tarkoitus tehdä ja miten se tehdään vaatimusten mukaisesti. Tutkimusten tulosten perusteella nämä laatuportit ovat projektissa käytössä, mutta niiden tarkempi määrittely sekä koko projektin yhteinen ymmärrys siitä mitä vaaditaan, että jokin toiminnallisuus voidaan ottaa työn alle tai sen voidaan todeta olevan valmis konkreettisesti tarkoittavat. Tutkimuksen mukaan projektin olisi hyvä käydä nämä käytössä olevat DoR ja DoD-kriteerit läpi ja varmistavan, että jokaiselle projektissa työskentelevällä on sama ymmärrys kriteereistä ja niiden täyttymisestä. Tutkimuksen tulosten perusteella esiin nousi myöskin DoD-laatuporttiin ja laadunvarmistukseen liittyvä niin sanottu

”valmiin määritelmä”. Etenkin laadunvarmistuksen ja testauksen näkökulmasta ohjelmiston sisäänrakennettua laatua voitaisiin kehittää, kun hyväksymistestaukseen siirretään vain ”valmiita” toiminallisuuksia. Tulosten perusteella hyväksymistestaukseen on siirretty toiminallisuuksia, jotka eivät ole olleet valmiita. Tämä on lisännyt merkittävästi hyväksymistestausvaiheen työmäärää ja keskittyminen vain hyväksymistestauksessa testattaviin kokonaisuuksin on kärsinyt. Tämän tutkimuksen mukaan sisäänrakennetun laadun viidennen elementin julkaisun laadun näkökulmasta olisikin erittäin tärkeää, että ennekuin toiminallisuuksia siirretään hyväksymistestaukseen niin, kaikki vaatimuksissa olleet ominaisuudet on rakennettu sekä niille on suoritettu aiemmat yksikkötestaukset sekä integraatiotestaukset laadukkaasti. Tutkimuksen tulosten perusteella tulisi yhdessä käydä läpi vaatimukset, milloin toiminnallisuudet voidaan siirtää hyväksymistestaukseen, jotta ne ovat kaikille selvät. Tähän liittyen olisi hyvä konkreettisesti käydä läpi voiko toiminnallisuuteen liittyen olla esimerkiksi auki kriittisen tai korkean tason virheitä, jotka on löydetty jo integraatiotestauksessa tai miten toimitaan, kun havaitaan, että toiminallisuuden kaikki määritellyt vaatimukset eivät ole täyttyneet integraatiotestausvaiheessa.

Koodin laatu on yksi SAFe-viitekehyksen sisäänrakennetun laadun viidestä elementistä. Tutkimusten tulosten perusteella tutkimuksen kohteena olevassa ohjelmistokehitysprojektissa on havaittu laatuongelmia liittyen koodiin laatuun. Projektissa havaittujen laatuongelmien korjaaminen aiemmissa tutkimuksissa esitellyillä käytännöillä kuten koodikatselmuksilla voisi olla tässä tapauksessa hyödyllistä ja parantaa sisäänrakennettua laatua. Toisaalta tavoitteena tulisi olla näiden koodin laatuun liittyvien ongelmien ennaltaehkäisy, jotta ohjelmisto on rakennettu alusta alkaen laatu huomioiden, kuten ketterät ohjelmistokehitys menetelmät ja periaatteet painottavat. Tämän tutkimuksen mukaan näiden ongelmien ennalta ehkäisyyn tulisi kiinnittää erityisesti huomiota esimerkiksi koodiin laatuun liittyvien tavoitteiden ja vaatimusten läpikäynnillä kaikkien projektin sidosryhmien kesken. Lisäksi tulisi tehdä suunnitelma siitä, miten näihin vaatimuksiin jatkossa päästään ja kuinka sitä seurataan. Esimerkiksi laadukas yksikkö- ja automaatiotestaus voivat parantaa koodin laatua, sillä niiden avulla koodin virheet ja puutteet on mahdollista havaita aiemmin, kuten tässä tutkimuksessa luvussa 3.2 esitelty shift left-periaate esittää. Tämän avulla voidaan

vähentää virheiden korjaamiseen kuluvaa aikaa ja näin parantaa ohjelmiston sisäänrakennettua laatua.

Sisäänrakennettua laatua voidaan kehittää ohjelmistokehitysprojektissa tässä luvussa aiemmin läpikäydyillä käytänteillä ja toimintavoilla. Erityisen tärkeää on pyrkiä toimimaan ketterien periaatteiden mukaisesti hyödyntämällä ketteriä menetelmiä tai osaa menetelmien käytänteistä projektiin sopivalla tavalla. Kaiken kaikkiaan ketterän ohjelmistokehitysprojektin sisäänrakennetun laadun parantaminen vaatii systemaattista ja jatkuvaa työtä kaikilta projektin osapuolilta. On tärkeää, että laatu on osa projektin kulttuuria ja että kaikki tiimin jäsenet ymmärtävät sen merkityksen ja tekevät aktiivisesti töitä sen eteen. Tämä edellyttää selkeästi sovittuja toimintatapoja ja käytäntöjä sekä selkeää kommunikaatiota ja yhteistyötä kaikkien projektin osapuolten välillä.

6.2 Tutkimuksen luotettavuus ja eettisyys

Reliabiliteetti ja validiteetti ovat keskeisiä käsitteitä, kun arvioidaan tieteellisen tutkimuksen luotettavuutta. Reliabiliteetilla tarkoitetaan tulosten pysyvyyttä ja toistettavuutta, joka tarkoittaa sitä, että voidaanko olettaa, että jos tutkimus uusitaan, niin samasta tutkimusaineistosta saadaan samat tutkimustulokset. Validiteetilla puolestaan määritellään tulosten yleistä luotettavuutta sekä miten hyvin tutkimus mittaa sitä, mitä sen on tarkoitus mitata. Validiteetin avulla voidaan tarkastella, että kohdistuuko tutkimus oikeisiin asioihin ja ovatko tulokset luotettavia ja päteviä. (Kananen, 2017, s. 174.) Hirsjärven ym. (2007) mukaan on erittäin tärkeää että, henkilöiden, paikkojen ja tapahtumien kuvaukset sekä tarkka selvitys tutkimuksen toteuttamisen kaikista vaiheista tutkimuksen luotettavuuden kannalta. Tämän johdosta tämän tutkimuksen luotettavuutta onkin pyritty lisäämään läpinäkyvyyden avulla: kaikki tutkimusprosessin liittyvät valinnat, vaiheet, haastateltavat sekä tapahtumat on pyritty kuvaamaan ja perustelemaan huolellisesti, jolloin tutkimus on helposti toistettavissa, jonka voidaan ajatella lisäävän tutkimuksen luotettavuutta. Hirsjärven ym. (2017, s. 178) tutkimuksen luotettavuuden kannalta on tärkeää, että tutkimuksen alkuperäinen tutkimusaineisto on säilytettävä, jotta tutkimustulosten luotettavuus ja aitous

olisi todennettavissa myöskin jälkikäteen. Tämän johdosta tutkimusaineisto säilytetään huolellisesti ja turvallisesti tutkijan toimesta sähköisessä muodossa. Myöskin tutkimuksen menetelmälliset valinnat ja aineistonkeruumenetelmä on perusteltu ja avattu tässä tutkimusraportissa luotettavuuden lisäämiseksi. Lisäksi haastateltavien suoria lainauksia on nostettu esiin tutkimusten tulosten esittelyn yhteydessä, jotta tutkimustuloksia voidaan tarkastella kriittisesti ja arvioida johtopäätösten luotettavuutta.

Tutkimuksen validiteettia voidaan puolestaan Hirsjärven ym. (2007) mukaan tarkastella esimerkiksi triangulaation avulla eli vertaamalla tutkimustuloksia aiempiin tutkimuksiin. Tätä vertailua voidaan heidän mukaansa esimerkiksi tehdä tutkimuksen tuloksia raportoidessa viittaamalla aiempiin tutkimuksiin ja kirjallisuuteen. Tuomen ja Sarajärven (2009, s. 141-142) mukaan triangulaation avulla onkin mahdollista lisätä tutkimuksen validiteettia. Tutkimuksen validiteetin kasvattamiseksi tässä tutkimuksessa on käytetty viittauksia aiempiin tutkimuksiin ja kirjallisuuteen tutkimustulosten esittämisen yhteydessä sekä johtopäätöksissä. Lisäksi tutkimuksen aiheena olevaa ohjelmistokehitysprojektia on tarkasteltu monipuolisesti erilaisista näkökulmista, jotka nousivat esiin aiemmissa tutkimuksissa.

Kun kyseessä on laadullinen tapaustutkimus, niin se ei pyri edustamaan yleistettävyyttä, joten sen tuloksia ei voida suoraan hyödyntää muihin ketteriin ohjelmistokehitysprojekteihin. Kuitenkin tätä tutkimusta ja sen tuloksia voidaan käyttää suuntaa antavana tutkimuksena ohjelmistokehitysprojekteille, joissa käytetään ketteriä ohjelmistokehityksen menetelmiä ja halutaan keskittyä sisäänrakennetun laadun kehittämiseen. Tässä tutkimuksessa tutkimuksen luotettavuuteen ja yleistettävyyteen negatiivisesti vaikuttavana tekijänä voidaan nähdä tutkimuksen kapea otanta. Tutkimuksessa oli kohteena vain yksi ohjelmistokehitysprojekti, joten tämä rajoittaa tutkimuksen yleistettävyyttä. Tuloksista ei voida siis tällaisenaan vetää kaikkia ketteriä ohjelmistokehitys projekteja koskevia johtopäätöksiä. Lisäksi tutkimuksen kohteena ollut ketterä ohjelmistokehitysprojekti keskittyy yritysten väliseen liiketoimintaan, mitä voidaan pitää mahdollisena rajoitteena tutkimuksen yleistettävyydelle.

Grossoehmen (2014, s.111) mukaan tutkimuksen luotettavuutta lisää se, kun tutkimuksen aineisto analysoidaan usean tutkijan toimesta, jotta tutkijan vaikutus aineistoon vähenee ja muun muassa ennakko-oletusten ja väärinymmärrysten mahdollisuudet vähenevät. Tässä tutkimuksessa, kuten yleensä muissakin ylempien ammattikorkeakoulujen tutkimuksellisissa kehittämistöissä tutkijoita on vain yksi, niin tämän voidaan nähdä heikentävän tutkimusaineiston analyysin täydellistä objektiivisuutta. Lisäksi kahdeksan haastattelun pohjalta koottua tutkimusaineistoa ei voitane pitää täysin luotettavana, jolloin tutkimuksen tärkein arvo on ymmärtää ja verrata näiden henkilöiden henkilökohtaisia ajatuksia ja kokemuksia aiempiin tutkimuksiin aiheesta. Aineiston analyysin luotettavuutta onkin pyritty lisäämään haastattelujen litteroinnilla ja huolellisella teorialähtöisellä aineiston analysoinnilla, kuten luvussa 4.3 on kuvattu.

Tuomen & Sarajärven (2009, s. 126-128) mukaan monet tekijät vaikuttavat tutkimuksen eettisyyteen, alkaen tutkimusaiheen valinnasta ja etenemisestä aina tutkimuksen kuvaamiseen ja raportointiin asti. Eettiset näkökohdat vaikuttavat merkittävästi myös tutkimuksen laatuun ja sen luotettavuuteen. Tutkimuksen eettisyyden arviointi on keskeistä erityisesti silloin kun tutkimus liittyy ihmisiin. On ehdottoman tärkeää varmistaa, että kaikilta tutkimukseen osallistuvilta on saatu suostumus ja että tutkimus ei aiheuta heille haittaa. Myöskin tutkittavien anonymiteetti on turvattava, jotta heidän henkilöllisyytensä pysyy salassa. (Gray 2009, 69.) Tässä tutkimuksessa haastattelut perustuivat vapaaehtoisuuteen ja kaikki haastateltavat osallistuivat haastatteluihin täysin omasta tahdostaan. Lisäksi tutkimuksessa on varmistettu, että suorat lainaukset ja muut viittaukset aineistoon on tehty siten, että yksittäiset vastaajat eivät ole tunnistettavissa. Tämän vuoksi tarkempia tietoja kuten haastateltavien ikää, sukupuolta tai työtehtävää ei ole tuotu esiin tutkimusten tulosten raportoinnissa ja läpikäynnissä. Myös haastatteluaineisto säilytetään huolellisesti ja tietoja käsitellään äärimmäisellä tarkkuudella.

Gray myöskin (2009, s.91) korostaa, että tutkijan on kiinnitettävä erityistä huomiota eettisyyteen silloin, kun hän työskentelee samassa organisaatiossa, jossa tutkimus toteutetaan. Tällaisessa tilanteessa tutkijan ja haastateltavien

välillä voi syntyä haasteita, jossa haastateltavat eivät välttämättä uskalla kertoa rehellistä mielipidettään tutkittavasta ilmiöstä. Tässä tutkimuksessa tutkija työskentelee tutkimuksen kohteena olevassa projektissa hierarkisesti vastaavassa roolissa haastateltavien kanssa, joten esimerkiksi esimies-alainen suhde ei vaikuta tutkimukseen. Lisäksi tutkija rohkaisi haastateltavia rehellisten mielipiteiden ja kokemusten esiin nostamiseksi sillä, että tutkimusaineisto tullaan anonymisoimaan, jotta haastateltavia ei voida tunnistaa vastauksista sekä haastattelut ovat täysin luottamuksellisia.

Tutkimuseettinen neuvottelukunta on kehittänyt tutkimuseettiset ohjeet, jotka sisältävät keskeisiä periaatteita hyvistä tieteellisistä käytännöistä. Nämä periaatteet korostavat rehellisyyttä, yleistä huolellisuutta ja tarkkuutta tutkimusprosessin eri vaiheissa. Lisäksi ohjeissa painotetaan avoimuutta ja vastuullista tiedeviestintää tutkimustulosten raportoinnissa. Ohjeissa korostetaan myös muiden tutkijoiden työn kunnioittamista, esimerkiksi asianmukaisten viittausten käyttöä. (TENK, 2023, s.11-15.) Tässä opinnäytetyössä on pyritty noudattamaan näitä hyviä tieteellisiä käytäntöjä ja eettisiä periaatteita mahdollisimman tarkasti, jotta voidaan varmistaa tutkimuksen korkea laatu.

6.3 Pohdinta ja mahdolliset jatkotutkimukset

Tämän tutkimuksellisen kehittämistyön tutkimusongelmana oli selvittää keinoja, miten tutkimuksen kohteena olevan ketterän ohjelmistokehitys projektin sisäänrakennettua laatua saadaan parannettua, jotta rakennettava ohjelmisto vastaa mahdollisimman hyvin sille asetettuja vaatimuksia. Tavoitteena oli tuottaa konkreettisia kehitysehdotuksia tutkimuksen kohteena olevan ketterän ohjelmistokehitysprojektin sisäänrakennetun laadun parantamiseksi. Tutkimus aloitettiin tutustumalla ilmiöön liittyviin teorioihin kuten ketteriin periaatteisiin ja menetelmiin, SAFe-viitekehykseen sekä viitekehyksen ydinarvoihin, joihin sisäänrakennettu laatu sisältyy. Lisäksi ennen empiirisen tutkimuksen toteutusta käytiin läpi lukuisia tutkimuksia miten erilaisten ketterien käytänteiden ja menetelmien avulla sisäänrakennettua laatua voidaan parantaa. Aiheeseen liitty-

vän kirjallisuuden ja aiempien tutkimusten perusteella muodostettiin teemahaastattelurunko ja teemoihin liittyvät kysymykset. Tämän jälkeen suoritettiin empiirinen tutkimus, jonka aineiston litteroitiin sekä analysoitiin systemaattisella sisällönanalyysillä. Lisäksi esitettiin tutkimustulokset ja verrattiin niitä aiempiin tutkimuksiin. Lopuksi tutkija teki näiden tutkimustulosten ja aiempien tutkimusten perusteella johtopäätökset tuloksista ja esitti myöskin konkreettisesti kehitysehdotukset.

Tutkijan mielestä tutkimusprosessi onnistui melko hyvin ja tätä edesauttoi erityisesti huolellinen tutustuminen tutkimukseen aiheena olevaan ilmiöön sekä aiheeseen liittyviin aiempiin tutkimuksiin. Eniten haasteita aiheutti empiirisellä tutkimuksella kerätyn aineiston analysointi ja keskittyminen vain kaikista olennaisimpiin aineistosta esiin nousseisiin teemoihin. Tämä johtunee siitä että, haastatteluissa syntyi hyvin paljon aineistoa liittyen jokaiseen tutkimuksen tuloksissa esitettyyn teemaan liittyen. Vaikka tutkimuksen aihe olikin melko tarkkaan rajattu keskittymään sisäänrakennettua laatua, niin tutkimuksen toteuttamisen aikana tutkija ymmärsi, että vaikka sisäänrakennettu laatu onkin vain yksi SAFe-viitekehyksen ydinarvosta, niin se on erittäin laaja kokonaisuus ja oikeastaan kaikki ohjelmistokehitysprojektin prosessit ja toimintatavat vaikuttavat sisäänrakennettuun laatuun joko positiivisesti tai negatiivisesti. Vaikka ketterä ohjelmistokehitys ja ketterät menetelmät olivatkin tutkijalle tuttuja jo ennen tutkimuksellisen kehittämistyön toteuttamista niin, tutkimuksen johdosta myöskin tutkija syvensi ymmärrystään ja osaamistaan liittyen ketterään ohjelmistokehitykseen sekä ketteriin periaatteisiin ja menetelmiin. Suurin tutkijan oivallus oli se, että pelkkä ketterien menetelmien tai viitekehyksen käyttö ei tee ohjelmistokehitysprojektista ketterää vaan todellista ketteryyttä on mahdollista saavuttaa vain pyrkimällä toimimaan ketterien periaatteiden mukaisesti. Lisäksi tutkija ymmärsi että, ketterät menetelmät, käytänteet sekä erilaiset viitekehykset ovat vain konkreettisia työkaluja joiden avulla on helpompi toimia ketterien periaatteiden mukaisesti, jos niitä osataan hyödyntää oikein.

Tutkimuksen tulosten hyödynnettävyyttä kyseisessä ohjelmistokehitysprojektissa voitaisiin opinnäytetyön tekijä mielestä mitata esimerkiksi myöhemmin tehtävällä jatkotutkimuksella, jossa tutkitaan, onko opinnäytetyön tuloksena

nousseista kehitysehdotuksista ja havainnoista ollut hyötyä konkreettisena sisäänrakennetun laadun parantumisena. Kyselyssä tulisi erityisesti keskittyä niihin seikkoihin, jotka opinnäytetyön tuloksena on nostettu esille sekä onko ohjelmistokehitysprojekti hyödyntänyt näitä opinnäytetyössä syntyneitä kehitysehdotuksia ja havaintoja. Tällä tavalla on mahdollista mitata, onko opinnäytetyön tuloksena syntyneet kehitysehdotukset sekä havainnot relevantteja ja ovatko ne vaikuttaneet sisäänrakennetun laadun kehittymiseen tutkimuksen kohteena olleen projektin jatkovaiheessa.

LÄHTEET

Abbas, N., Gravell, A. M. & Wills, G. B. (2008). Historical Roots of Agile Methods: Where Did “Agile Thinking” Come From? School of Electronics and Computer Science, University of Southampton.

Agile Alliance. (2023). What is Agile? Viitattu 1.5.2023 https://www.agilealliance.org/agile101/

Balan, L., Bertoletti de Marchi, M. F., & Oliveira, A. C. (2019). Self-assessment of teamwork in Scrum projects for continuous improvement. In 2019 IEEE/ACM International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE.

Bass, L., Weber, I., & Zhu, L. (2015). Iterative architectural design and implementation to manage technical debt. Journal of Software: Evolution and Process, 27(10).

Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffrie, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. ja Thomas, D. (2001). Manifesto for Agile Software Development. Agile Alliance. Viitattu 2.5.2023. https://agilemanifesto.org/

Boehm, B. & Turner, R. (2005). Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley Longman Publishing Co.

Briand, L. C., Morasca, S., ja Basili, V. R. (2006). Property-based software engineering measurement. IEEE Transactions on software engineering 32(2).

Cockurn, A. & Highsmith, J. (2001). Agile Software Development: The Business of Innovation. Computer 34(9).

Conboy K. (2009). Agility from first principles reconstructing the concept of agility in information systems development. Information Systems Research.

Deming, W. E. (1986). Out of the crisis: Quality, productivity and competitive position. Cambridge University Press.

Dexter, S. (2020). Beware SAFe (the Scaled Agile Framework for Enterprise), an Unholy Incarnation of Darkness. Viitattu 13.3.2023. https://seandexter1.medium.com/beware-safe-the-scaled-agile-framework-for-enterprise-an-unholy-incarnation-of-darkness-bf6819f6943f

Dumitrou, F., Mesnita, G. & Radu, L. (2019). Challenges and Solutions of Applying Large Scale Agile at Organizational Level. University of Iasi. Viitattu 13.3.2023. http://revistaie.ase.ro/content/91/06%20-%20dumitriu,%20mesnita,%20radu.pdf

Dybå, T., Dingsoeyr, T. ja Moe, NB. (2014). Agile Project Management. Viitattu 19.2.2023. https://www.researchgate.net/publication/263276642_Agile_Project_Management

Evans, K. (2019). The Major Problems with Safe - Putting the Focus on Frameworks Rather Than Outcomes. Viitattu 13.3.2023. https://productcoalition.com/the-major-problems-with-safe-1e797f7e48f8

Fowler, M. (2010). Continous integration. Viitattu 15.3.2023. https://martinfowler.com/articles/continuousIntegration.html

Gray, D. E. 2009. Doing research in the real world. 2.p. Los Angeles: SAGE Publications.

Grossoehme, D. (2014). Research Methodology Overview of Qualitative Research. Journal of Health Care Chaplaincy. 20.

Johnson, P., & Tjahjadin, D. (2008). Improving software quality: The benefits of peer reviews. IEEE Software, 25(1).

Juran, J. M. (1999). Juran's Quality Handbook. New York: McGraw-Hill.

Hirsjärvi, S & Hurme, H. (2022). Tutkimushaastattelu. Teemahaastattelun teoria ja käytäntö. Helsinki: Yliopistopaino.

Hirsjärvi, S., Remes, P. & Sajavaara, P. 2007. Tutki ja kirjoita. Keuruu: Otavan Kirjapaino Oy.

Kananen, J. (2013). Case-tutkimus opinnäytetyönä. Tampere: Tampereen Yliopistopaino Oy.

Kananen, J. (2008). Kvali: Kvalitatiivisen tutkimuksen teoria ja käytänteet. Jyväskylä: Jyväskylän Yliopistopaino.

Kananen, J. (2017). Laadullinen tutkimus pro graduna ja opinnäytetyönä. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kannan, V., Jhajharia, S., & Verma, S. (2014). Agile vs waterfall: A Comparative Analysis.

Lacey, M. (2015). The Scrum Field Guide: Agile Advice for Your First Year and Beyond. Addison-Wesley Professional.

Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Pearson Education, Inc.

Larman, C. & Basili, V. R. (2003). Iterative and incremental developments. a brief history. Computer, 36(6).

Leffingwell, Dean. (2007). Scaling Software Agility: Best Practices for Large Enterprises. Addison-Wesley Professional.

Linz, T. (2014). Testing in Scrum. Rocky Nook.

Liu, S., & Bai, C. (2009). Integrating product design and development for quality improvement. Journal of Engineering Design, 20(3).

Mikkonen, T. & Taipale, J. (2017). The Importance of Flow Efficiency as a Key Metric in Software Development. Journal of Software Engineering Research and Development, 5(1).

Misra, S., Kumar, V. & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. Journal of Systems and Software, 82(11).

Misra, S., Kumar, V., & Kumar, U. (2010) Identifying some critical changes required in adopting agile practices in traditional software development projects. 27(4).

Mustafee, N., & Taylor, S. J. E. (2016). Quality assurance in agile software development: A multi-methods study. Journal of Systems and Software

Myers, M. D. & Newman, M. (2007). The qualitative interview in IS research: Examining the craft. Information and Organization, 17(1).

Nagappan, N., Maximilien, E. M., & Williams, L. (2006). Using high-coverage testing to detect concurrency defects in concurrent programs. IEEE Transactions on Software Engineering, 32(9).

Niazi, M., Mahmood, K., ja Alshayeb, M. (2018). Agile software development methodologies and practices: A systematic review. Journal of Systems and Software.

Ojasalo, K., Moilanen, T. ja Ritalahti, J. (2020). Kehittämistyön menetelmät – Uudenlaista osaamista liiketoimintaan. Helsinki: Sanoma Pro.

Zulkarnain, A., & Ali, N. (2017). A case study on the impact of definition of done on software quality in agile development. Journal of Software Engineering and Applications, 10(02).

It-wiki. (2023). Ohjelmistokehitys. Viitattu 18.2.2023. https://www.itewiki.fi/opas/ohjelmistokehitys/

Rigby, P. C., Potvin, P., & German, D. M. (2014). The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects. IEEE Transactions on Software Engineering, 40(12).

Saaranen-Kauppinen, A. ja Puusniekka, A. (2006). KvaliMOTV. Viitattu 21.4.2023 https://www.fsd.tuni.fi/menetelmaopetus/kvali/index.html

Sánchez-Gordón, M. L., García, F. M. ja Fernández-Sanz, L. (2018). Assessing the impact of agile practices on software quality. Journal of Systems and Software.

Scaled Agile, Inc (2021). Scaled Agile Framework. Viitattu 3.5.2023 https://www.scaledagileframework.com/

Schultze, U. & Avital, M. (2011). Designing interviews to generate rich data for information systems research. Information and Organization, 21(1).

Schwaber, K., & Sutherland, J. (2020). The Scrum Guide. Scrum.org. Viitattu 26.4.2023 https://www.scrum.org/resources/scrum-guide

Shahin, A., Sharifi, H., & Parsa, F. (2002). Relationship between quality improvement programs and firm performance. International Journal of Production Economics, 79(3).

Stack Overflow. (2018). Developer Survey Results. Viitattu 3.3.2023 https://insights.stackoverflow.com/survey/2018#development-practices

Stankovic, D., Nikolic, V., Djordjevic, M. & Cao, D. B. (2013). A survey study of critical success factors in agile software projects in former Yugoslavia IT companies. Journal of Systems and Software, 86(6).

Sweeney, M. (2022). Agile vs Waterfall: Which Method is More Successful? Clearcode. Viitattu 3.3.2023. https://clearcode.cc/blog/agile-vs-waterfall-method/

TENK. 2023. Hyvä tieteellinen käytäntö (HTK). Helsinki. Viitattu 17.5.2023. https://tenk.fi/sites/default/files/2023-03/HTK-ohje_2023.pdf

Tuomi, J. & Sarajärvi, S. (2009). Laadullinen tutkimus ja sisällönanalyysi. Helsinki: Tammi.

Rouse, M. (2022). What is software development? Definition, stages, tools. TechTarget.

Reinertsen, D. (2009). The principles of product development flow: second generation lean product development. Celeritas Publishing

Wijaya, D. T., & Raharjo, H. (2018). Defining the definition of ready in agile software development: An empirical investigation. 2018 IEEE International Conference on Industrial Engineering and Engineering Management.

Yang, C., Liang, P. & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. Journal of Systems and Software 111.

Yin, R. 2018. Case study sesearch and applications: design and methods. Los Angeles: SAGE.

LIITTEET

LIITE 1. Teemahaastattelurunko

1. Ketterät menetelmät ja SAFe-viitekehys
    - Tuoko viitekehysten tapahtumat, roolit ja periaatteet mielestäsi lisäarvoa vai käytetäänkö käytänteitä vain koska niitä tulee käyttää? (esim. selkeyttääkö roolit vastuita/päätöksentekoa, auttaako työn priorisoinnissa, PI / Sprint suunnittelut, demot)
    - Millaisia hyötyjä/haittoja viitekehysten ketterien käytänteiden käyttö on mielestäsi tuonut?

2. Sisäänrakennettu laatu
    - Mitä sisäänrakennettu laatu tarkoittaa mielestäsi?
    - Mistä sisäänrakennettu laatu koostuu?

3. Jatkuva arvovirta
    - Miten iteraatio/inkrementaalinen tason story/feature toteuttaminen (koodaus ja testaaminen) on mielestäsi onnistunut sprinttien ja inkrementtien puitteissa?
    - Onko päällekkäistä tekemistä (Work In Progress) onnistuttu rajoittamaan, jotta ei synny ylimääräisiä pullonkauloja ja saadaan suunnitellut työ valmiiksi?
    - Miten mielestäsi DoR-laatuportin käytössä on onnistuttu?? Onko tuonut positiivia vaikutuksia vai aiheuttanut ongelmia, kuten epätietoisuutta mitä tehdään ja miten tehdään?

4. Järjestelmän arkkitehtuuri ja koodin laatu
    - Onko arkkitehtuurissa otettu mielestäsi hyvin huomioon ohjelmiston suorituskyky, luotettavuus ja ylläpidettävyys?
    - Onko mielestäsi valmistuvalla ohjelmisto paljon teknistä velkaa? Jos on, niin mistä tämä johtuu?
    - Onko koodin (code quality) laadun varmistamista tehty mielestäsi tarpeeksi? Millaisia menetelmiä tähän on tietääksesi käytetty? (Esim. koodin katselmus, yhteisomistajuus, pariohjelmointi?)

5. Laadunvarmistus ja testaaminen
    - Miten koet eritasoisten testausten sujuneen? (yksikkö, SIT, UAT ja automaatiotestaus) sujuneen? Oletko havainnut ongelmia? Millaisia?

- Havaitaanko virheet (bugit) mielestäsi tarpeeksi aikaisissa testausvaiheissa niiden korjaamiseksi mahdollisimman aikaisin ja nopeasti? (ns. Shif-left periaate)
- Onko tästä ollut hyviä/huonoja seurauksia sisäänrakennettuun laatuun?
- Miten DoD-laatuportin käytössä on mielestäsi onnistuttu?
- Miten yhteistyö laatutiimin (UAT-tiimi) ja kehitystiimin (toimittaja) on mielestäsi sujunut?

6. Jatkuva parantaminen ja itsearviointi
    - Onko yhteistyö ja kommunikointi parantuneet projektin edetessä?
    - Onko kommunikointi mielestäsi tarpeeksi avointa vai yritetäänkö asioita ”piilotella”?
    - Opitaanko retroista oikeasti asioita, miten seuraavissa iteraatioissa voitaisiin toimia paremmin?