Matias Partanen

KETTERÄN MENETELMÄN RÄÄTÄLÖINTI

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2014

TIIVISTELMÄ

Partanen, Matias
Ketterän menetelmän räätälöinti
Jyväskylä: Jyväskylän yliopisto, 2014, 105 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Leppänen, Mauri

Ketterien menetelmien käyttäminen on lisääntynyt 2000–luvulla. Mikään niistä ei kuitenkaan sovellu sellaisenaan käytettäväksi, vaan niitä tulee räätälöidä vastaamaan paremmin kehittämiskontekstia ja sen erityispiirteitä. Tässä tutkielmissa tarkastellaan ketterien menetelmien räätälöintiä. Tarkoituksena on selvittää, millä tavoilla ketteriä menetelmiä on esitetty räätälöitäväksi ja miten niitä on käytännössä räätälöity ja minkälaisin kokemuksin. Tutkimus perustuu laajaan kirjallisuuskatsaukseen.

Tutkielmassa esitellään ensin lyhyesti, mitä kehittämismenetelmällä tarkoitetaan, millä tavoilla menetelmän kehittämistä ja räätälöintiä voidaan suorittaa sekä millaisia ohjelmistokehitykseen ja räätälöintiin vaikuttavia tilannetekijöitä on olemassa. Tämän jälkeen tutkielmassa käsitellään ketterää lähestymistapaa ja kuvataan ketteristä menetelmistä tarkemmin Scrumia, XP:tä ja Kanbania prosessien, roolien ja käytänteiden näkökulmasta.

Tämän jälkeen tutkitaan millaisia teoreettisia ehdotuksia ja ohjeita ketterän menetelmän räätälöimiseksi on esitetty. Yhdeksästä tutkimuksesta selvitetään erityisesti räätälöintistrategia ja -lähtökohta, räätälöintiprosessi sekä räätälöinnin kohde. Tämän jälkeen käydään läpi ketterän menetelmän räätälöintiä koskevia tapaustutkimuksia. Kuuden tapaustutkimuksien käsittely tapahtuu työssä luodun ketterän menetelmän viitekehyksen avulla, joka koostuu kehittämisympäristön kontekstista ja sen erityispiirteistä, prosessista ja strategiasta, räätälöinnin tuloksesta ja kokemuksien haltuun ottamisesta. Tutkielman tuloksista selviää, millä eri tavoin ketteriä menetelmiä on esitetty räätälöitäviksi sekä millaisiin tilanteisiin ketteriä menetelmiä on räätälöity, miten sitä on tehty ja millaisia tuloksia räätälöinnistä on saatu. Tuloksia voidaan hyödyntää ketterää menetelmää räätälöitäessä organisaation tai projektin käyttöön.

Asiasanat: ketterä menetelmä, räätälöinti, konfigurointi, kustomisointi, Scrum, XP, Kanban

ABSTRACT

Partanen, Matias
Tailoring Agile Methods
Jyväskylä: University of Jyväskylä, 2014, 105 p.
Information Systems Science, Master's thesis
Supervisor: Leppänen, Mauri

During the last decade we have been able to witness rapid increase of the use of agile methods However, organizations and projects rarely adopt and use the methods "by the book", but rather tailor them to better meet the development contexts and their features.

The purpose of this thesis is to find out which kinds of suggestions for tailoring agile methods have been presented in the literature and how agile methods have been tailored in practice, and with which kinds of experience. The thesis is based on a comprehensive literature review.

This thesis will first briefly discuss what the method means, in which ways it can be engineered and tailored and what situational factors are related to software development. After that the concepts of agility and agile method will be discussed. Three agile methods, Scrum, XP and Kanban, are described in terms of processes, roles and practices.

Third, the thesis will discuss what kinds of theoretic suggestions and instructions have been given regarding tailoring of agile methods. Nine studies will be discussed with the emphasis on tailoring strategy and basis, tailoring process and tailoring target. Fourth, the thesis will describe and analyze six case studies on agile method tailoring based on a framework composed of development context and its special features, tailoring process and strategy and outcome. The results of this thesis show what kind agile method tailoring suggestions and instructions have been made, what kinds of situations agile methods have been tailored for, how it has been done and with what kind of experiences. The results can be used when tailoring an agile method for the use of an organization or a project.

Keywords: agile method, tailoring, configuration, customization, Scrum, XP, Kanban

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ohjelmistokehitys perustuu liki poikkeuksetta jonkinlaiseen menetelmään. Menetelmällä tarkoitetaan Iivarin, Hirschheimin ja Kleinin (1998) mukaan organisoitua kokoelmaa käsitteitä, metodeja, uskomuksia, arvoja ja ohjeellisia periaatteita, joita materiaaliset resurssit tukevat. Menetelmät eroavat toisistaan varsin paljon. Niiden tutkimisen ja valinnan helpottamiseksi on menetelmiä luokiteltu eri tavoin. Ne voidaan luokitella esimerkiksi formaaleihin ja ei-formaaleihin menetelmiin (Fitzgerald, 1996). Osa menetelmistä sisältää paljon etukäteissuunnittelua ja osa vähemmän (Boehm, 2002). Mikään menetelmä ei sovi kaikkiin kehittämistilanteisiin (Anderson, 2010).

Jotta menetelmästä saadaan mahdollisimman suuri hyöty käytännön ohjelmistokehitykseen, tarvitaan menetelmän räätälöintiä. Karlssonin ja Åkerfalkin (2004) mukaan räätälöinnillä tarkoitetaan menetelmän sopeuttamista erilaisten tilannetekijöiden mukaan. Räätälöinnin lähtökohtana voi olla yksittäinen menetelmä, ns. pohjamenetelmä (Karlsson & Åkerfalk, 2004), tai joukko menetelmiä, joiden piirteitä integroidaan uudeksi menetelmäksi (Leppänen, 2005). Räätälöinti voidaan tehdä joko tietyn organisaation tai projektin käyttöön. Käyttökonteksti tulee kuvata tilanne- eli kontigenssitekijöiden mukaisesti. Keskeisiä projektikohtaisia tilannetekijöitä voivat olla muun muassa projektin kesto, laajuus, budjetti, käytettävä teknologia sekä kehittäjien ammattitaito (Van Slooten ja Schoonhoven, 1994). Räätälöidyn menetelmän ominaisuuksien tulisi vastata näihin tekijöihin (Feiler & Humphrey, 1992).

Ketterät kehittämismenetelmät ovat voimakkaasti yleistyneet 2000-luvulla. Highsmith (2002) määrittelee ketteryyden kyvyksi luoda muutosta ja vastata muutoksiin, jotta yritys voi tehdä voittoa myrskyisässä liiketoimintaympäristössä. Ohjelmistokehityksessä ketteryyden määritelmä ja arvot pohjautuvat Agile–Allianssin (2001a; 2001b) ketterän ohjelmistokehityksen manifestiin ja sen periaatteisiin. Abrahamsson, Salo, Ronkainen ja Warsta (2002) määrittelevät menetelmän olevan ketterä, mikäli se ohjaa toimimaan inkrementaalisesti, korostaa yhteistoiminnallisuutta ja asiakaslähtöisyyttä, on suoraviivainen ja helposti opittava sekä valmis mukautumaan muutoksiin. Ketterällä kehittämisellä

on nähty olevan monenlaisia hyötyjä verrattuna perinteisiin menetelmiin. (Dybå & Dingsøyr, 2008).

Ketterän ohjelmistokehityksen alkuvaiheessa monet suosittivat ketterän menetelmän yhteydessä noudattamaan kaikkia sen käytänteitä, jotta se toimisi halutulla tavalla (esim. Beck, 1999). Viime vuosina tarve räätälöidä myös ketteriä menetelmiä on laajasti tunnustettu. Ketterän menetelmän räätälöinnistä on tehty useita tutkimuksia, joista osa käsittelee aihetta yleisellä tasolla perustuen käsitteellis-teoreettiseen tarkasteluun, kun taas osa perustuu tapaustutkimuksiin. Sen sijaan on puutetta tutkimuksesta, joka muodostaisi kokonaisvaltaisen käsityksen ketterien menetelmien räätälöinnistä ja kytkisi tämän aiempaan menetelmäkehitystä ja räätälöintiä yleisesti tarkastelevaan tutkimukseen.

Tämän tutkimuksen tarkoituksena on tutkia aihetta kokonaisvaltaisesti kirjallisuuskatsauksena. Tutkimuksen tutkimusongelmat voidaan määritellä seuraavasti:

- Millä perusteilla ja tavoilla ketteriä menetelmiä on esitetty räätälöitäväksi?
- Millaisiin kehittämistilanteisiin ketteriä menetelmiä on räätälöity, miten sitä on tehty ja millaisia kokemuksia räätälöinnistä on saatu?

Tutkimusongelmia on tarkennettu seuraavilla tutkimuskysymyksillä:

- Millaisista osista ketterät menetelmät koostuvat?
- Millä tavalla yleisestä menetelmäkehityksestä ja -räätälöinnistä esitetyt ajatukset ovat nähtävissä ketterien menetelmien räätälöinnille annetuissa jäsennyksissä ja ohjeissa?
- Millä tavalla ketterien menetelmien räätälöinnille esitettyjä jäsennyksiä ja yleisiä ohjeita on noudatettu tapaustutkimuksen kohteena olleissa tapauksissa?

Vastausten saamiseksi tutkimusongelmiin ja -kysymyksiin luodaan ensin käsitteellinen perusta yhtäältä menetelmiä, menetelmien kehittämistä ja räätälöintiä yleisesti käsittelevälle asiakokonaisuudelle ja toisaalta ketterää kehittämistä ja ketteriä menetelmiä käsittelevälle asiakokonaisuudelle. Ketteristä menetelmistä tarkastelun kohteiksi on valittu Scrum (Schwaber, 1995; Schwaber & Sutherland, 2013), eXtreme Programming (XP) (Beck, 1999; Beck & Andres, 2004) ja Kanban (Anderson, 2010).

Tämän jälkeen keskitytään työssä ketterien menetelmien räätälöinnistä julkaistuihin tutkimuksiin. Tutkimustietokantoihin (esim. Springer Link, IEEE Xplore, ScienceDirect, IGI Global ja ACM Digital Library) suunnatun laajaan kirjallisuushaun perusteella jaetaan tutkimukset kahteen osaan, niihin joissa tarkastellaan aihetta käsitteellisesti ja niihin joissa raportoidaan tapaustutkimuksista. Edellisistä esimerkkeinä ovat Ayed, Vanderose ja Habra (2012), jotka käsittelevät ketterän menetelmän räätälöintiä menetelmän metamallin ja laadun mittaamisen mallin avulla, Mikulenas ja Kapocius (2011), jotka käsittelevät räätälöintiä koostamalla menetelmän erilaisista ketteristä käytänteistä, sekä Qumer

ja Henderson-Sellers (2008), jotka esittelevät mallin, joka tarjoaa tiekartan ketterän menetelmän omaksumiselle organisaatioon vaiheittain. Tutkimuksista käydään läpi tarkemmin räätälöintistrategia ja -lähtökohta (esim. pohjamenetelmän käyttäminen), räätälöinnin kohde (esim. XP:n käytänteet) ja räätälöintiprosessi (esim. käytänteiden valitseminen arvoja ja kustannuksia vertailemalla).

Esimerkkeinä tapaustutkimuksista ovat Hong, Yoo ja Sungdeok (2010), jossa tutkitaan Scrumin räätälöintiä ulkoistetuissa elektronisen liiketoiminnan projekteissa, Díaz, Pérez, Yagüe ja Garbajosa (2011), jossa käsitellään Scrumin räätälöintiä ohjelmistotuoteperheiden yhteydessä sekä Scott, Johnson ja McCullough, joka tarkastelee Scrumin ja XP:n räätälöintiä julkisen sektorin ohjelmistokehitysprojektissa. Tapaustutkimuksista käydään läpi pohjamenetelmä (esim. Scrum), konteksti ja erityispiirteet (esim. julkinen sektori), räätälöintistrategia (esim. pilottiprojekti), räätälöidyn menetelmän eroavaisuudet (esim. vain tiettyjen käytänteiden käyttö) ja otettiinko kokemuksia räätälöinnistä haltuun.

Tutkimuksen tuloksia voidaan hyödyntää organisaatioissa, jotka pohtivat tapoja räätälöidä ketteriä menetelmiä joko organisaation käyttöön tai yksittäisen projektin käyttöön. Tuloksia voidaan käyttää hyödyksi myös jatkotutkimuksissa.

Tutkielma on jäsennetty seitsemään lukuun. Luvussa 2 esitetään, mitä kehittämismenetelmällä tarkoitetaan, millä tavoilla menetelmän kehittämistä ja räätälöintiä voidaan suorittaa sekä millaisia ohjelmistokehitykseen ja räätälöintiin vaikuttavia tilannetekijöitä on olemassa. Luvussa 3 esitetään, mitä ketterällä kehittämisellä tarkoitetaan ohjelmistokehityksen yhteydessä sekä esitetään ketteristä menetelmistä tarkemmin Scrum, XP ja Kanban. Luvussa 4 esitetään ketterän menetelmän räätälöintiä käsitteleviä käsitteellis-teoreettisia tutkimuksia. Luvussa 5 esitellään ketterän menetelmän räätälöintiä käsitteleviä tapaustutkimuksia. Luvussa 6 pohditaan tutkimuksen tuloksia ja vedetään johtopäätöksiä. Lopuksi esitetään yhteenveto.

2 MENETELMÄ JA SEN RÄÄTÄLÖINTI

Tämän luvun tarkoituksena on antaa yleiskuva siitä mitä menetelmällä ja sen räätälöinnillä tarkoitetaan. Ensiksi tarkastellaan sitä, mitä menetelmä tarkoittaa ohjelmistokehityksen kontekstissa, millaisia menetelmiä on olemassa, mistä osista se koostuu, mitä hyötyjä voidaan sen käyttämisellä saavuttaa sekä mitä haasteita sen käyttämiseen liittyy. Toiseksi jäsennetään menetelmän kehittämisen käsitettä viitekehyksen avulla. Kolmanneksi tarkastellaan menetelmän räätälöinnin käsitettä esittelemällä eri lähteissä olevia määritelmiä sekä tapoja esittää räätälöinti ja siihen vaikuttavia tekijöitä. Lopuksi käydään läpi ohjelmistokehitykseen vaikuttavia tilannetekijöitä ja sitä, miten ne liittyvät menetelmän käyttöön sekä sen räätälöintiin.

2.1 Menetelmästä yleisesti

Tässä alaluvussa kuvataan ensin menetelmän käsitteelle annettuja erilaisia määritelmiä, jonka jälkeen käydään lävitse menetelmän eri osia. Lopuksi tarkastellaan menetelmän erilaisia rooleja sekä sen käyttöön liittyviä hyötyjä ja haasteita.

2.1.1 Menetelmäkäsite

Menetelmällä (engl. method, methodology) tarkoitetaan yleisesti ottaen "tapaa tai joukkoa tapoja tehdä jotakin" (Webster, 1989). *Ohjelmistokehitys* (engl. software development, software engineering) on järjestelmällisen, kurinalaisen ja mitattavissa olevan lähestymistavan soveltamista ohjelmistojen kehittämiseen, käyttämiseen ja ylläpitoon (IEEE, 1990). Ohjelmistokehityksen yhteydessä menetelmälle on annettu erilaisia määritelmiä, joiden esittäjät ovat painottaneet määrittelyssään erilaisia rakenne- ja tarkoitusnäkökulmia. Roberts Jr., Gibson, Fields sekä Kelly Rainer Jr. (1998) toteavatkin, että määritelmiä menetelmälle on

olemassa niin paljon kuin erilaisia kehittämismenetelmiä on olemassa. Ne kuitenkin sisältävät paljon samoja piirteitä.

Iivari ym. (1998) määrittelevät tietojärjestelmän kehittämismenetelmän olevan organisoitu kokoelma käsitteitä, metodeja, uskomuksia, arvoja ja ohjeellisia periaatteita, joita materiaaliset resurssit tukevat. Tarkemmin menetelmä on tällöin nähty joukoksi johonkin tavoitteeseen tähtääviä toimintatapoja, jotka ohjaavat tietojärjestelmää rakentavien osapuolien työtä sekä yhteistyötä, ja näitä toimintatapoja tukee joukko suositeltuja tekniikoita, työkaluja sekä aktiviteetteja. Roberts Jr. ym. (1998) määrittelevät kehittämismenetelmän olevan kokonaisstrategia tietokonepohjaiselle järjestelmäkehittämiselle pitäen sisällään sarjan kehittämistehtäviä sekä tekniikoita, joilla kehittämistehtävät saadaan tehtyä. Tolvanen (1998) taas määrittelee kehittämismenetelmän olevan ennalta määritetty ja järjestetty kokoelma tekniikoita ja sääntöjä, jotka määrittelevät, millä tavalla, missä järjestyksessä ja kuka tekniikoita käyttää, jotta saavutetaan tietyt tavoitteet. Henderson-Sellers ja Ralyté (2010) määrittelevät menetelmän lähestymistavaksi ohjelmistoprojektin toteuttamiseen tietyllä ajatusmallilla, joka pitää sisällään muun muassa ohjeet, heuristiikat ja säännöt. Menetelmä on samalla systemaattisesti jäsennetty tiettyihin kehittämistehtäviin (engl. development activities), joihin liittyy vastaavat kehitystyön tulokset ja kehittäjäroolit, jotka kuuluvat ihmisille tai automatisoiduille työkaluille. (Henderson-Sellers & Ralyté, 2010.)

Avisonin ja Fitzgeraldin (2006) mukaan menetelmä tarkoittaa yksinkertaistettuna kokoelmaa käytänteitä, tekniikoita, työkaluja ja dokumentaation apuvälineitä, jotka auttavat kehittäjiä kehitteillä olevan uuden ohjelmiston kehittämisessä. Menetelmä pitää sisällään ohjelmiston kehittämisvaiheet sekä osavaiheet, joita ovat muun muassa esitutkimus (engl. feasibility study), vaatimusmäärittely sekä ohjelmiston sisäisen logiikan suunnittelu. Vaiheet ohjaavat kehittäjiä sopivien tekniikoiden valitsemisessa ja käyttämisessä. Näitä ovat esimerkiksi vaiheeseen ja kohdealueeseen soveltuvat kaaviot. Vaiheistaminen auttaa myös projektien suunnittelussa, johtamisessa, kontrolloinnissa ja arvioinnissa. Tämän lisäksi menetelmän ominaisuuksina voidaan pitää koulutussuunnitelmaa, jota sovelletaan tarvittaessa henkilöiden tullessa uusiin rooleihin organisaatioon, sekä menetelmän takana olevaa filosofista näkökulmaa. Tämän kaltainen, joskus implisiittisesti ilmaistu näkökulma voi olla esimerkiksi ihmiskeskeinen näkökulma tai mahdollisimman suureen automatisointiin pyrkivä näkökulma. (Avison & Fitzgerald, 2006.)

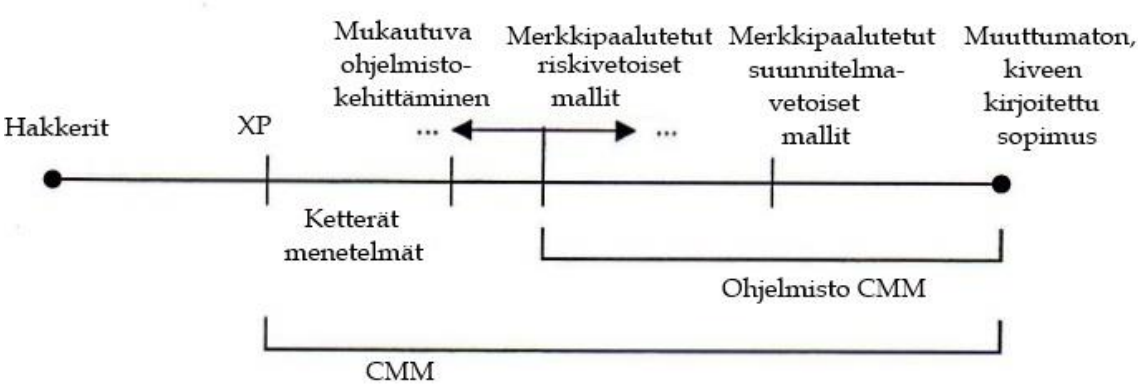
Tässä työssä noudetaan Avisonin ja Fitzgeraldin (2006) käsitystä menetelmästä. Tiivistetysti menetelmä voidaan määritellä seuraavasti: *Menetelmä* tarkoittaa ennalta määriteltyä joukkoa käsitteitä, tekniikoita, käytänteitä ja muita seikkoja, joita ohjelmistokehittäjät ja sidosryhmät voivat käyttää ohjelmiston kehittämisessä hyödyksi.

Vuosikymmenien aikana on kehitetty lukuisa määrä erilaisia menetelmiä. 1950-luvulla ja 1960-luvun alussa käytettiin paljon niin sanottua ”koodaa ja korjaa” – mallia, mutta liiallinen joustavuus teki kehitetyistä ohjelmistoista liian vaikeita testata ja ylläpitää (Yeh, 1991). Pääpaino oli ohjelmoinnilla ja teknisten

ongelmien ratkaisemisella, etenkin niiden ongelmien, jotka johtuivat kalliista ja rajallisesta laitteistosta (Avison & Fitzgerald, 2003). Ohjelmat olivat kalliita kehittää, kehitystyö kesti pitkään eivätkä ne toimineet kovin hyvin. Fitzgeraldin (1996) mukaan jo 1970-luvun alussa Langefors (1973) argumentoi formalisoidumman, tieteellisemmän ja rationaalisemman prosessin puolesta järjestelmäkehityksessä, jossa on muun muassa omissa prosesseissa määrittely mitä järjestelmä tekee ja miten se sen tekee. Vuonna 1970 esitelty vesiputousmalli onkin yksi viitatuimmista ohjelmistokehitysmalleista (Royce, 1970). Ensimmäiset menetelmät saivat paljon vaikutteita matemaattisista ja insinööritieteistä. Ne noudattivat pääasiassa lineaarista mallia, jossa eri vaiheet, kuten suunnittelu ja toteutus, seurasivat toisiaan. Näissäkin menetelmissä oli kuitenkin jo jonkin verran iteratiivisuutta. (Fitzgerald, 1996.) Nämä perinteiset menetelmät ovat usein korostaneet laajaa suunnittelua, strukturoituja prosesseja ja täsmällistä uudelleenkäyttöä, ollen enemmän suunnitelmavetoisia menetelmiä. 2000-luvulla yleistyneitä ketteriä menetelmiä taas voidaan pitää enemmän muutos-vetoisina menetelminä. (Boehm, 2002.)

Yksikään menetelmä ei ole maailmanlaajuisesti käytössä tai maailmanlaajuisesti hyödyllinen. Mikään menetelmä ei sovi kaikkialle muun muassa siksi, että organisaatioilla on erilaisia markkinoita ja arvoketjuja, projekteilla erilaisia budjetteja, riskejä, aikatauluja ja kokoja sekä tiimeillä erilaisia taitoja, kyvykkyyksiä ja kokemusta (Anderson, 2010). Jokainen ohjelmistoprojekti tarvitsee kuitenkin jonkinlaista vaatimusten keräystä, jonkinlaista suunnittelua, jonkinlaista kehitystä tai ohjelmointia sekä jonkinlaista virheiden korjausta (Jones, 2007).

Menetelmiä voidaan luokitella eri tavoin. Eräs luokittelu perustuu siihen, miten tarkkaan määriteltyjä ohjeistoja menetelmät sisältävät ja miten tarkkaan niitä oletetaan seurattavan. Toinen luokittelu on jakaa menetelmät formaaleihin ja ei-formaaleihin menetelmiin. Tässä luokittelussa ei-formaalit menetelmät nähdään käytännön kehittämistyön ad hoc – soveltamisena, kun taas formaalit menetelmät ovat hyvin tarkasti määriteltyjä, nimettyjä ja julkaistuja menetelmiä. (Fitzgerald, 1996.) Menetelmiä voidaan luokitella myös sen mukaan, miten paljon etukäteissuunnittelua edellytetään ennen ohjelmistojen koodaamista. Tällaisen luokituksen on suunnitteluspektrin muodossa esittänyt Boehm (2002) (kuvio 1).



KUVIO 1 Suunnitteluspektri (Boehm, 2002, 65)

Suunnitteluspektrissä vasemmassa reunassa ovat niin kutsutut hakkerit, jotka eivät tee mitään suunnitelmia. Oikeassa laidassa taas ovat ne menetelmät, joissa edellytetään ”kiveenhakattuja” sopimuksia vaatimuksiksi ennen kuin niitä lähdetään edes toteuttamaan. Aiemmin mainitut perinteiset tietojärjestelmien kehittämismenetelmät ovat olleet etukäteissuunnittelultaan kuvion oikeassa laidassa, mutta ketterien menetelmien käytön myötä myös kuvion vasemmassa laidassa olevaa, etukäteissuunnittelun suhteen kevyempää tapaa (esim. ketterät menetelmät ja mukautuvat ohjelmistokehittäminen) on käytetty entistä enemmän. (Boehm, 2002.)

2.1.2 Menetelmä osista muodostuvana kokonaisuutena

Ohjelmistojen kehittämismenetelmään kuuluu erilaisia osia, jotka tukevat toisiaan. Seuraavassa esitellään tunnetuimpia rakenteellisia kuvauksia menetelmästä ja sen jälkeen tämän tutkimuksen näkemys.

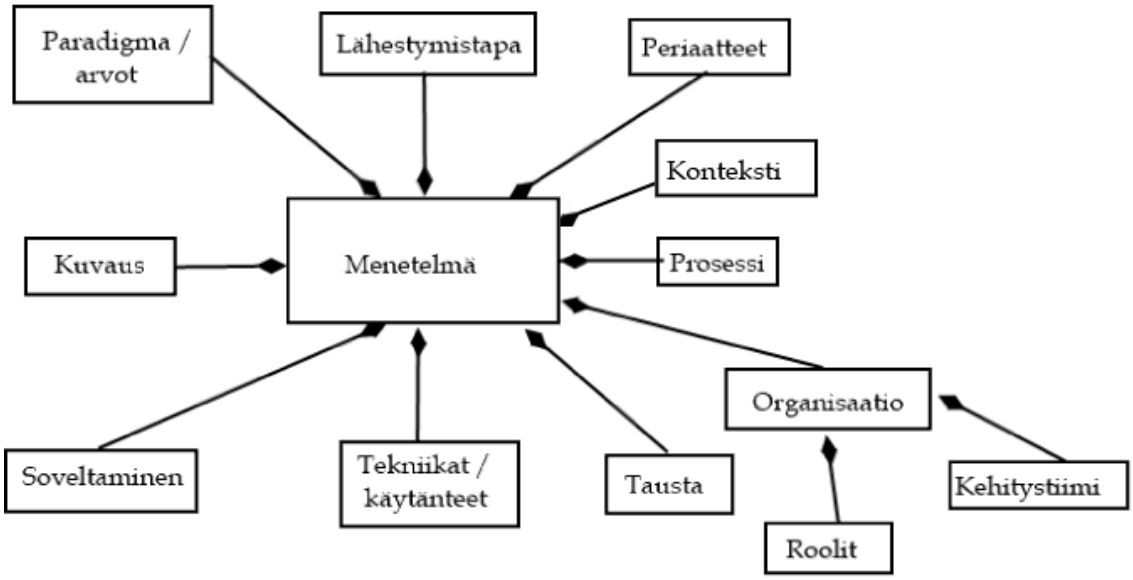
Heym ja Österle (1992) määrittelevät menetelmään kuuluviksi osiksi tekniikat, prosessit, aktorit, toimitettavat osat (deliverables), merkkipaalut (milestones) ja resurssit. Prosessin on nähty tällöin olevan aktiviteetti tai vaihe. Vaihe koostuu useammasta osavaiheesta tai aktiviteetista. Aktiviteetti on niin pieni osa työtä, että sitä ei enää jaeta osa-aktiviteetteihin. Toimitettava osa on tällaisesta vaiheesta tai aktiviteetista tuleva lopputulos, kuten ohjelmakoodin osa. Merkkipaalut vastaavat tiettyä vaihetta prosessista, jolloin toimitettavien osien täytyy olla tietyn tasoisia. Aktori on henkilö, ryhmä henkilöitä tai organisaatioyksikkö, joka on vastuussa esimerkiksi yhdestä aktiviteetista. Tekniikalla tarkoitetaan tarkkaa kuvausta kehittämisestä, jota käytetään jonkin tietyn toimitettavan osan kehittämiseksi. Resurssit ovat prosessissa vaadittuja ei-inhimillisiä välineitä. Näiden lisäksi Heym ja Österle (1992) sisällyttävät menetelmään erilaiset ohjeet, joita kehittäjät tarvitsevat menetelmää soveltaessa. Nämä ohjeet ovat dynaamisempi osa menetelmää. (Heym & Österle, 1992.)

Henderson-Sellers ja Ralyté (2010) esittelevät tilannekohtaisesti sovellettavan menetelmäkehityksen mallin. Siinä menetelmä voi sisältää prosessia koskevia piirteitä. Menetelmä voi olla kiinteä (fixed) menetelmä, räätälöity (tailored) menetelmä tai konstruoitu (constructed) menetelmä. Konstruoitu menetelmä koostuu joko menetelmäpaloista (fragment) tai menetelmälohkoista (chunk). Menetelmälohko koostuu useammasta menetelmäpalasta. Menetelmäpalaan liittyy ohje (guideline), joka voi olla strateginen, taktinen tai yksinkertainen. Menetelmäpalat pidetään esimerkiksi organisaation omassa tietokannassa, jolloin niistä voidaan tarvittaessa koostaa oma menetelmäkokonaisuus. Menetelmään on tällöin kolme toisistaan riippuvaa näkökulmaa: tuote-, prosessi- ja ihmisnäkökulma. (Henderson-Sellers & Ralyté, 2010.)

Leppänen (2005) on esitellyt tietojärjestelmän kehittämismenetelmäontologian, jolla voi järjestelmällisesti ja yksityiskohtaisesti määritellä niin kehittämismenetelmän luonne, sisältö kuin myös sen rakenne. Menetelmäontologia on jaettu seitsemään eri näkymään. Ontologia koostuu käsitteistä ja rakenteista, joiden avulla sen asiayhteyteen liittyvät näkökulmat voidaan ymmärtää, käsit-

tää, rakenteistaa ja esittää. Historiallinen näkymä valaisee menetelmän taustaa ja kokemuksia menetelmän kehittämisestä sekä menetelmän käytöstä. Soveltamisnäkymä esittää, missä ja miten menetelmää voidaan soveltaa. Tällöin menetelmä pitää sisällään kuvaukset tarkoitetuista ohjelmistonkehittämiskonteksteista, johon menetelmä on tarkoitettu, sekä tarkoitetut menetelmänkehittämiskontekstit, joissa menetelmä räätälöidään ja sitä käytetään. Geneerinen näkymä antaa yleiskuvan menetelmästä, korostaen sen takana olevia filosofisia arvoja ja oletuksia sekä ohjelmistonkehittämiskontekstissa noudatettavia lähestymistapoja ja periaatteita. Sisältönäkymä pitää sisällään menetelmän käsitteellisen sisällön. Esitysnäkymä taas näkee menetelmän joukkona ilmaisuja, jotka esitetään joidenkin kielien avulla. Fyysinen näkymä pitää sisällään ne esitystavat, joilla menetelmästä tehdään näkyvä ja toimiva, kuten esimerkiksi erilaiset ohjekirjat tai internet. Viimeinen näkymä, eli rakenteellinen näkymä, on modulaarinen kokonaisuus, joka koostuu esimerkiksi ohjelmistokehityksen oletuksista, periaatteista, tekniikoista, malleista sekä säännöistä. Joitakin rakenteellisen näkymän osista voidaan pitää menetelmän komponentteina. (Leppänen, 2005.)

Edellä esitetyt rakenteelliset kuvaukset sisältävät jossain määrin samankaltaisia piirteitä. Esimerkiksi Heymin ja Österlen (1992) määrittelemä aktiviteetti voi olla osaltaan Henderson-Sellersin ja Ralytén (2010) määrittelemä yksittäinen menetelmäpala. Leppänen (2005) on määritellyt menetelmän laajemmin ja on ottanut huomioon erilaiset näkökulmat. Tässä tutkielmassa menetelmän on nähty koostuvan osista, jotka on esitelty kuviossa 2. Piirtämisessä on sovellettu UML-kielen notaatioon (Booch, Rumbaugh & Jacobson, 1999) kuuluvaa koostesuhdetta (composition).



KUVIO 2 Menetelmä osista koostuvina kokonaisuutena

Kuviossa menetelmä on keskiössä. Se pitää sisällään tietyn paradigman ja arvot sekä lähestymistavan, jollaisena menetelmässä nähdään ohjelmistokehitys. Menetelmän taustalla on myös erilaisia periaatteita, ja sillä on olemassa jokin kon-

teksti, jossa sitä sovelletaan. Tämä voi olla esimerkiksi menetelmän kehittämisen tai ohjelmistokehityksen konteksti.

Menetelmä voidaan kuvata erilaisin tavoin, kuten esimerkiksi ilmaisemalla sen sisältö kirjallisesti. Erilaiset toimijat, kuten organisaatiot, kehitystiimit ja menetelmäkehittäjät, voivat käyttää menetelmää hyväkseen. Organisaation jäsenillä voi olla erilaisia rooleja, kuten kehittäjiä tai projektipäälliköitä, ja erilaiset roolit usein käyttävät menetelmää eri tavoin. Menetelmässä on usein erilaisia tekniikoita ja käytänteitä, kuten erilaisia palavereita tai ohjelmointikäytänteitä. Menetelmä pitää sisällään prosessin, jota ohjelmistokehitys noudattaa. Prosessin eri vaiheissa eri tekniikoita ja käytänteitä voidaan käyttää hyväksi. Käytännön työssä menetelmän käyttäminen on sen soveltamista käytännössä.

Menetelmän voi määritellä osista koostuvina kokonaisuutena seuraavalla tavalla. *Menetelmä* on ennalta määritelty, tietyn taustan, lähestymistavan ja periaatteita omaava kokonaisuus, jonka käyttämisen tavoitteena on parantaa ohjelmistokehitystä. Menetelmä määrittelee prosessin, jonka eri vaiheissa käytetään eri tekniikoita ja käytänteitä. Menetelmää käyttävät erilaiset henkilöt ja organisaatiot tietyssä ainutlaatuisessa kehittämiskontekstissa, ja eri menetelmät soveltuvat parhaiten eri konteksteihin.

2.1.3 Menetelmän hyötyjä, rooleja ja haasteita

Menetelmän käytöllä on havaittu olevan useita hyötyjä. Sen on havaittu parantavan ohjelmistokehityksen prosessia, vähentävän rahan, ajan ja työvoiman tarvetta sekä parantavan myös mahdollisesti itse kehitystyön lopputuloksen laatua (Leppänen, 2005). Tolvanen (1998) sanoo menetelmän käytön parantavan dokumentaatiota, systematisoivan ohjelmistokehityksen prosessia, parantavan mahdollisuuksia saada kehityksen lopputuloksen vastaamaan vaatimuksiaan sekä lisäävän käyttäjien osallistumista kehitystyöhön. Tämän lisäksi menetelmien on huomattu lisäävään kontrollia projekteista ja lisäävän sen ennustettavuutta sekä tuovan yhteisen pohjan keskustelulle ja ymmärrykselle. Menetelmä auttaa lisäksi uusia työntekijöitä tunnistamaan, mistä heidän tulee tietää enemmän, ja helpottaa keskusteluja kokeneempien työntekijöiden kanssa ja tukevan tietämyksen kokoamista ja jakamista tarjoamalla muun muassa yhteisen terminologian ja työnkulut keskustelujen pohjaksi. (Schönström & Carlsson, 2003.)

Edellä mainitut hyödyt liittyvät menetelmän ns. rationaaliseen rooliin (Fitzgerald, Russo & Stolterman, 2002). Mutta menetelmällä on myös poliittinen rooli. Siihen liittyvinä höytyinä voidaan mainita seuraavia (Fitzgerald ym., 2002, 104-106): menetelmä voi auttaa ammatillistamaan tietojärjestelmätyötä, parantamaan kehittämistyöstä vastaavan osaston asemaa yrityksessä, lisäämään luottamusta lopputuloksen laatuun (comfort or confidence factor), jäljittämään sen kohdan kehittämisprosessia, jossa mahdollinen väärä päätös tehtiin, sekä tarjoamaan määrämuotoisen perustan kehittämistä koskevan sopimuksen tekemiselle ja tulkinnalle oikeudellisen perustan. Menetelmän käyttö johtaa usein organisaatiossa ns. menetelmäekspertin (method champion) syntymiseen, jolla

voi olla muodolliseen asemaansa verrattuna suurempi valta. Chang, Tung-ching ja Sheng (2002) tutkivat 56 tapaustutkimusta ohjelmistonkehitysprosesseista, joista he löysivät yhteensä 192 poliittista peliä, jotka voitiin jakaa 41 eri kategoriaan. Esimerkkeinä poliittisista peleistä voivat olla etäisenä pysyminen, jolloin pyritään välttämään vastuuta, varman päälle pelaaminen, jolloin pyritään välttämään epävarmuutta ja osallistutaan mieluummin varmoihin projekteihin, sekä koalitioiden muodostaminen niin työntekijöiden kuin myös muiden asianomaisten kanssa. (Cheng, Tung-ching & Sheng, 2002.)

Menetelmän käyttö on joissain tapauksissa nähty jopa välttämättömäksi. Ramamoorthy, Garg ja Prakash (1986) toteavat, että ainoastaan kurinalainen ja menetelmää käyttävä kehitystyö voi saada ison ohjelmistoprojektin onnistumaan. Palvia ja Nosek (1993) taas huomasivat, että heidän havaintonsa menetelmän käytön vähyydestä on pelottava ja häiritsevä. Fitzgerald (1996) kuitenkin toteaa, että ihmiset kehittävät ohjelmistoja, ei menetelmät, ja että menetelmät ainoastaan voivat parhaimmillaan tarjota hyödyllisen viitekehyksen kehittämiselle. Kokeneet kehittäjät voivatkin ymmärtää menetelmien rajoitteet, ja jättää ne käyttämättä, jos niistä ei nähdä olevan hyötyä. (Fitzgerald, 1996.)

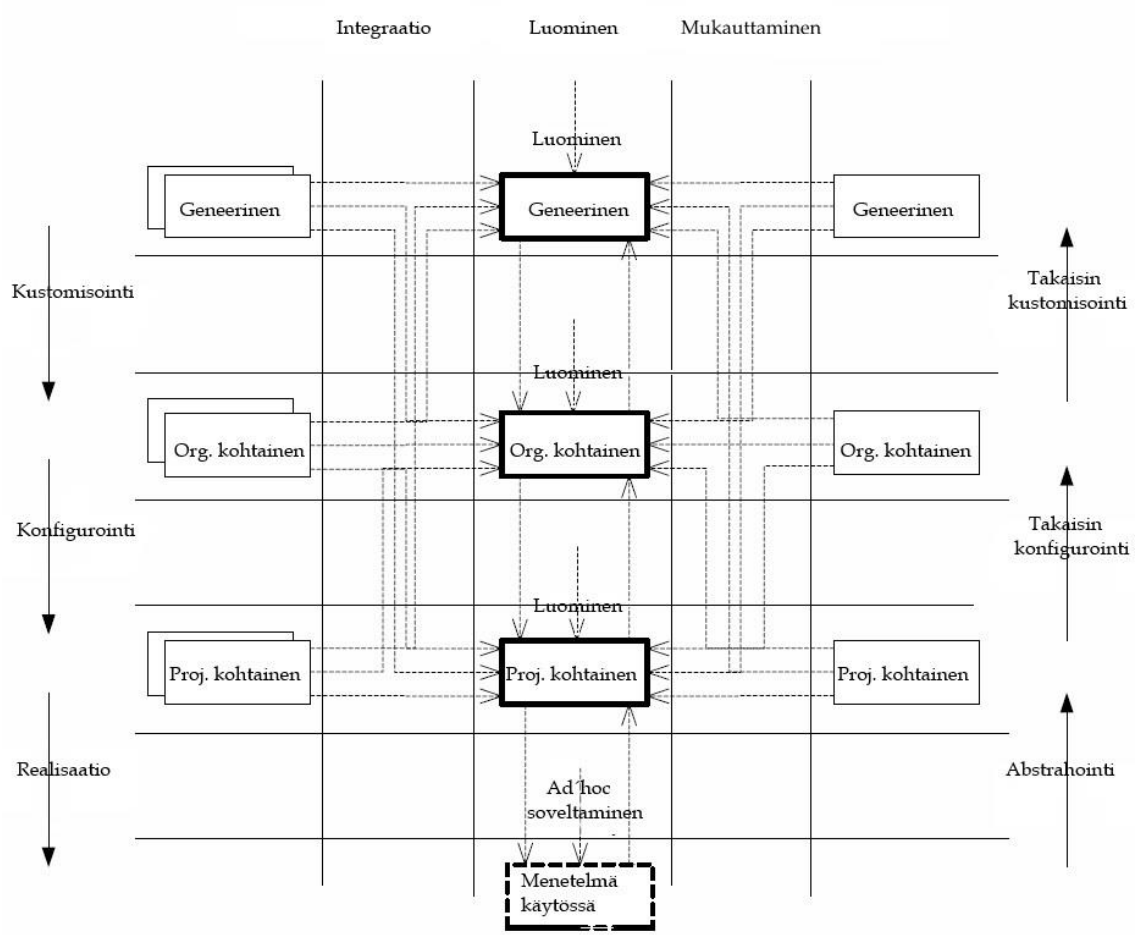
Menetelmän käytöllä on omat haasteensa, eikä niiden käyttö ohjelmistokehityksessä takaa millään tavalla projektin onnistumista. Ne on koettu muun muassa sopiviksi vain suuriin ja monimutkaisiin projekteihin, ja ne voivat olla joustamattomia tai vastata huonosti tilannetekijöihin, jotka liittyvät käsillä olevaan organisaatioon, projektiin tai teknologiaan. Ne eivät myöskään aina tuo luvattuja parannuksia tuottavuuteen, ja niiden taustalla voi olla yksinkertaistavia oletuksia esimerkiksi siitä, että käyttäjät ovat tietoisia ohjelmiston vaatimuksista, mikä ei käytännössä aina pidä paikkaansa. (Avison & Fitzgerald, 2003.) Menetelmän työkalut voivat olla myös kalliita ja vaatia erityisiä teknisiä taitoja (Leppänen, 2005). Menetelmää on käytetty myös sosiaalisena puolustuksena epämiellyttäviä asioita vastaan, jolloin se pahimmillaan estää oppimista ja luovaa ajattelua (Wastell, 1996).

2.2 Menetelmäkehitys

Seuraavaksi tarkastellaan sitä, miten menetelmiä kehitetään yleisellä tasolla ja tietyn organisaation tai projektin käyttöön. Brinkkemper (1996) määrittelee *menetelmän kehittämisen* (engl. *method engineering*) tarkoittavan työtä, jonka tarkoituksena on suunnitella, rakentaa ja soveltaa menetelmiä, tekniikoita ja työkaluja tietojärjestelmien kehittämistä varten. Tolvanen (1998) sanoo menetelmäkehityksen idean olevan samankaltainen kuin esimerkiksi järjestelmäkehityksen. Kun järjestelmäkehityksessä kehitetään ja ylläpidetään järjestelmiä tukemaan liiketoimintaprosesseja, menetelmäkehityksessä kehitetään ja ylläpidetään menetelmiä tukemaan järjestelmäkehitystä. (Tolvanen, 1998.) Leppänen (2005) määrittelee menetelmän kehittämisen seuraavalla tavalla:

> Menetelmän kehittäminen tarkoittaa kaikkia niitä aktiviteetteja, joiden avulla tietojärjestelmän kehittämismenetelmä on kehitetty, ja mahdollisesti myöhemmin räätälöity ja konfiguroitu vastaamaan organisaation ja/tai projektin tarpeita. Leppänen (2005, 442)

Leppänen on esittänyt menetelmän kehittämiselle viitekehyksen (kuvio 3). Viitekehyksessä menetelmän kehittämiselle on määritelty kolme erillistä strategiaa, kolme erillistä kontekstia sekä kuusi erillistä prosessia, joiden avulla voidaan selventää menetelmän kehittämisen periaatteita ja tilanteita. Strategia tarkoittaa menetelmän kehittämisen yhteydessä yleistä keinoa suorittaa jokin tietty suunnittelun pyrkimys. Strategioista ensimmäinen, *luominen*, tarkoittaa tietojärjestelmän kehittämismenetelmän suunnittelua ilman aiempien menetelmien käyttämistä pohjana. *Integraatio* tarkoittaa menetelmän kehittämistä kokoamalla komponentteja tai palasia jo olemassa olevista menetelmistä. *Mukauttaminen* taas tarkoittaa olemassa olevan menetelmän komponenttien muokkaamista tai poisjättämistä, tai menetelmän laajentamista uusilla osilla. (Leppänen, 2005.)



KUVIO 3 Menetelmän kehittämisen viitekehys (Leppänen, 2005, 439)

Menetelmän kehittämisen prosesseilla viitataan prosesseihin, joiden avulla voidaan johtaa menetelmä joko seuraavalle tai edelliselle tasolle. *Kustomisointi (*customization*)* tarkoittaa organisaatiokohtaisen menetelmän johtamista joko

ylemmästä geneerisestä tai sovellusaluekohtaisesta menetelmästä, muuttaen sitä vastaamaan organisaation kulttuuria, tapoja, rakenteita, johtotapoja ym. haluttuja piirteitä. *Konfiguroinnissa* (configuration) taas johdetaan tietylle projektille erityinen menetelmä organisaatiokohtaisesta menetelmästä, jolloin sille konkreettisesti suunnitellen johdetaan muun muassa, ketkä tekevät mitä, missä ja milloin. *Toteutus* (realization) tarkoittaa tämän menetelmän ottamista käyttöön. (Leppänen, 2005.)

Kolmessa muussa prosessissa, eli *abstrahoinnissa* (abstraction), *takaisin konfiguroinnoissa* (re–configuration) ja *takaisin kustomisoinnissa* (re–customization) tarkoitetaan projekti- ja organisaatiospesifististen piirteiden suodattamista (Leppänen, 2005).

Menetelmän kehittämisen kontekstit vaihtelevat riippuen kehittämistavoitteista. *Menetelmän kehittämisen kontekstissa* pyritään suunnittelemaan joko geneeristä tai sovellusaluekohtaista menetelmää. Kustomisointikontekstissa taas pyritään tuottamaan organisaatiolle organisaatiokohtainen menetelmä, jolloin se tapahtuu organisaation sisällä esimerkiksi silloin, kun organisaatio haluaa ottaa käyttöön uudet menetelmät. Konfigurointikontekstissa taas johdetaan projektikohtaista menetelmää usein organisaation omista käytössä olevista menetelmistä, mutta joskus se johdetaan suoraan tarjolla olevista geneerisistä menetelmistä. (Leppänen, 2005.) Tässä työssä esitettyä konfigurointia ja kustomisointia kutsutaan yhdessä *menetelmän räätälöinniksi*.

Konkreettisemman kuvan menetelmäkehityksestä tarjoaa tilannekohtaisen menetelmäkehityksen lähestymistapa (situational method engineering) (Kumar & Welke, 1992). Tilannekohtaisessa menetelmäkehityksessä rakennetaan projektikohtaiset menetelmät olemassa olevien menetelmien osista. Tätä tekniikkaa kutsutaan menetelmän kokoamiseksi. (Brinkkemper, Saeki & Harmsen, 1999.) Tarkoituksena on tällöin yhdistää eri menetelmien vahvuudet uudeksi menetelmäksi (Karlsson & Åkerfalk, 2004). Henderson-Sellersin ja Ralytén (2010) mukaan tilannekohtainen menetelmäkehityksen lähestymistapa pitää sisällään kolmella tavalla luotuja menetelmiä: kiinteitä (fixed), räätälöityjä (tailored) tai konstruoituja (constructed). Näistä konstruoidut menetelmät noudattavat tilannekohtaista menetelmäkehitystä, jolloin menetelmä sovitetaan tiettyyn tilanteeseen ja usein tiettyihin projektin ominaispiirteisiin. Organisaatio pitää menetelmätietokannassaan menetelmäpaloja ja menetelmälohkoja, joita käyttämällä voidaan luoda uusia menetelmiä. Nämä menetelmät ovat ainutlaatuisia ja niiden omistus pysyy organisaatiolla. Tilannekohtaista menetelmäkehitystä käyttämällä organisaation tulee luoda menetelmäpaloja tiettyihin olosuhteisiin, ylläpitää niistä tietokantaa ja myös koostaa kokonaiset menetelmät tarvittaessa. (Henderson-Sellers & Ralyté, 2010.) Harmsen, Brinkkemper ja Oei (1992) jakavat menetelmäosat prosessiosiin sekä tuoteosiin. Prosessiosat pitävät sisällään tehtäviä, aktiviteetteja ja vaiheita, joiden avulla luodaan tuoteosia, kuten malleja, diagrammeja sekä erilaisia toimitettavia osia (deliverables).

2.3 Menetelmän räätälöinti

Menetelmän räätälöinnistä käytetään kirjallisuudessa monenlaisia nimityksiä. Englanninkielisessä kirjallisuudessa siihen viitataan muiden muassa termeillä ”method tailoring”, ”method customisation”, ”method adaptation”, ”method configuration”, ”situational or situated method engineering”, ”context-specific method engineering”, ”method fragment adaptation” sekä ”method engineering”, joskin nimityksien määrittelyt voivat vaihdella. Menetelmän räätälöintiä voi tapahtua niin organisaatiotasolla kuin myös projektitasolla. Räätälöintiä on suoritettu käytännössä jo ennen kuin tutkimuskirjallisuus on huomannut sen merkityksen (Patel, de Casare, Iacovelli ja Merico, 2004). Conboyn ja Fitzgeraldin (2010) mukaan jo 1980–luvulla DeMarco (1982) määritteli, että menetelmää tulisi käyttää ainoastaan lähtöpisteenä kehitystyölle ja sille tulisi suorittaa räätälöintiä, eikä menetelmän tarkoituksena ole olla vain joustamaton kasa sääntöjä. Räätälöinnille on huomattu käytännön tarve keskeisten tilannetekijöiden, kuten sovellusalueen tarpeiden, kehittäjien osaamisen tai ohjelmiston laajuuden, mukaan jo pitkään, niin käytännössä kuin tutkimuksissa. Backlund ym. (2003) toteavat, että menetelmä tulee ensiksi sopeuttaa ja räätälöidä organisaatioon sopivaksi, mikäli siitä halutaan tehdä osa organisaation omaa tietämyspohjaa.

Menetelmän räätälöinnille on kirjallisuudessa esitetty useamman kaltaisia määritelmiä. Aydin ym. (2005) määrittelevät menetelmän räätälöinnin prosessiksi tai kyvykkyydeksi, jossa toimijat määrittävät projektikohtaisen lähestymistavan järjestelmän kehittämiseen. Tämä tapahtuu tekemällä reagoivia muutoksia konteksteihin, aikomuksiin ja menetelmän osiin. (Aydin ym., 2005.) Pedreira, Piattini, Luaces ja Brisaboa (2007) määrittelevät ohjelmistoprosessin räätälöinnin seuraavasti:

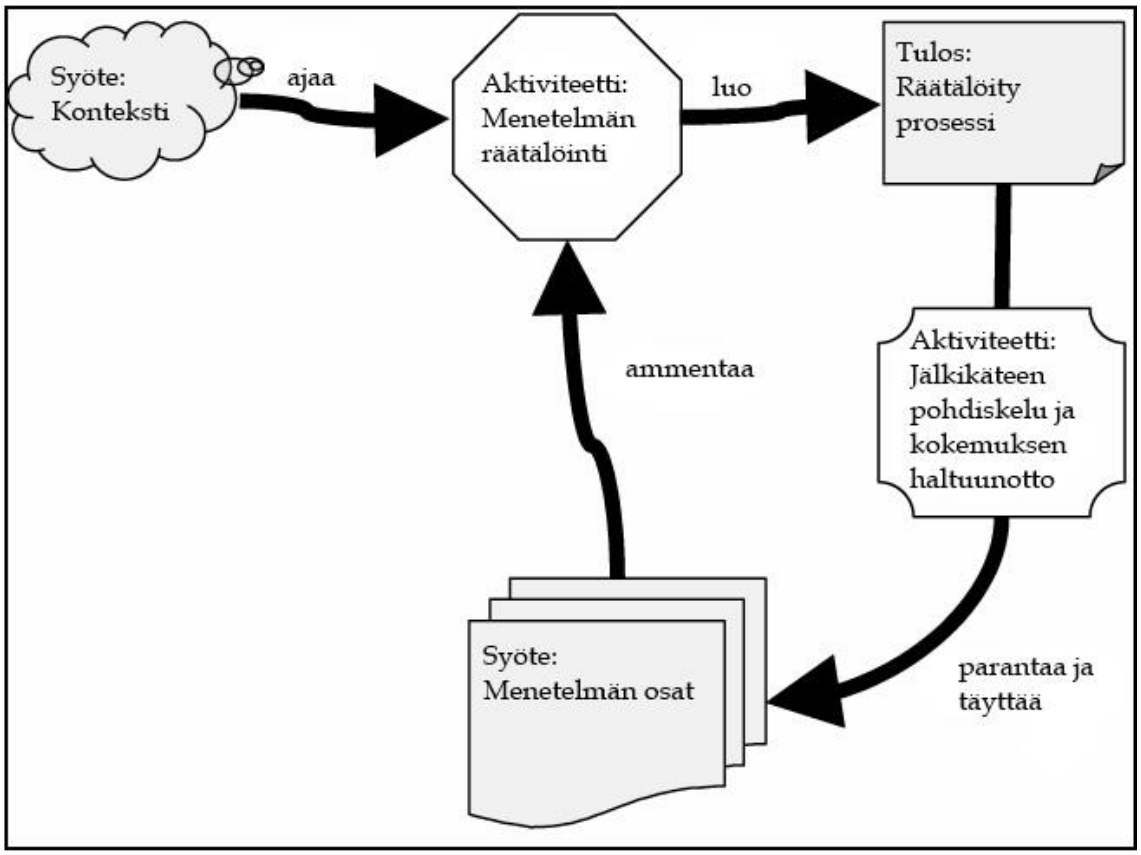
> Ohjelmistoprosessin räätälöinti on yleisen prosessikuvauksen määritelmien säätämistä ja/tai käsitteiden tarkentamista, jolla johdetaan uusi prosessi soveltumaan vaihtoehtoiseen (ja luultavasti vähemmän yleiseen) ympäristöön. Toisin sanoen se on standardoidun ohjelmistoprosessin mukauttamista kohtaamaan tietyn organisaation tai projektin tarpeet. Ohjelmistoprosessin räätälöintiä voi tapahtua organisaatiotasolla ja projektitasolla (Pedreira ym., 2007, 1).

Fitzgerald ym. (2002, 147) määrittelevät menetelmän räätälöinnin kontingenssitekijöiden suhteen, käytännön kehittäjien soveltaessa menetelmää harvoin virallisesti dokumentoidulla tavalla. Karlsson ja Ågerfalk (2004) taas määrittelevät räätälöinnin tietyn menetelmän sopeuttamisena erilaisten tilannetekijöiden mukaan, jossa pääpaino on yhden tietyn menetelmän käyttämisestä pohjana useiden menetelmien sijaan. Patel ym. (2004) määrittelevät räätälöinnin taas yhden metodologisen viitekehyksen valitsemiseksi ja mukauttamiseksi tiettyyn kehittämisprojektiin. Räätälöinnin onnistumiselle on tärkeää kyseisen menetelmän tai viitekehyksen joustavuus tilanteen mukaan. (Patel ym., 2004.) Tehokkaan menetelmän avainominaisuutena voidaankin pitää sitä, että sitä voidaan myös tehokkaasti räätälöidä (Conboy & Fitzgerald, 2010.) Karlsson ja

Ågerfalk (2009) toteavat, että tiettyä menetelmää räätälöidessä tulisi luoda menetelmä, joka ottaa tilanteen huomioon, mutta on ”samalla viivalla” alkuperäisen menetelmän kanssa. Menetelmän räätälöinti on nähty perinteisesti teknisenä ongelmana, mutta siinä tulisi huomioida laajemmin myös muitakin tilannetekijöitä, kuten liiketoimintaan vaikuttavia seikkoja (Sauer & Lau, 1997).

Myös tilannekohtaisessa menetelmäkehityksessä luotuja menetelmiä voidaan räätälöidä, jolloin räätälöinti on yksi tilannekohtaisen menetelmäkehityksen muoto (Karlsson & Ågerfalk, 2004). Räätälöinnillä on pohjamenetelmä (base method), joka voi koostua kokonaisesta menetelmästä tai joukosta menetelmäosia. Räätälöinnin aikana kyseistä menetelmää voidaan täydentää lisäosilla muista menetelmistä. (Henderson-Sellers & Ralyté, 2010.)

Menetelmän räätälöinnistä on kirjallisuudessa tehty erilaisia viitekehyksiä ja kattavampaan kuvaan tähtääviä tutkimuksia. Patel ym. (2004) ovat tutkineet useita menetelmiin liittyviä tutkimuksia (Harmsen, Brinkkemper & Oei, 1994; Baskerville & Stage, 2001; Henninger, Ivaturi, Nuli & Thirunavukkaras, 2002; Fitzgerald, Russo ja Stolterman, 2002), joiden perusteella he ovat rakentaneet näiden pohjalta menetelmän räätälöinnin viitekehyksen (kuvio 4).



KUVIO 4 Menetelmän räätälöinnin viitekehys (Patel ym., 2004, 6)

Patel ym. (2004) vertailivat tutkimuksia toisiinsa, muun muassa sitä, mistä lähtökohdista tutkimukset ovat lähteneet liikkeelle sekä millaista terminologiaa ne käyttävät räätälöinnissä. Viitekehys koostuu kontekstista, menetelmän räätä-

löintiprosessista, menetelmäosista, räätälöidystä prosessista sekä kokemuksen haltuunotosta. *Kontekstilla* tarkoitetaan ympäristöä, josta räätälöintiprojekti lähtee liikkeelle ja joka dynaamisesti muuttuu sekä kehittyy. Fitzgerald ym. (2002) määrittelee tämän annetuksi, sellaiseksi mitä ei voi muuttaa. Se pitää sisällään niin toimittajan kuin asiakkaan organisaation, ja se otetaan menetelmän räätälöinnin lähtökohdaksi. Sillä on tässä viitekehyksessä neljä kategoriaa: organisaation ominaispiirteet, tiimin dynamiikka ja rakenne, projektin ominaispiirteet sekä tuotteen ominaispiirteet. Menetelmät räätälöidään vastaamaan tätä kontekstia, esimerkiksi valitsemalla sopivat menetelmäosat. Menetelmän tulee jossain määrin tukea tämän kaltaista modulaarisuutta, jotta räätälöintiä voidaan suorittaa.

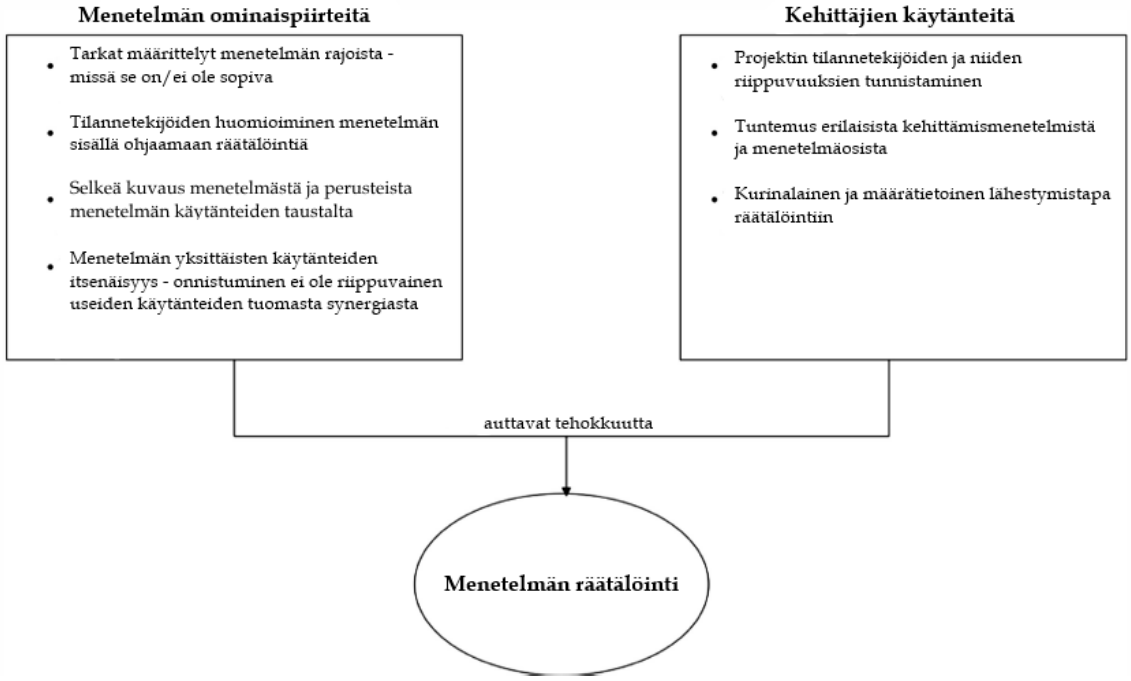
Menetelmän räätälöintiprosessi viittaa menetelmäosien valitsemis- ja kokoamisprosessiin, jonka avulla kehitetty menetelmä saadaan sopimaan kehittämiskontekstiin osien lisäämisellä, poistamisella sekä muuttamisella. *Räätälöity prosessi/menetelmä* on tämän prosessin lopputulema, jota käytetään itse kehittämisprojektissa tai organisaatiossa. Se pitää sisällään kaikki tarvittavat menetelmäosat. (Patel ym., 2004.)

Kokemuksen haltuunotto perustuu siihen havaintoon, että ihmiset perustavat menetelmien räätälöinnin aikaisempiin kokemuksiin. Tämä mahdollistaa organisaation oppimisen muun muassa menneistä onnistumisista ja epäonnistumisista. Näin menetelmän räätälöinnin kokemus ja sen haltuunotto tavalla, jolla siitä saa luotua uudelleenkäytettävää informaatiota, on tärkeää. Tällä tavalla esimerkiksi perustelut menetelmän tiettyjen osien käyttämiselle tai käyttämättä jättämiselle jakaa tietämystä niin tiimin jäsenille kuin myös muille organisaatiossa toimiville henkilöille. (Patel ym., 2004.)

Conboy ja Fitzgerald (2010) pyrkivät selvittämään, mitkä seikat menetelmissä ja toisaalta menetelmien käyttäjissä eli kehittäjissä edesauttavat menetelmien räätälöintiä (kuvio 5). Heidän mukaansa menetelmässä tulisi olla selkeästi sanottu, mihin tilanteisiin se sopii. Rajat kertovat kehittäjille suoraan, mihin tilanteisiin menetelmä ei sovi. Tilannetekijöiden huomioiminen menetelmän sisällä auttaa myös räätälöintiä, sillä tällöin kehittäjät voivat saada neuvoja siitä, miten menetelmää voisi räätälöidä jotta tietyt projektin olosuhteet täyttyvät. Menetelmän kuvaus ja käytänteiden takana oleva perusteet (rationale) tulisi kertoa mahdollisimman selkeästi, jotta kehitystiimi saa mahdollisimman paljon tietoa menetelmäosista ja syistä ottaa ne käyttöön menetelmää rakentaessaan. Mahdollisimman itsenäiset käytänteet ovat myös olennaisia räätälöinnin kannalta, sillä silloin ne voidaan ottaa käyttöön tai jättää käyttämättä ilman pelkoa tuntemattomista sivuvaikutuksista koko menetelmään kokonaisuutena. (Conboy & Fitzgerald, 2010.)

Myös kehittäjien ominaisuudet ovat tärkeässä roolissa menetelmää räätälöitäessä, sillä päävastuu räätälöinnistä on heillä. Tilannetekijät ovat keskeisessä osassa räätälöintiä, mutta kehittäjillä on vastuu eri tekijöiden sekä niiden välisten riippuvuuksien tunnistamisesta. Hyvin tunnistetut tilannetekijät parantavat mahdollisuutta onnistua räätälöinnissä. Kehittäjien olisi hyvä tuntea myös erilaisia kehittämismenetelmiä ja menetelmäosia. Tämä auttaa tilanteeseen sopi-

van menetelmän valitsemisessa sekä tarvittaessa menetelmää täydentämisessä osia lisäämällä muista menetelmistä tai tarvittaessa kokonaan uuden menetelmän tekemisessä. (Conboy & Fitzgerald, 2010.)



KUVIO 5 Viitekehys ominaispiirteistä, jotka helpottavat menetelmän räätälöintiä (Conboy & Fitzgerald, 2010, 216)

Viimeisenä ominaispiirteenä kehittäjien tulisi soveltaa kurinalaista ja määrätietoista lähestymistapaa räätälöintiin. Usein räätälöinti tapahtuu ad hoc -tavalla, eikä kokemusta räätälöinnistä tai sen lopputuloksesta oteta hyötykäyttöön tuleviin projekteihin. Yksi keino hallitusti johtaa menetelmän räätälöintiä on luokitella menetelmien osia ominaisuuksien mukaisesti tai pitää arvot ja tavoitteet esillä sekä tehdä menetelmän, jonka rakenne on niiden mukainen. (Conboy & Fitzgerald, 2010.)

2.4 Tilannetekijät

Kontekstia pyritään kuvailemaan tiettyjen ominaispiirteiden mukaisesti. Näitä piirteitä kutsutaan *kontingenssitekijöiksi* (engl. contingency factors) tai *tilannetekijöiksi* (engl. situational factors). Ohjelmistoprosessin ja menetelmän tärkeä vaatimus on, että sen ominaisuudet sopivat kyseisen projektin tarpeisiin (Feiler & Humphrey, 1992). Subramanian, Klein, Jiang sekä Chan (2009) toteavat, että kehittämislähestymistavan tulisi parhaiten sopia organisaation ja markkinoiden tavoitteisiin, tuotteeseen, lahjakkuuteen ja olosuhteisiin. Bekkers, van de Weerd, Brinkkemper ja Mahieu (2008) määrittelevät tilannetekijän miksi tahansa tekijäksi, joka on olennainen tuotteen kehityksen ja palveluiden kannalta. Van Sloo-

ten ym. (1996) näkevät kontingenssitekijät projektin olosuhteina, jotka järjestelmäkehityksessä jollakin tapaa vaikuttavat valittavaan tai rakennettavaan lähestymistapaan. Kontingenssitekijät ohjaavat menetelmän ja menetelmäosien valintaa menetelmäportfoliosta tilanteeseen sopivan menetelmän aikaansaamiseksi. Myös Benediktssonin, Dalcherin ja Thorbergssonin (2006) mukaan sopivimman menetelmän valitseminen on riippuvainen kontekstista ja osallistujista.

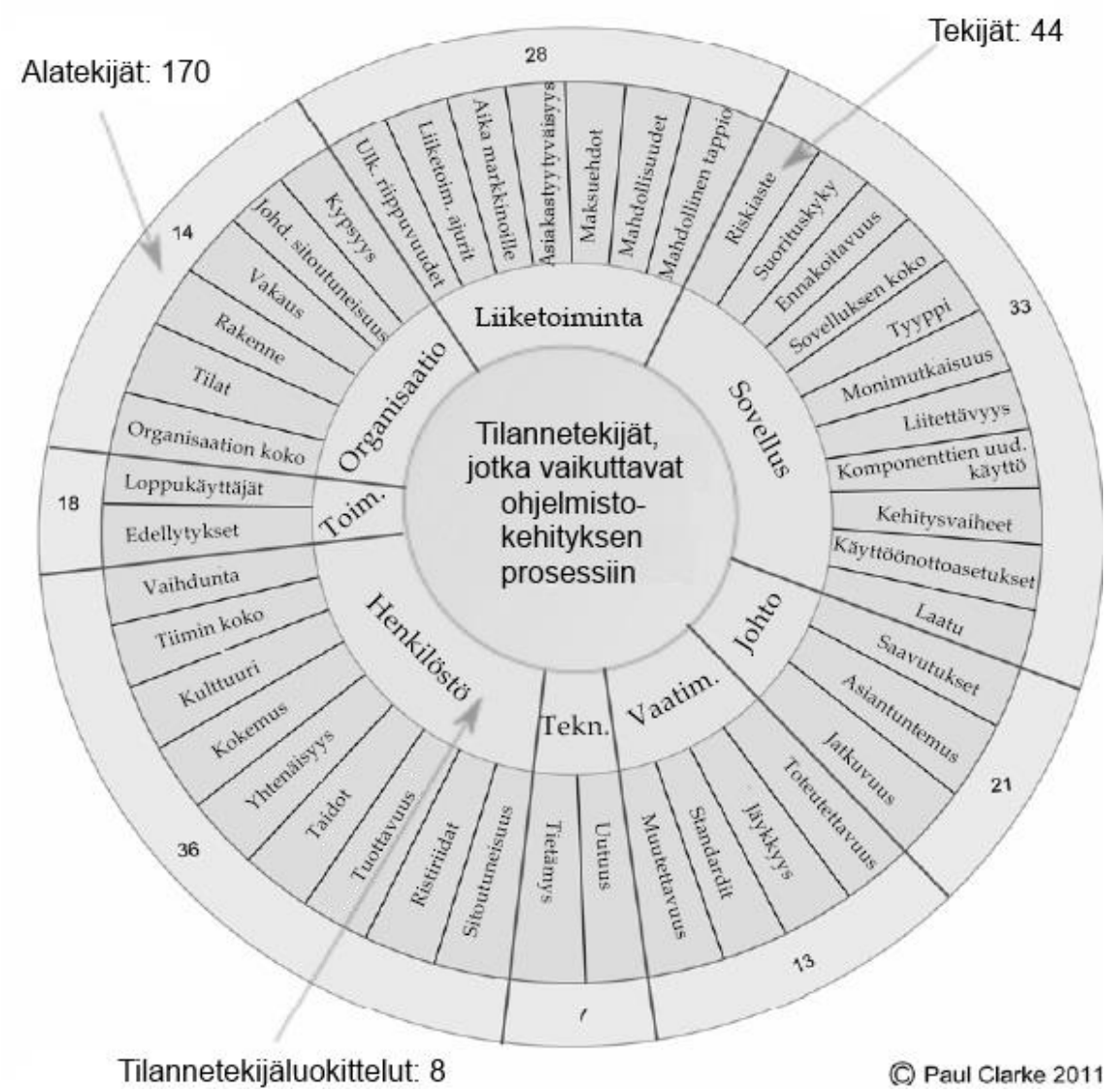
Menetelmän valintaa tilannetekijöiden mukaan on myös kritisoitu. Fitzgerald, Russo ja O´Kane (2002) toteavat, että kehittäjien oletetaan tuntevan kehittämismenetelmät niin hyvin, että he pystyvät valitsemaan tilannetekijöiden perusteella tilanteeseen kaikkein sopivimman. Käytännössä kuitenkin menetelmien tuntemus edes yhden menetelmän kohdalla ei ole välttämättä tarpeeksi hyvää, saati että tietämys riittäisi menetelmäportfoliosta oikean valitsemiseen. (Fitzgerald ym., 2002). Myös tilannekohtainen menetelmäkehitys lähtee tilannetekijöiden huomioimisesta. Tällöin tilannetekijät, kuten sovelluksen ominaispiirteet, tekniset ja ulkoiset tekijät sekä kehittäjien asiantuntemus, muodostavat projektin karakterisoinnin. Sen tulee ottaa huomioon projektiympäristön dynaamisuus. Karakterisointi toimii pohjana sopivien menetelmäosien valitsemiselle, ja niistä itse menetelmän koostamiselle. (Harmsen, 1997.)

Erilaisia tilannetekijöitä voi ottaa huomioon eri tavoin, ja niiden painotusarvo voi myös vaihdella projekteittain. MacCormack ja Verganti (2003) toteavat, että epävarmuuden ollessa pientä, esimerkiksi käytettävän alustan ollessa tuttu, ei ole tarvetta keskittyä aikaisen vaiheen integraatioon tai kattavaan beta-testaamiseen. Vastaavasti epävarmuuden lisääntyessä se tulee ottaa huomioon jo projektin aikaisemmissa vaiheissa, sillä niihin reagointi vasta myöhemmin ei välttämättä enää riitä. MacCormack ja Verganti huomasivat tutkimuksessaan, että käytettävän alustan sekä markkinoiden epävarmuuden kehittämisprosessissaan huomioivat projektit saavuttivat paremman suorituskyvyn. Kehitettävän ohjelmiston koko on hyvin merkittävä yksittäinen tilannetekijä, ja pienien sekä laajojen ohjelmistoprojektien prosessit eroavat toisistaan huomattavasti (Jones, 2007).

Kontingentti- ja tilannetekijöitä voidaan luokitella eri tavoin. Van Slooten ja Schoonhoven (1994) listaavat tekijöiksi muun muassa projektin keston, suunnan, laajuuden, syvyyden, alkuperän, henkilöiden keskinäiset suhteet, budjetin, teknologian, asiakkaan standardit sekä kehittäjien ammattitaidon. Backlund (2002) määrittelee tilannetekijät viiteen luokkaan ryhmiteltyinä. Nämä ovat liiketoiminta/kehityskonteksti, kehitettävä järjestelmä, kehittäjät, menetelmä sekä menetelmän rooli. Nämä jaetaan edelleen osiin. Clarke ja O´Connor (2012) toteavat, että laaja-alaista viitekehystä erilaisista tilannetekijöistä ei heidän kirjoituksensa julkaisuhetkellä vielä ole esitelty. He itse esittävät hyvin laaja–alaisen viitekehyksen tilannetekijöiden jäsentämiseksi (kuvio 6). Seuraavassa kuvataan tätä viitekehystä tarkemmin.

Clarken ym. (2012) viitekehyksessä päätasolla käytetään jäsennyksenä jakoa henkilöstöön, vaatimuksiin, sovellukseen, teknologiaan, organisaatioon, toimintaan, johtamiseen sekä liiketoimintaan. Kukin näistä on jaettu edelleen osatekijöihin, joita on kaiken kaikkiaan 44 kappaletta. Näiden alla on tunnistet-

tu vielä lisää (yhteensä 170 kappaletta) yksittäisiä tekijöitä. Tilannetekijät voivat vaikuttaa niin positiivisesti kuin myös negatiivisesti ohjelmistokehityksen prosessiin. (Clarke & O´Connor, 2012.)



KUVIO 6 Tilannetekijät, jotka vaikuttavat ohjelmistokehityksen prosessiin (Clarke & O´Connor, 2012, 16)

Liiketoimintatasolla otetaan huomioon strategiset ja taktiset liiketoiminnalliset tekijät. Näitä on seitsemän kappaletta. Ulkoiset riippuvuudet pitävät sisällään muun muassa riippuvuudet alihankkijoista tai muista projekteista sekä riippuvuuden eri sidosryhmien ja toimijoiden määrästä. Liiketoiminnan ajurit koostuvat muun muassa kulujen minimoimisesta, voiton maksimoimisesta ja markkinointitoimenpiteistä. Aika markkinoille tarkoittaa aikaa, joka menee, ennen kuin tuote on myytävänä. Asiakastyytyväisyys pitää sisällään niin yleisen asiakastyytyväisyyden kuin myös tyytyväisyyden käyttöliittymään. Maksuehdot pitävät sisällään mm. kiinteän hinnan periaatteen tai jonkun muun maksusuunnitelman. Mahdollisuudet taas kuvaavasti pitävät sisällään projektin tuo-

mat mahdollisuudet liiketoiminnalle. Mahdollinen tappio taas voi johtaa esimerkiksi rahoituksellisen tilanteellisen muuttumiseen, organisaation maineen heikentymiseen ja muutoksiin markkina-asemassa, kilpailuasemassa tai asiakastyytyväisyydessä. (Clarke & O´Connor, 2012.)

Sovellustason tilannetekijät liittyvät rakennettavaan ohjelmistoon. Riskiaste pitää sisällään esimerkiksi ihmismäärän, joihin projekti vaikuttaa, sekä sen, kuinka paljon ohjelmisto tulee vaikuttamaan loppukäyttäjien työskentelyyn. Suorituskyky pitää sisällään suorituskykyvaatimukset, reaaliaikaisen suorituskyvyn vajeet, tarvitun luotettavuuden ja arviot laitteiston- sekä ohjelmiston valmiuksista. Ennakoitavuus tarkoittaa sovellusalustan epävakaisuutta sekä viimeaikaisia muutoksia. Sovelluksen koko pitää sisällään niin laitteiston- kuin myös ohjelmiston, vaaditun muistin ja projektin suhteellisen koon sekä ajan. Ohjelmiston tyyppi taas sisältää esimerkiksi mahdolliset kriittisyydet, arkkitehtuurityypin ja itse sovellustyypin. Ohjelmiston monimutkaisuus tilannetekijänä voi tarkoittaa niin tuotteen, arkkitehtuurin kuin myös tehtävien kompleksisuutta. Liitettävyys taas koostuu yhteyksistä niin nykyisiin kuin myös tulevaisuudessa kehitettäviin järjestelmiin. Komponenttien uudelleenkäytössä otetaan huomioon ulkopuolisten komponenttien käyttömahdollisuus sekä uudelleenkäytön tarve. Kehitysvaiheet pitävät sisällään perinteiset ohjelmistonkehitysvaiheet, kuten kehittämisen ja ylläpidon. Käyttöönottoasetuksilla tarkoitetaan käyttöönotettujen ohjelmistojen määrää sekä käyttöönotettujen ohjelmistojen erilaisten versioiden määrää. Viimeisenä tekijänä sovelluksen laatu koostuu vaaditusta ohjelmistolaadusta ja ylläpidettävyydestä. (Clarke & O´Connor, 2012.)

*Johdon taso*on kuuluvat tilannetekijät liittyvät ohjelmistokehitystiimin johdon ominaispiirteistä ja kokoonpanosta. Asiantuntemus pitää sisällään laajan määrän tekijöitä, kuten projektin johtamismenetelmän tehokkuuden, kyvykkyydet projektin suunnittelussa, johdon viestintätaidot sekä projektin etenemisen hallitsemisen. Jatkuvuus tarkoittaa mahdollisia muutoksia johdossa. Saavutukset tarkoittavat projektin johtamisen kokemusta ja johtajan operatiivista tietämystä. (Clarke & O´Connor, 2012.)

Vaatimuksien alla on niihin liittyviä ominaispiirteitä. Toteutettavuus pitää sisällään mahdollisuudet, mihin ohjelmistokehitys pystyy venymään. Jäykkyys taas käsittää sen, kuinka tiukasti vaatimuksia tulee noudattaa. Standardit sisällyttävät muun muassa käyttäjien ymmärryksen vaatimuksista, ristiriitaiset vaatimukset sekä vaatimusten väärinymmärtämisen. Muutettavuus taas käsittää muun muassa jatkuvasti muuttuvat vaatimukset tai epäselvät vaatimukset. (Clarke & O´Connor, 2012.)

Teknologia tilannetekijänä tarkoittaa teknologiaa tai teknologioita, joita käytetään ohjelmistonkehitystyössä. Uutuus on teknologian tapauksessa kyseessä, mikäli se on uusi tai vasta kehittymässä. Tietämys teknologiasta pitää sisällään lukuisat teknologiaan liittyvät tietämyspiirteet. Näitä ovat muun muassa tietämys kielestä, työkaluista ja yleisesti teknologiasta sekä tarve uusille laitteistoille tai ohjelmistoille. (Clarke & O´Connor, 2012.)

Henkilöstöön liittyvät tilannetekijät ovat ohjelmistonkehittämisessä mukana olevien ei–johtotasolla toimivien työntekijöiden ominaispiirteet ja kokoonpano. Sitoutuneisuus käsittää sitoutuneisuuden projektiin itse tiimin sisällä. Ristiriidat ovat tiimin sisällä olevia henkilöiden välisiä konflikteja. Tuottavuus taas koskee yleistä tuottavuutta sekä tiimin kykyä suorittaa annetut tehtävät nopeasti. Taidot liittyvät ohjelmistokehityksessä tarvittaviin ammattitaitoihin, kuten kykyihin analysoida, ohjelmoida ja testata sekä tiimin ymmärrykseen kehitettävästä ohjelmistosta. Yhtenäisyys tilannetekijänä on muun muassa tiimin sisäistä yhtenäisyyttä, tehtävien menestyksekästä suorittamista tai liiallista riippuvuutta tiimin jäsenistä. Kulttuuri koostuu tiimin kulttuurista ja mahdollisesta vastustuksesta muutokseen. Tiimin koko on suhteellinen tiimin koko, kun taas vaihdunta tarkoittaa henkilöstössä tapahtuvaa vaihtuvuutta. (Clarke & O´Connor, 2012.)

Toiminallinen tai operatiivinen taso pitää sisällään toiminnalliset näkökulmat ja rajoitteet, jotka on edelleen jaettu loppukäyttäjien ja edellytysten alle. Näistä molemmat pitävät sisällään lukuisia alatasoja. Loppukäyttäjien taso koostuu muun muassa käyttäjien sitoutuneisuudesta, muutoksen vastustuksesta, osallistumisesta kehitykseen, ymmärrykseen järjestelmän mahdollisuuksista ja rajoitteista sekä perehtyneisyydestä ohjelmistotyyppiin. Edellytykset taas koostuvat esimerkiksi soveltuvista laeista, organisaation käytänteistä ja yleisistä käytänteistä. (Clarke & O´Connor, 2012.)

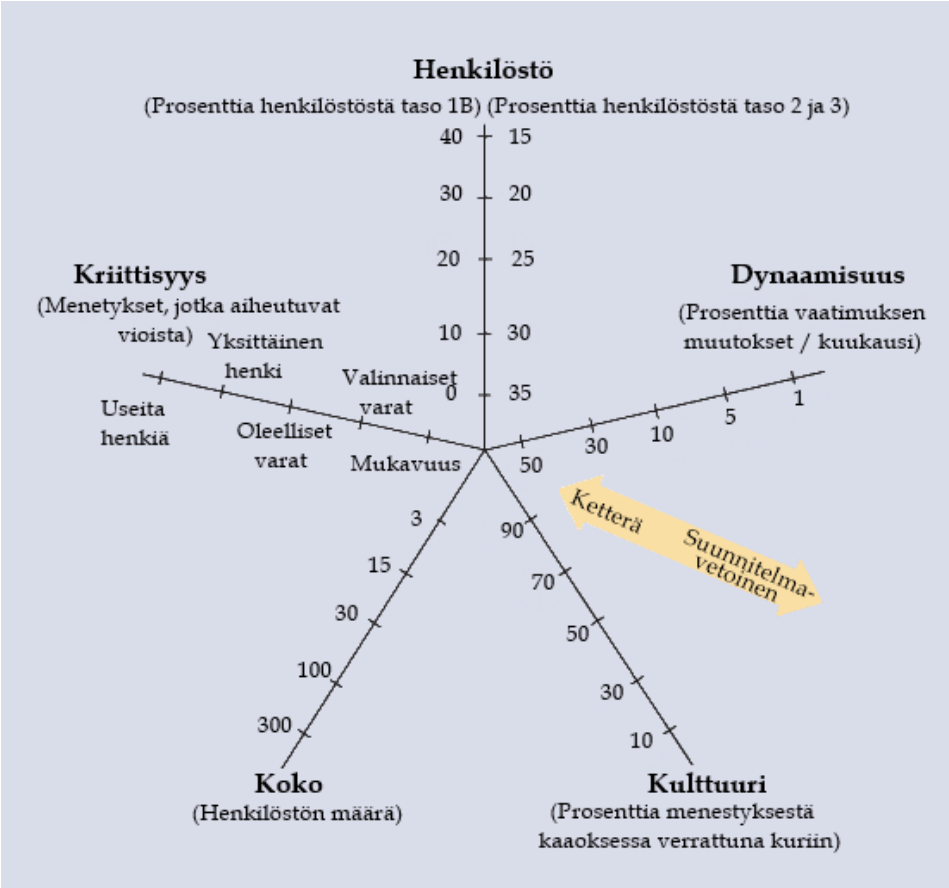
Viimeinen ylätaso on *organisaatiotaso*, joka on läpileikkaus organisaatiosta. Organisaation koko ja rakenne ovat molemmat tilannetekijöitä, joka vaikuttavat ohjelmistokehitykseen. Tilat taas tarkoittavat fyysisiä työskentelyolosuhteita, joissa projektia työstetään. Vakaus tarkoittaa muun muassa resurssien siirtymistä projektista toiseen prioriteettien mukaan, organisaation epävakautta tai organisaation käytänteiden vaikutusta projektiin. Johdon sitoutuneisuus tarkoittaa ylemmän johdon sitoutuneisuutta projektiin, tai sen puuttumista. Kypsyys taas on tilannetekijä, joka käsittää organisaation kypsyyden, nykyaikaisten ohjelmistokäytänteiden käytön sekä teknisen tuen saatavuuden. (Clarke & O´Connor, 2012.)

Boehm ja Turner (2003) ovat esitelleet riskiperusteisen lähestymistavan, jonka avulla voidaan tilannetekijöitä tutkimalla verrata suunnitelmavetoisia menetelmiä sekä ketteriä eli muutosvetoisia menetelmiä. Tätä vertailua voidaan käyttää hyväksi tehtäessä valintoja menetelmien suhteen. Lähestymistapa pitää sisällään viisi vaihetta. Ensimmäisessä vaiheessa tulee selvittää erilaisia riskejä tilannetekijöinä, jotka liittyvät ympäristöön sekä ketteriin ja suunnitelmavetoisiin menetelmiin. Tällöin voidaan saada arvokasta informaatiota tekijöistä, jotka luovat epävarmuutta kehityksen ajaksi. Toisessa vaiheessa kartoituksen perusteella katsotaan, sopiiko projekti suoraan ketterille tai suunnitelmavetoisille menetelmille. (Boehm & Turner, 2003.)

Kolmanteen vaiheeseen siirrytään, mikäli analyysi ei puolla vahvasti ketteriä tai suunnitelmavetoisia menetelmiä, tai jos tietyt tekijät puoltavat vahvasti toisia toisten tekijöiden puoltaessa toisia. Tällöin pyritään kehittämään arkkitehtuuria käyttämällä ensin ketteriä menetelmiä, ja sen jälkeen käytetään suun-

nitelmavetoisia menetelmiä. Mikäli sopivaa arkkitehtuuria ei pystytä luomaan, voidaan siirtyä kokonaan suunnitelmavetoisiin menetelmiin. Uusien riskien paljastuessa voidaan palata aikaisempaan vaiheeseen. Neljännessä vaiheessa keskitytään yleiseen projektin strategian luomiseen riskien perusteella. Jokaiselle riskille määritellään ratkaisustrategia ja niistä muodostetaan yhdistetty projektin strategia. Esimerkiksi tiimin kehittäjien ollessa kokeneita voidaan uudelleen käyttää prosessien osia, mutta vähemmän kokeneiden kehittäjien kanssa menee aikaa myös opettelemiseen. Viimeisessä vaiheessa seurataan ja arvioidaan valittuja prosesseja ja ympäristöä. Tällöin voidaan reflektoida aiemmin tehtyjä valintoja ja tarvittaessa tehdä niihin muutoksia. Samalla voidaan myös tunnistaa mahdollisia liiketoimintamahdollisuuksia. (Boehm & Turner, 2003.)

Boehm ja Turner (2003) esittelevät lähestymistavan yhteydessä niin sanotun viiden akselin kaavion (kuvio 7), jonka avulla voidaan tutkia viittä erilaista keskeistä tilannetekijää sen selvittämiseen, puoltavatko ne ketteriä menetelmiä vai suunnitelmavetoisia menetelmiä. Akselit ovat: projektin koko, kriittisyys, kulttuuri, dynaamisuus ja henkilöstö.



KUVIO 7 Viiden akselin kaavio (Boehm & Turner, 2003, 59)

Projektin koko on ilmeinen tekijä, ja sitä on helppo mitata henkilöstön määrällä. *Kriittisyys* koskee ohjelmiston virheiden seurauksia. Mikäli ihmisiä voi menehtyä ohjelmistovirheiden takia, ketterien menetelmien yksinkertainen suunnitte-

lu ja vähäinen dokumentaatio voi olla ongelma. *Kulttuuri*akseli lähtee siitä oletuksesta, että ketterät menetelmät sopivat paremmin organisaatioihin, joiden kulttuuri menestyy kaaoksessa kun taas suunnitelmavetoiset menetelmät viihtyvät kurinalaisessa kulttuurissa. Suunnitelmavetoiset menetelmät toimivat hyvin ei-*dynaamisissa* ympäristöissä, kun vaatimuksiin ei tapahdu paljoa muutoksia, kun taas ketterillä menetelmät sopivat molemmissa tapauksissa. Viimeisenä akselina on *henkilöstön* taitotaso, joka perustuu Cockburnin (2002) määrittelemiin luokituksiin. Tasolla -1 on henkilöitä, jotka eivät noudata tai osallistu menetelmien käyttöön. Tasolla 1B taas on henkilöitä, jotka koulutuksen avulla voivat suorittaa yksinkertaisia tehtäviä. Tason 1A henkilöt pystyvät tekemään monimutkaisempia tehtäviä, kuten käyttäjätarinoiden luokittelemista inkrementteihin. Tason 2 kehittäjät osaavat räätälöidä menetelmiä erilaisiin aikaisemmin olleisiin tilanteisiin, kun taas tason 3 henkilöt osaavat rikkoa menetelmän rajoja ja sovittaa sen myös ennakoimattomiin tilanteisiin. Ketterässä kehittämisessä tarvitaan enemmän ylempien tasojen henkilöitä. (Boehm & Turner, 2003.)

2.5 Yhteenveto

Tässä luvussa käsiteltiin menetelmiä ja niiden räätälöintiä. Menetelmällä tarkoitetaan kokoelmaa käytänteitä, tekniikoita, työkaluja ja dokumentaation apuvälineitä, jotka auttavat kehittäjiä ohjelmistokehityksessä. Se koostuu vaiheista, jotka jakautuvat edelleen osavaiheisiin. Menetelmiä voidaan luokitella monella tapaa, esimerkiksi sen suhteen kuinka paljon etukäteissuunnittelua niissä edellytetään ennen ohjelmointia. Perinteisesti menetelmät ovat pitäneet sisällään enemmän etukäteissuunnittelua, mutta 2000-luvulla etenkin ketterät menetelmät ovat olleet enemmän muutosvetoisia menetelmiä. Menetelmien käyttämisellä on havaittu olevan useita hyötyjä, kuten ohjelmiston laadun paraneminen sekä rahan ja ajan tarpeen väheneminen. Niiden käyttö ohjelmistoteollisuudessa onkin hyvin yleistä. Menetelmien käyttäminen ei kuitenkaan automaattisesti johda haluttuun lopputulokseen, ja menetelmiä onkin kritisoitu muun muassa joustamattomuudesta.

Mikään menetelmä ei sovellu sellaisenaan kaikkiin tilanteisiin. Käytettävä menetelmä voidaan kehittää ”tyhjästä”, mikä kuitenkin olisi varsin kallis ja työläs prosessi. Se voidaan myös luoda integroimalla jo olemassa olevien menetelmien osia toisiinsa. Käytettävä menetelmä voidaan myös valita useiden menetelmien joukosta projektin tilannetekijöiden mukaan, mutta käytännön työssä tämä on joissain tapauksissa nähty ongelmalliseksi. Sopiva menetelmä voidaan myös valita organisaation tai projektin käyttöön, jonka jälkeen sitä räätälöidään muun muassa prosessin osalta. Valitun menetelmän tulee olla joustava, jotta se soveltuu räätälöitäväksi. Räätälöinti tehdään tällöin joko organisaation tai prosessin tarpeiden mukaisesti. Räätälöinnin lähtökohdaksi otetaan tällöin kyseisen kontekstin (organisaation tai projektin) tilanne- eli kontingenssitekijät, jotka voivat liittyä henkilöstöön, vaatimuksiin, sovellukseen, teknologiaan, organi-

saatioon, toimintaa, johtamiseen ja liiketoimintaan. Tilannetekijöiden huomioon ottaminen onkin usein tärkeää ohjelmistoprojektin onnistumisen kannalta.

3 KETTERÄ KEHITTÄMINEN

Tässä luvussa tarkastellaan ensin lyhyesti ketterää lähestymistapaa ja sen jälkeen kolmea ketterää menetelmää. Lähestymistavasta pohditaan ketteryyden käsitettä ja esitellään Agile–manifestia. Lukuisista ketteristä menetelmistä tarkasteluun on valittu Scrum, XP ja Kanban. Niistä esitellään käytänteitä, prosesseja, rooleja ja vastuita.

3.1 Ketterä lähestymistapa

*Ketteryys (*engl. *agility)* ei ole käsitteenä uusi, mutta sille ei ole täsmällistä tai täydellistä määritelmää (Qumer & Henderson–Sellers, 2006a). Lyytinen ja Rose (2006) toteavat Abrahamssonin, Conboyn ja Wangin (2009) mukaan, että ohjelmistokehityksessä ketteryyden tulisi käsitteenä olla moniulotteinen ja kontekstisidonnainen ja että ketteryys saavutetaan monin eri keinoin riippuen projektin ympäristöstä. Tällöin jokaisen organisaation tulee omaksua oma yksilöllinen määritelmänsä ketteryydelle. Ketterää ohjelmistokehitystä koskevissa tutkimuksissa ketteryyden määritteleminen aina uudestaan on kuitenkin ongelmallista, sillä se ei tällöin tarjoa vakaata pohjaa, jonka päälle voisi rakentaa yhtenäistä tietopohjaa. (Abrahamsson ym., 2009.)

Ketteryys ei rajoitu pelkästään ohjelmistokehityksen alalle. Esimerkiksi Wong ja Whitman (1999) määrittelivät yrityksen olevan ketterä, mikäli se pystyy nopeasti vastaamaan huomisen odottamattomiin muutoksiin. Highsmith (2002) määrittelee ketteryyden kyvyksi luoda muutosta ja vastata muutoksiin, jotta yritys voi tehdä voittoa myrskyisässä liiketoimintaympäristössä. Ohjelmistokehityksessä ketteryyden määritelmä ja arvot pohjautuvat usein pääasiallisesti eri kehittäjistä koostuvan Agile–allianssin (2001a) laatimaan ketterän ohjelmistokehityksen manifestiin, sen arvoihin ja periaatteisiin. Arvot on esitetty manifestissa seuraavalla tavalla (Agile-Alliance, 2001a):

> ”Löydämme paremmiksi keinoiksi kehittää ohjelmistoja tekemällä sitä ja auttamalla muita tekemään sitä. Tämän kautta olemme tulleet arvostamaan:

Yksilöitä ja vuorovaikutusta enemmän kuin prosesseja ja työkaluja

Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentointia

Yhteistyötä asiakkaan kanssa enemmän kuin sopimusneuvottelua

Muutoksiin vastaamista enemmän kuin suunnitelman seuraamista."

Agile-manifestista on tärkeää huomata, etteivät oikealla puolella mainitut asiat ole turhia. Esimerkiksi työkalujen käyttäminen on kriittistä ohjelmistokehityksen nopeuttamiseksi ja dokumentaatio rajallisesti käytettynä helpottaa viestintää, parantaa tietämyksen siirtoa, säilyttää historiatietoa sekä täyttää juridisia sekä valtiollisia vaatimuksia. Agile-manifesti myös haastaa pohtimaan perinteisiä oletuksia ohjelmistokehityksestä, kuten esimerkiksi suunnitelmallisuuden korostamista. Kun äärimmäiset kohdat ovat esillä, kuten ei-dokumentaatiota ja täysin toimiva ohjelmisto tai täydellinen dokumentaatio ilman toimivaa ohjelmistoa, organisaatiot, tiimit ja yksilöt voivat löytää oman tasapainokohtansa niiden väliltä. (Highsmith, 2002.) Manifestin lisäksi Agile-allianssi (2001b) julkaisi 12 ketterää periaatetta, jota sen esittäjät noudattavat. Periaatteiden suomennokset on lainattu Agile-allianssilta (2001c).

1. Tärkein tavoitteemme on tyydyttää asiakas toimittamalla tämän tarpeet täyttäviä versioita ohjelmistosta aikaisessa vaiheessa ja säännöllisesti.

2. Otamme vastaan muuttuvat vaatimukset myös kehityksen myöhäisessä vaiheessa. Ketterät menetelmät hyödyntävät muutosta asiakkaan kilpailukyvyn edistämiseksi.

3. Toimitamme versioita toimivasta ohjelmistosta säännöllisesti,parin viikon tai kuukauden välein, ja suosimme lyhyempää aikaväliä.

4. Liiketoiminnan edustajien ja ohjelmistokehittäjien tulee työskennellä yhdessä päivittäin koko projektin ajan.

5. Rakennamme projektit motivoituneiden yksilöiden ympärille. Annamme heille puitteet ja tuen, jonka he tarvitsevat ja luotamme siihen, että he saavat työn tehtyä.

6. Tehokkain ja toimivin tapa tiedon välittämiseksi kehitystiimille ja tiimin jäsenten kesken on kasvokkain käytävä keskustelu.

7. Toimiva ohjelmisto on edistymisen ensisijainen mittari.

8. Ketterät menetelmät kannustavat kestävään toimintatapaan. Hankkeen omistajien, kehittäjien ja ohjelmiston käyttäjien tulisi pystyä ylläpitämään työtahtinsa hamaan tulevaisuuteen.

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

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

> 11. Parhaat arkkitehtuurit, vaatimukset ja suunnitelmat syntyvät itseorganisoituvissa tiimeissä.
>
> 12. Tiimi tarkastelee säännöllisesti, kuinka parantaa tehokkuuttaan, ja mukauttaa toimintaansa sen mukaisesti. (Agile-alliance, 2001b)

Erilaiset ketterät menetelmät noudattavat Agile-manifestia ja sen periaatteita, mutta voivat keskittyä painottamaan joitain periaatteita enemmän kuin toisia. Manifestia on kritisoitu siitä, että se on liian ympäripyöreä tieteellisen työn pohjaksi ja ettei siinä ole kunnollista perustaa johtamisteorioista ja filosofiasta. (Laanti, Similä & Abrahamsson, 2013.)

Highsmith ja Cockburn (2001) määrittelevät ketterien menetelmien strategian olevan muutoksista aiheutuvien kulujen vähentäminen koko projektin ajan. Abrahamsson ym. (2002) määrittelevät menetelmän olevan ketterä, mikäli se toimii inkrementaalisesti, korostaa yhteistoiminnallisuutta ja asiakaslähtöisyyttä, on suoraviivainen ja helposti opittava sekä valmis mukautumaan muutoksiin. Qumer ja Hendersons–Sellers (2006) määrittelevät ketteryyden ominaisuuksiksi joustavuuden, nopeuden, keveyden, oppimisen ja reagoivuuden. Tarkemmin he määrittelevät sen olevan jatkuvaa käyttäytymistä tai kykyä joltain entiteetiltä, joka joustavasti reagoi odotettuihin tai odottamattomiin muutoksiin nopeasti, noudattaa lyhyitä aikajaksoja, käyttää taloudellisia, yksinkertaisia ja laadukkaita työvälineitä, työskentelee dynaamisessa ympäristössä ja käyttää ajan tasalla olevaa aiempaa tietämystä ja kokemusta oppiakseen sisäisestä ja ulkoisesta ympäristöstään (Qumer & Henderson–Sellers, 2006). Conboy (2009) taas toteaa, että tuskin mikään ketterä menetelmä määrittelee ketteryyden samalla tavalla ja että menetelmien lähtökohdat tai painotukset vaihtelevat. Hän näkee joustavuuden (flexibility) ja keveyden (leaness) osana ketteryyttä. Ketteryyden hän määrittelee seuraavasti:

> Tietojärjestelmän kehittämismenetelmän jatkuva valmius nopeasti tai luonnostaan luoda muutosta, ennakoivasti tai reaktiivisesti omaksua muutosta, ja oppia muutoksesta, samalla kun se edistää asiakkaan koettua arvoa (taloudellisuutta, laatua ja yksinkertaisuutta) sen yhteisten osien ja suhteiden ympäristöönsä kautta (Conboy, 2009, 340).

Agile-manifestin julkaisemisesta on jo yli kymmenen vuotta aikaa. Tämän ajanjakson aikana korostus erilaisissa määritelmissä on vaihtunut kehitystiimikeskeisyydestä ja itsestään muotoutuvasta toiminnasta enemmän kohti kontrolloitua, organisoitua ja hallittua kokonaisuutta. Laanti ym. (2013) ovatkin tutkimuksissaan huomanneet, että ihmiset usein tarkoittavat eri asioita kun puhutaan ketterästä tai ketteryydestä. Tämä myös vaikeuttaa ketterien menetelmien käyttöönottoa. (Laanti ym., 2013.)

Ketterän ohjelmistokehityksen alkuvaiheessa monet henkilöt suosittivat ketterän menetelmän yhteydessä noudattamaan kaikkia sen käytänteitä, jotta se toimisi halutulla tavalla. Viime vuosina tähän ajatteluun on kuitenkin suhtauduttu aiempaa negatiivisemmin. (Anderson, 2010.) Ketterä ohjelmistokehitys ei ole ainoa keino joustaviin ohjelmistoprosesseihin, vaan samankaltainen tavoite

on usein myös tilannekohtaisella menetelmäkehityksellä ja menetelmien räätälöinnillä (Ågerfalk & Fitzgerald, 2006).

Ketterän lähestymistavan omaksumisen on sanottu olevan helpompaa innovatiivisille organisaatioille ja kulttuureille kuin byrokraattisille ja formaaleille. Myös käytetyllä teknologialla on merkitystä. Esimerkiksi olio-suuntautuneita teknologioita käyttävät organisaatiot voivat hyödyntää ketteriä menetelmiä paremmin kuin suurtietokoneille ohjelmistoja kehittävät organisaatiot. (Nerur, Mahapetra & Mangalaraj, 2005.) Ketterien menetelmien onkin sanottu vetoavan innovatiivisiin kulttuureihin niiden yksilön kykyjä, ohjelmiston toimittamista ja vähennettyä formaalisuutta korostavien painotuksien vuoksi (Highsmith, 2003). Ketteryyden on sanottu olevan mahdollista ainoastaan, jos jokainen projektissa mukana oleva noudattaa itse tarkemmin kurinalaisuutta, eivätkä ketterät menetelmät ole tekosyy yksipuoliselle käytökselle (Beck & Boehm, 2003).

Ketterää lähestymistapaa ja ketterää kehittämistä on myös kritisoitu. Nerur ym. (2005) toteavat, että perinteisiä menetelmiä pitkään käyttäneillä on todennäköisesti ongelmia ketterien menetelmien omaksumisessa. Niiden on myös sanottu epäonnistuvan arkkitehtuuria ja suunnittelua koskevien kysymysten huomioimisessa, ja sen vuoksi se voi aiheuttaa ei-optimaalisia suunnittelupäätöksiä (Mordinyi, Kühn, & Schatten, 2010). Itseorganisoituvia tiimejä on kritisoitu siitä, että eri yksilöistä on vaikea muodostaa tiimiä, joka pystyisi organisoitumaan itse ja tuottamaan hyviä tuloksia (Cohn, 2009). Joidenkin ketterien menetelmien on myös sanottu kattavan vain osan kehittämisen elinkaaresta eivätkä ne välttämättä anna ohjeita räätälöintiin (Abrahamsson, Warsta, Siponen & Ronkainen, 2003). Ketterien menetelmien on myös huomattu johtavan tehottomiin päätöksiin tiimin jäsenten halutessa miellyttää toisia ja vähentävän tätä kautta myös oppimista (McAvoy & Butler, 2007). Kritiikkiä on kohdistunut muun muassa rajoitettuun tukeen alihankintaa hyödyntävän kehityksen, hajautetun kehityksen, turvallisuuskriittisten ohjelmistojen, laajojen ja monimutkaisten ohjelmistojen sekä uudelleenkäytettävien komponenttien rakentamisen yhteydessä (Turk, France & Rumpe, 2002). Tutkimuksia ketterien menetelmien käytöstä tämänkaltaisissa projekteissa on kuitenkin olemassa, kuten turvallisuuskriittisten ohjelmistojen kanssa (Bowers, May, Melander, Baarman & Ayoob, 2002) ja hajautetussa ohjelmistokehityksessä (Hossain, Bannerman & Jeffery, 2011). Tekijöihin voi vaikuttaa huomioimalla niitä eri tavoin. Esimerkiksi laajojen ohjelmistojen kehityksessä voi ketterien menetelmien kanssa käyttää hallittua, mutta kevyttä tietohallintoa. Tämän on huomattu auttavan laajojen projektien onnistumisessa. (Qumer & Henderson-Sellers, 2008.) Valitettavasti ketteriä menetelmiä ja niiden hyötyä koskevien laadukkaiden empiiristen tutkimuksien määrää on vielä vähäinen (Dybå & Dingsøyr, 2008).

Erilaisia ketteriä menetelmiä ja menetelmän tapaisia menettelytapoja on esitelty lukuisia. Näitä ovat muun muassa eXtreme Programming (XP) (Beck, 1999), the Dynamic System Development Method (DSDM) (Stapleton, 1997), Scrum (Schwaber & Beedle, 2002), Crystal (Cockburn, 2001), Agile Modeling (Ambler, 2002), Feature Driven Design (FDD) (Coad, de Luca & Lefebvre, 1999), Lean Software Development (LSD) (Poppendieck, 2001) sekä Kanban (Ander-

son, 2010). Tässä tutkielmassa rajoitutaan tarkastelemaan Scrumia, XP:ä sekä Kanbania, koska ne ovat käytännön työssä suosituimpia (VersionOne, 2014; Rodriguez, Markkula, Oivo & Turula, 2012) ja myös eniten tutkittuja ketteriä menetelmiä.

3.2 Scrum

Scrum on 1990–luvun puolessa välissä kehitetty ketterä kehittämismenetelmä (Schwaber, 1995), joka toi ideoita teollisuuden prosessikontrolliteoriasta ohjelmistokehityksen käyttöön. Se korostaa joustavuutta, mukautuvuutta ja tuottavuutta. Sen sanottiin soveltuvan etenkin pienten, mielellään 3–10 henkilöstä koostuviin tiimeihin. Scrumin on sanottu olevan kevyt ja helppo ymmärtää, mutta vaikea taitaa. (Schwaber & Sutherland, 2013.) Menetelmä tarjoaa etenkin projektin johtamisen käytäntöjä enemmän kuin muut ketterät menetelmät (Conboy, 2009).

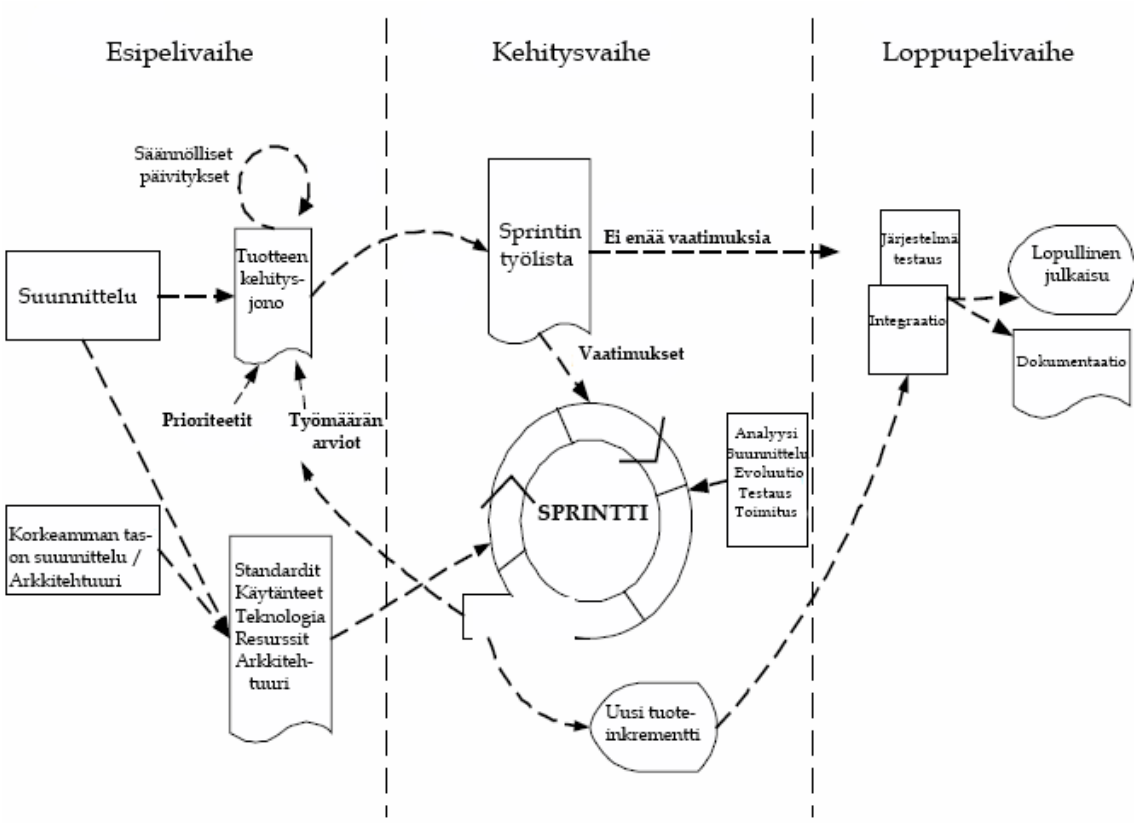
Scrumissa kehittämisprosessi on jaettu kolmeen vaiheeseen, esipelivaiheeseen (engl. pregame phase), peli/kehitysvaiheeseen (engl. development phase) sekä loppupelivaiheeseen (engl. postgame phase). Kehitys etenee iteratiivisesti sekä inkrementaalisesti. (Abrahamsson ym., 2002.) Kuviossa 8 on esitetty Scrumin prosessimalli Abrahamssonin ym. (2002) esityksen mukaisesti. Tämä esitys noudattaa Schwaberin (1995) kuvausta. Myöhemmissä esityksissä (esim. Schwaber & Sutherland, 2013) ei Scrumin prosessia ole vaiheistettuna kuvattu.

Esipelivaiheessa on kaksi alavaihetta, suunnitteluvaihe ja arkkitehtuurin/korkeamman tason suunnittelu. *Tuotteen kehitysjono* (engl. *Product Backlog)* on projektin työlista, johon muodostetaan rakenne priorisoiduista vaatimuksista. Sitä ylläpidetään koko projektin aikana muun muassa lisäämällä uusia kehitettäviä ominaisuuksia ja muuttamalla vanhoja. Samalla suunnitellaan kehitysjonon avulla myös korkeamman tason ja arkkitehtuurin mallinnukset. Tämän lisäksi esipelivaiheen aikana määritellään projektitiimi, työkalut, muut resurssit, koulutustarpeet ja hyväksymisen johtaminen. (Abrahamsson ym., 2002.)

Kehitysvaiheessa ohjelmistoa kehitetään *sprinteissä*, joiden suositeltu kesto on kahdesta neljään viikkoa. Sprintin aikana tarkkaillaan erilaisia ympäristöllisiä ja teknisiä muuttujia, joita ovat muun muassa aikaikkuna, laatu, vaatimukset, resurssit ja toteutusteknologiat. Sprinttien aikana tehdään ohjelmistoon uusia ominaisuuksia tai parannellaan jo olemassa olevia, valittavien tehtävien valikoituessa *Sprintin työlistaan (*engl. *Sprint Backlog)* tuotteen kehitysjonosta. (Abrahamsson ym., 2002.) Sprintin aikana ei tehdä muutoksia, jotka vaikuttavat sen tavoitteisiin, eikä sen laatutavoitteita lasketa, ja kehitystiimin koostumus pidetään samana. Tarvittaessa työlistaa voidaan muuttaa ja tarkentaa, mikäli ratkaistava ongelma tullaan tämän avulla ymmärtämään paremmin. (Schwaber & Sutherland, 2013.)

Loppupelivaiheeseen tullaan kun on yhteisymmärrys ympäristötekijöiden suhteen, kuten vaatimusten saavuttamiseen pääsemisestä. Tällöin sprintin työlista on tyhjä ja voidaan tehdä muita tarpeellisia toimintoja, joita voivat olla

muun muassa integrointi, dokumentaatio ja järjestelmätestaus. (Abrahamsson ym., 2002.)



KUVIO 8 Scrumin prosessi (Abrahamsson ym., 2002, 28)

Scrum-tiimissä on kolmenlaisia henkilörooleja. *Scrum–mestari (*engl. *Scrum Master)* on hallinnollinen henkilö, jonka vastuulla on projektin ja sprinttien eteneminen Scrumin käytäntöjen, arvojen ja sääntöjen mukaisesti. Samalla hän myös toimii tiimiä palvelevana johtajana ja pyrkii maksimoimaan tiimin työn arvon. *Kehitystiimi (*engl. *development team)* koostuu ohjelmistoalan ammattilaisista. Tiimi tuottaa kehitettävää ohjelmistoa inkrementeissä, jolloin ominaisuus siirtyy työlistasta valmiiksi. Kehitystiimi on itseorganisoituva. Tämä tarkoittaa muun muassa sitä, että tiimi itse päättää kuinka se valitsee ominaisuudet tuotteen kehitysjonosta, ja miten se toteuttaa ne valmiiksi. Tiimin jäsenet toimivat eri rooleissa sen sisällä ja samat henkilöt voivat suorittaa esimerkiksi ohjelmointia ja testausta. *Tuoteomistaja (*engl. *product owner)* on henkilö, joka on vastuussa tuotteen kehitysjonon ylläpidosta ja sen priorisoinnista. Tämän lisäksi hän on vastuussa tuotteen kehitysjonon sisällön selvittämisestä ja läpinäkyvyydestä kehittäjätiimille, sen selvyydestä eli mitä tulee tehdä seuraavaksi sekä tiimin työn arvon varmistamisesta. Tuoteomistaja voi esimerkiksi olla työntekijä kohdeorganisaatiosta, jolle ohjelmistoa ollaan kehittämässä. Scrum korostaa läpinäkyvyyttä, ja erilaisten *tuotosten* (engl. artifact) kuten sprintin työlistan ja tuotteen kehitysjonon tulee olla mahdollisimman läpinäkyviä. Scrum-mestarin tulee työskennellä tuoteomistajan, kehitystiimin ja muiden asianomaisten kans-

sa ja varmistua siitä, että tuotokset todella ovat läpinäkyviä kaikille. (Schwaber & Sutherland, 2013.)

Scrumissa järjestetään erilaisia tapahtumia. *Suunnittelupalaveri (*engl. *sprint planning meeting)* koostuu kahdesta osasta. Ensimmäisessä osassa tuoteomistaja esittelee tuotteen kehitysjonon, jonka jälkeen kehittäjätiimi itse päättää, mitkä tullaan valitsemaan sprintin työlistaan. Tämän jälkeen sprintille asetetaan konkreettinen tavoite, joka antaa kehittäjätiimille konkreettisen näkökulman miksi sprintin inkrementtiä ollaan kehittämässä. Toisessa osassa tiimi itseorganisoituu ja suunnittelee miten valittu työ tullaan toteuttamaan, neuvotellen tarvittaessa tuoteomistajan kanssa sprintin työlistasta. (Schwaber & Sutherland, 2013.)

*Päiväittäinen Scrum (*engl. *Daily Scrum)* on päivittäinen palaveri, jonka aikana kehittäjätiimi kokoontuu johonkin tilaan enintään 15 minuutiksi. Palaverissa tiimi tahdistaa keskinäiset työnsä asettaen suunnitelman ja ennustuksen mitä seuraavan 24 tunnin aikana tullaan tekemään. Päivittäisessä Scrumissa tiimin jäsenet kertovat muun muassa, mitä on saatu aikaan viimeisen 24 tunnin aikana sekä mitä ongelmia on kehityksessä ollut. (Schwaber & Sutherland, 2013.)

Kohdassa 2.1.2 jäsennettiin menetelmä eri näkökulmien mukaisesti osiin (vrt. Kuvio 2). Samaa jäsennystä noudattaen on taulukossa 1 esitetty tiivistelmä Scrum-menetelmästä.

TAULUKKO 1 Scrum jaettuna osiin

Menetelmän osa	**Scrum**
Tausta	1990-luvun puolessa välissä kehitetty menetelmä, joka tuo ideoita teollisuuden prosessikontrolliteoriasta ohjelmistokehitykseen
Lähestymistapa	Ketterä lähestymistapa Empiirinen lähestymistapa
Periaatteet	Ketterän kehittämisen periaatteet
Soveltaminen	Alun perin tarkoitettu pienille, 3–10 henkilöstä koostuville kehitystiimeille, mutta nykyään sovelletaan myös laajemmissa projekteissa
Prosessi	Iteratiivinen prosessimalli, jossa esipelivaihe, kehitysvaihe ja loppupelivaihe seuraavat toisiaan
Roolit	Scrum-mestari, kehittäjä, tuoteomistaja
Käytäntöjä	Suunnittelupalaveri, päivittäinen palaveri, sprinttikatselmus, sprintin retrospektiivi, tuotteen kehitysjono, sprintin kehityslista, edistymiskäyrä, valmiin määritelmä

Sprintin lopussa järjestetään *sprinttikatselmus (*engl. *Sprint Review)*, jossa tuoteomistaja katsoo, mikä osa työstä on ”valmista”, ja mikä ei ole ”valmista”. *Valmiin määritelmä* (engl. *definition of done)* voi vaihdella Scrum-tiimien välillä, mutta tiimin sisällä tulee olla jaettu ymmärrys siitä, mitä valmiilla kehitysjonon tehtävällä ja inkrementillä tarkoitetaan. Sprinttikatselmuksessa tiimi ja eri sidos-

ryhmät tutkivat kehitettyä tuoteversiota keskustellen ja antaen palautetta. Tarvittaessa tämän jälkeen tehdään muutoksia tuotteen kehitysjonoon. Scrumin jokaisessa vaiheessa tulisi olla mahdollista arvioida työmäärä, joka on jäljellä päämäärän saavuttamiseksi. Tuoteomistajan tulee tarkkailla jäljellä olevaa työmäärää vähintään jokaisen sprinttikatselmuksen aikana. Tätä voidaan graafisesti tarkastella *edistymiskäyrän (*engl. *burndown chart)* avulla. *Sprintin retrospektiivissä (engl. sprint retrospective)* Scrum–tiimi tarkastelee omaa työskentelyään ja kehitysprosessiaan ja tarvittaessa tekee muutossuunnitelman toiminnan parantamiseksi. (Schwaber & Sutherland, 2013.)

3.3 XP

Extreme Programming eli XP (Beck, 1999) on toinen suosittu ketterä kehittämismenetelmä. Siinä pääpaino on itse kehitystyössä (Conboy, 2009). Se on suunniteltu vastaamaan ohjelmistokehityksen eritystarpeisiin: pienet tai keskisuuret tiimit tekevät kehitystyötä projekteissa, joissa vaatimukset ovat epämääräisiä ja muuttuvia. Sen neljä keskeistä arvoa ovat viestintä, yksinkertaisuus, palaute sekä rohkeus. XP pitää sisällään erilaisia rooleja, prosesseja ja käytänteitä. (Beck, 1999.) Seuraavassa kuvataan ensin rooleja, sitten prosessia ja lopuksi käytänteitä. Kuvaus perustuu Beckin (1999) esitykseen. XP:stä on myös julkaistu Beckin ja Andresin (2004) toimesta uudempi versio, joka pitää sisällään mm. 24 eri käytäntöä. Tieteenalan sisällä ja käytännössä laajennettua versiota on kuitenkin tutkittu ja sovellettu vähemmän kuin alkuperäistä versiota (Conboy & Fitzgerald, 2010).

*Ohjelmoija (*engl. *programmer)* on kehitystiimissä toimiva kehittäjä, joka kirjoittaa ohjelmakoodia ja yksikkötestejä. Kehityksessä pyritään mahdollisimman yksinkertaiseen toteutukseen ja pieniin rakenteisiin. Ohjelmoijan tulee myös osata viestiä sidosryhmille, kuten toisille ohjelmoijille ja asiakkaalle. *Asiakas (*engl. *customer)* on parhaassa tapauksessa myös kehitettävän ohjelman lopullinen käyttäjä. Hänen tulee tietää, mitä ohjelmoijien tulee toteuttaa, ja viestiä se mahdollisimman selvästi sekä tehdä päätöksiä myös mahdollisissa ristiriitatilanteissa. Viestiminen tapahtuu muun muassa käyttäjätarinoita kirjoittamalla. Myös toiminnallisten testien kirjoittaminen on osittain asiakkaan vastuulla. (Beck, 1999.)

*Testaaja (*engl. *tester)* auttaa asiakasta toiminnallisten testien kirjoittamisessa. Hän on myös vastuussa niiden ajamisesta, tuloksista tiedottamisesta sekä testityökaluista. *Seuraaja (*engl. *tracker)* on henkilö, joka antaa palautetta kehitystiimille. Palautteen on tarkoitus auttaa tiimejä ja kehittäjiä kehittymään. Tämä pitää sisällään muun muassa arviointia kokonaisuutena ja yksittäisille kehittäjille. Tämän lisäksi seuraajan tulee pitää yllä yleiskuvaa kehitettävästä ohjelmistosta ja arvioida, missä tahdissa ohjelmisto esimerkiksi täyttää tietyt vaatimukset. Hän myös dokumentoi muun muassa toiminnallisten testien tulokset.

*Valmentaja (*engl. *coach)* on vastuussa kehitysprosessista kokonaisuutena. Hänen tulee ymmärtää ohjelmisto normaalia syvällisemmin ja ohjeistaa kehitys-

tiimiä. Kehitystiimin ja jäsenten on tarkoitus toimia itsenäisesti, mutta joissain määrin ohjaus ja palaute on tarpeellista. Myös käskyttäminen ristiriitatilanteissa on valmentajan vastuulla. *Konsultti (*engl. *consultant)* on ulkopuolinen henkilö, joka auttaa esimerkiksi tietyn ongelman selvittämisessä tai tarjoaa teknistä tietämystä aiheesta. *Iso pomo (*engl. *big boss)* on ylemmän tason johtohenkilö, joka vastaa tärkeistä päätöksistä.

XP:n prosessi jaetaan kuuteen osaan. *Tutkimusvaiheessa (*engl. *exploration)* asiakas tekee tarinakortteja tulevan ohjelmiston vaatimuksista. Samaan aikaan ohjelmoijat kehittävät arkkitehtuuri- ja teknologiaratkaisuja, tarvittaessa konsultoiden aiheesta asiantuntijoita. Myös laitteistoa voidaan testata ja miettiä reaalimaailman elementeillä, kuten realistisella määrällä verkkoliikennettä. Mikäli kehitystiimin jäsenet ovat ennestään tuttuja ja teknologia on omaksuttu, tutkimusvaihe voi olla vain muutaman viikon pituinen. Tarvittaessa se voi kuitenkin kestää pidempään. *Suunnitteluvaihe (*engl. *planning)* on muutaman päivän kestävä vaihe, jossa tarinakortteja priorisoidaan ja niiden avulla valitaan ensimmäiseen julkaisuun halutut ominaisuudet. *Iteraatiot ennen ensimmäistä julkaisua (*engl. *iterations to first release)* on vaihe, jossa kehitystyötä tehdään 1–4 viikon mittaisissa iteraatiojaksoissa. Aluksi iteraatioiden aikana kehitetään arkkitehtuuria, mutta myöhemmin toiminnallisuus on tärkeämpänä osana. Asiakas valitsee, mitkä tarinat valitaan iteraatioon toteutettavaksi. Jokaisen iteraation lopussa testit on mahdollista ajaa ja järjestelmä on toimintavalmis.

*Tuotteistaminen (*engl. *productionizing)* on julkaisun jälkeistä palautteen antamista ja suorituskyvyn testausta. Tuotteistamisen aikana otetaan jatkokehittämisideoita talteen, ja sitä suoritetaan lyhyemmissä iteraatioissa. *Ylläpitovaihe (*engl. *maintenance)* on vaihe, jolloin ohjelmistoa ja kehitystiimin koostumusta pidetään yllä. Ylläpitovaiheen aikana luodaan myös uutta toiminnallisuutta ohjelmaan. Yleensä tässä vaiheessa projektin nopeus vähenee ja työstä tulee enemmän rutiininomaista. Myös ohjelmistoon tarvittavat julkaisut julkaistaan. Ylläpitovaihe on yleensä yleisin vaihe XP:n prosessissa. *Lopetusvaihe (*engl. *death)* tulee, kun asiakas ei enää tuo uutta toiminnallisuutta kehitettäväksi. Tämä voi johtua esimerkiksi siitä, että järjestelmä täyttää kaikki halutut vaatimukset tai sitä ei jostain syystä kannata enää kehittää. Tämän jälkeen kirjoitetaan lopulliset dokumentaatiot ohjelmistosta.

Edellä mainittujen lisäksi XP:ssä on useita toisiinsa liittyviä käytänteitä. Käytänteiden yhdessä käytön yhteisvaikutuksen on katsottu tuovan erityistä hyötyä. Seuraavassa kuvataan lyhyesti yleisimmät käytänteet. *Suunnittelupeli (*engl. *planning game)* on käytänne, jossa ohjelmoijat arvioivat, kuinka paljon asiakaskorttien sisältöjen toteuttamiseen menee aikaa. Tämän lisäksi arvioidaan, mitä vaikutuksia niillä on ohjelmistoon ja miten työprosessi tulisi organisoida. Asiakas suunnittelee tämän perusteella, mitä toteutetaan, millaisissa julkaisuissa, miten priorisoiden ja milloin julkaisut tullaan julkaisemaan. *Pienet julkaisut (*engl. *small releases)* tarkoittaa, että jokaisesta julkaisusta pitää tehdä niin pieni kuin mahdollista ja niiden tulee pitää sisällään liiketoiminnallista arvoa. Ominaisuuksien tulee olla myös valmiiksi kehitettyjä. (Beck, 1999.)

*Yksinkertainen suunnittelu (*engl. *simple design)* tarkoittaa, että kehitettävä ohjelmakoodi ja arkkitehtuuri pidetään mahdollisimman yksinkertaisena. Myös saman asian tekemistä useampaan kertaan tulee välttää. *Metafora (*engl. *metaphor)* tarkoittaa kehitettävän ohjelmiston kuvailemista kielikuvien avulla. Tämä helpottaa osapuolten välistä viestintää. *Testivetoinen kehittäminen (*engl. *test–driven development)* on automaattisten testien luomista ennen itse koodin kirjoittamista. *Refaktorointi (*engl. *refactoring)* tarkoittaa olemassa olevan koodin muuttamista rakenteeltaan yksinkertaisemmaksi ilman, että itse toiminnallisuus muuttuu millään tavalla.

Käyttämällä samaa jäsennystä kuin Scrumin yhteydessä voidaan XP-menetelmä kuvata tiivistetysti taulukon 2 mukaisesti.

TAULUKKO 2 XP jaettuna osiin

Menetelmän osa	**XP**
Tausta	1990-luvun lopulla kehitetty menetelmä, jossa pääpaino on itse kehittämistyössä.
Lähestymistapa	Ketterä lähestymistapa Empiirinen lähestymistapa
Periaatteet	Ketterän kehittämisen periaatteet
Soveltaminen	Pienet ja keskisuuret tiimit
Prosessi	Iteratiivinen prosessimalli, jossa tutkimusvaihe, suunnitteluvaihe, kehitysvaihe (iteraatiot ennen julkaisua), tuotteistaminen, ylläpitovaihe ja lopetusvaihe seuraavat toisiaan
Roolit	Ohjelmoija, asiakas, seuraaja, valmentaja, konsultti, iso pomo
Käytäntöjä	Suunnittelupeli, pienet julkaisut, yksinkertainen suunnittelu, metafora, testivetoinen kehittäminen, refaktorointi, pariohjelmointi, yhteisomistajuus, jatkuva integraatio, 40 tunnin työviikko, paikalla oleva asiakas, koodausstandardit, avoin työtila

*Pariohjelmointi (*engl. *pair programming)* on käytäntö, jossa ohjelmakoodin kirjoittamista suoritetaan kehittäjäpareissa. Tällöin yhden tietokoneen ääressä työskentelee kaksi henkilöä. *Yhteisomistajuus (*engl. *collective ownership)* on koko kehitystiimiä koskeva käytäntö, jossa kukaan yksittäinen henkilö ei ”omista” ohjelmakoodin osia. Mikäli yksi kehittäjä näkee koodin, johon voisi lisätä jotain arvoa, hän on vapaa tekemään muutoksen siihen. Kaikki kehittäjät ovat vastuussa kehitetystä koodista. *Jatkuva integraatio (*engl. *continuous integration)* on yksittäisen kehittäjän tai parin kehittämän ohjelmakoodin integrointia ohjelmistoon lyhyissä aikajaksoissa. Kun koodi on saatu integroitua, kehittäjä tai kehittäjäpari ajaa tarvittavia testejä niin kauan, kunnes ne kaikki menevät lävitse. *40 tunnin työviikko (*engl. *40–hour week)* tarkoittaa työmäärän pitämistä kurissa. Ihminen ei pysty jatkuvasti työskentelemään ylikuormitettuna samaan aikaan

kun pitää olla luova työntekijä. Kahdella perättäisellä viikolla ylitöiden teettäminen on kiellettyä.

*Paikalla oleva asiakas (*engl. *On-site customer)* on käytäntö, jossa kehitystiimin mukana on jokin asiakkaan edustaja. Asiakas on henkilö, joka tulee käyttämään kehitettävää ohjelmistoa itse myöhemmin. Hän vastaa kehittäjätiimin kysymyksiin ja antaa palautetta kehitystiimille. *Koodausstandardit (*engl. *coding standards)* ovat projektin tai organisaation yhteisiä käytäntöjä ohjelmoinnille. Vaikka tiimin jäsenet vaihtuisivatkin kesken projektin, ohjelmakoodin rakenne ei muutu ja se on kaikille ymmärrettävää. *Avoin työtila (*engl. *open workspace)* on käytäntö jossa projektille määritellään tietty työtila, jossa on tarpeelliset välineet kehittämiselle. Avoin työtila korostaa viestinnän merkitystä. (Beck, 1999.)

3.4 Kanban

Kan-ban on japaninkielinen sana, joka kirjaimellisesti tarkoittaa viestikorttia. Teollisuustuotantoympäristössä korttia käytetään tuotantoprosessissa ilmaisemaan prosessin aiemmalle portaalle tuottaa lisää. Aiemman portaan työntekijät eivät saa tehdä enempää työtä, elleivät he saa kanban-korttia seuraavalta portaalta. Sitä on käytetty Japanissa muun muassa Toyotan tehtailla osana kevyttä (lean) tuotantoa. Kanban on nähty keskeiseksi työkaluksi *kaizen-periaatteen* eli jatkuvan parantamisen periaatteen noudattamisessa. Kaizen-kulttuurissa työvoima tuntee itsensä voimaantuneeksi tehdä asioita, jolloin se pystyy pelkäämättä ratkaisemaan ongelmia, keskustelemaan erilaisista vaihtoehdoista ja toteuttamaan korjauksia sekä parannuksia. Työntekijät pystyvät itse organisoitumaan ja päättämään, kuinka he tekevät työnsä. Myös epäonnistumisia suvaitaan, mikäli ne tapahtuvat parannuksia yritettäessä. Paikallisella tasolla parannuksia tehtäessä pyritään myös ajattelemaan organisaation ja tiimin hyvää, ja ne parantavat yleistä suorituskykyä. Yleisesti työympäristö ja kulttuuri ovat yhteistyökykyisiä, ja luottamus sekä arvostus toisia työntekijöitä kohtaan on korkealla, riippumatta työntekijöiden asemasta organisaatiossa. Luottamus tekee yleensä organisaation hierarkiasta matalamman. Kaizen-kulttuuri voikin vähentää turhia johtamistasoja ja vähentää sitä kautta myös työn koordinointikustannuksia. Monet kaizen-kulttuurin periaatteista ovat läntisen kulttuurin kanssa ristiriidassa. (Anderson, 2010.)

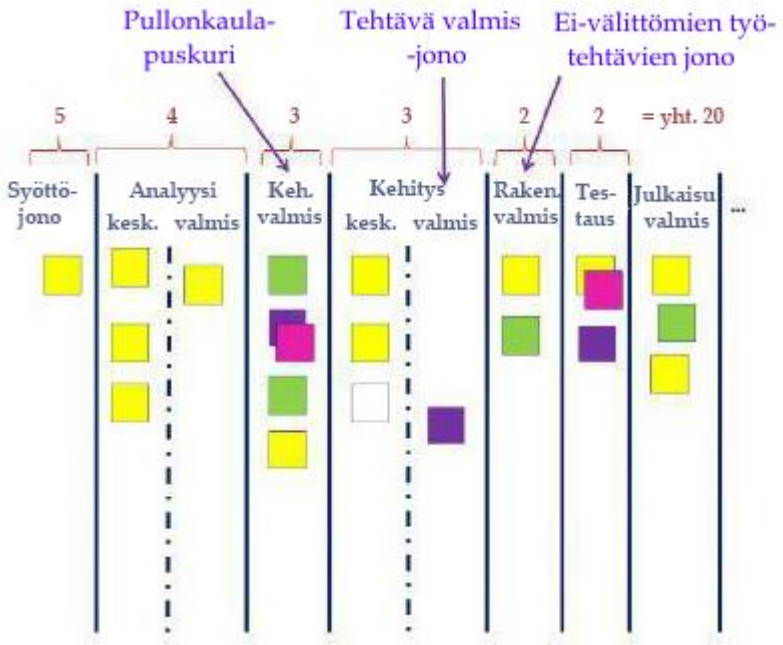
Ohjelmistonkehitysympäristössä Kanban kehitettiin soveltamaan kevyitä periaatteita. Andersonin mukaan Kanban ei ole varsinaisesti ohjelmiston kehittämismenetelmä, vaan se vaatii, että on jo jokin prosessi, johon Kanbania voidaan soveltaa. Myöskään olemassa olevia työnkulkuja, työnimikkeitä, rooleja, vastuita tai käytänteitä ei tulisi muuttaa. Vanhan prosessin optimointi on helpompaa ja nopeampaa, jolloin se kohtaa vähemmän vastustusta. (Anderson, 2010.)

Kirjallisuudessa on esitetty useita ehdotuksia siitä, miten Kanbania voidaan soveltaa ohjelmistokehityksen yhteydessä. Tällaisia ovat esittäneet muun muassa Ladas (2008), Shalloway, Beaver ja Trott (2010) sekä Kniberg ja Skarin

(2010). Tässä luvussa Kanbania kuvataan Andersonin (2010) näkemyksen mukaisesti.

Päinvastoin kuin monissa menetelmissä, Kanbania kehotetaan kokeilemaan, muuttamaan, räätälöimään ja optimoimaan kutakin tilannetta vastaavaksi. Anderson (2010) määrittelee Kanbanille viisi keskeistä periaatetta: *visualisoi työnkulkua (*engl. *visualize workflow)*, *rajoita keskeneräisen työn määrää* (engl. *limit work-in-progress)*, *mittaa ja hallitse virtausta* (engl. *measure and manage flow*), *tee prosessikäytänteistä selkeitä* (engl. *make process policies explicit*) sekä *käytä malleja parannusmahdollisuuksien havaitsemiseen* (engl. *use models to recognize improvement opportunities)*. (Anderson, 2010.)

Työn visualisoimiseksi ja keskeneräisen työn rajoittamiseksi käytetään *kanban-taulua (*engl. *kanban board)* (kuvio 9). Tällöin tiimi näkee visuaalisesti, mitä on työn alla ja voi organisoitua itse sekä asettaa omat tehtävät ilman, että johtotason henkilö määrää sen. Tiimin tulee suostua taulun käyttöön. Työntekijöiden tulee itse sijoittaa uudet tehtävät taululle ja siirtää niitä eteenpäin työn edetessä. Ennen ensimmäistä työvaihetta tehtävä on *syöttöjonossa* (engl. *input queue)*.



KUVIO 9 Kanban-taulu (Anderson, 2010)

Kanban-taululle tulee määrittää tiimin kehitysprosessi arvoketjuna, joka pitää sisällään eri työvaiheet omina sarakkeinaan. Sarakkeina voivat olla esimerkiksi analyysi, suunnittelu, toteutus, testaus ja valmis. Sarakkeiden jälkeen voi olla oma sarake jonolle, johon työvaiheesta valmiin tehtävän voi laittaa odottamaan seuraavaan vaiheeseen vetoa. Jokaiselle vaiheelle on määrätty tietty määrä (limit) töitä, ja jokainen keskeneräinen työvaihe on omassa sarakkeessaan, eikä määrärajan yli saa tulla lisää töitä ennen aiempien työvaiheiden valmistumista. Keskeneräisen työmäärän rajoittamisen on huomattu vähentävän tehtävän *lä-*

pimenoaikaa (engl. *lead time*), eli aikaa, joka kestää tehtävän työn aloittamisesta sen valmistumiseen.

Keskeneräisen työmäärän rajoittamisen on huomattu vähentävän tehtävän *läpimenoaikaa* (engl. *lead time*), eli aikaa, joka kestää tehtävän työn aloittamisesta sen valmistumiseen. Taululle kortteina asetettavat työtehtävät voidaan luokitella työtyypin mukaan, joita voi olla esimerkiksi vaatimus, virheen korjaus, refaktorointi, muutospyyntö tai ylläpitotyö. Työn *palveluluokka* (engl. *class of service*), joka tarkoittaa tehtävän kiireellisyyttä, voidaan esittää väreillä. Esimerkiksi nopeutetut ja erittäin tärkeät tehtävät voidaan merkitä harmailla lapuilla, tiettyyn päivämäärään mennessä valmiiksi saatavat tehtävät violeteilla lapuilla, tavalliset tehtävät keltaisilla lapuilla ja ei-kiireelliset tehtävät vihreillä lapuilla. Palveluluokan lisäksi on hyödyllistä sisällyttää kuvaukseen myös tehtävän lähde, kuten kenttämyynniltä tullut pyyntö tai strategisesta suunnittelusta tullut pyyntö. Myös työn koko (esim. pieni, keskikokoinen, suuri) voi ilmetä kortista.

Pullonkaulapuskuri (engl. *bottleneck buffer*) on jono, joka laitetaan ennen työvaihetta, joka usein on pullonkaulana. Näitä voivat olla esim. vaatimukset, jotka odottavat analyysiä, analyysi, joka odottaa suunnittelua, analysoitu työ, joka odottaa kehitystä jne. *Ei-välittömien työtehtävien jonoon* (engl. *non-instant availability queue*) kasaantuvat tehtävät, jotka odottavat jotain resurssia. Esimerkiksi mikäli testiympäristöstä ja sen laitteistosta vastaava työntekijä on lähtenyt töistä eikä pysty auttamaan testiympäristön ongelmissa ennen huomista iltapäivää, on vastaava tehtävä tässä jonossa siihen asti. Jonot ja puskurit pidentävät läpimenoaikaa, mutta tasoittavat virtausta. Tärkeää Kanban-taulua räätälöitäessä on mallintaa sen työkulku vastaamaan reaalimaailman työskentelymalleja. Kehitysaktiviteettiin voidaan merkitä valintaruuduin osa-aktiviteetteja, jotka voidaan merkitä tehdyksi niiden valmistuessa. Jos työtehtävän estää jokin tekijä tehtävän jossakin vaiheessa, se tulee merkitä erikseen estetyksi ja luoda uusi tehtävä ongelman ratkaisemiseksi.

Kanbanissa on monien muiden ketterien lähestymistapojen mukaisesti erilaisia palavereita. Päivittäisissä palavereissa säännöllisesti käyvät kehittäjät näkevät taululta, miten työtehtävät ovat edistyneet edellisen palaverin jälkeen. Samalla taulu vastaa kolmeen päivittäisissä palavereissa esiintyvään kysymykseen, eli mitä teit eilen, mitä teet tänään sekä tarvitsetko apua tai onko työssä ollut ongelmia. Tällöin työtehtävät voidaan käydä lävitse oikealta vasemmalle vetojärjestyksessä. Pääpaino tarkastelussa voi olla ongelmallisten tai pitkään samassa työvaiheessa olleissa työtehtävissä. Päivittäisen palaverin jälkeen pienet 2–3 henkilön kokoiset ryhmät voivat keskustella ajankohtaisista aiheista, kuten teknisestä suunnittelusta tai mahdollisista ongelmista. Jonon täydennyspalaverit tapahtuvat usein tuoteomistajien johdolla, ja niihin osallistuu myös muita sidosryhmien henkilöitä, kuten projektipäällikkö. Näissä palavereissa päätetään, mitä tehtäviä *kehitysjonosta* (engl. *backlog)* tulee syöttöjonoon. Palaverit pyritään pitämään niin myöhään kuin vain on mahdollista, mutta kuitenkin tarpeeksi usein ja tasaisin väliajoin. Syöttöjonon koon määrittäminen riippuu siitä, kuinka usein palavereita pidetään ja uusia tehtäviä lisätään. Myös kehitysjonoa tulee pitää yllä, ja turhia työtehtäviä poistaa. Esimerkiksi puoli vuotta

kehitysjonossa olleet tehtävät voidaan usein poistaa, sillä ne harvoin ovat olennaisia, tai mikäli ne ovat, ne voidaan myöhemmin lisätä sinne uudestaan. Tehtävien priorisointi kehitysjonossa ei ole kuitenkaan tarpeellista, sillä priorisointi tapahtuu siinä vaiheessa, kun tehtävät siirretään syöttöjonoon. (Anderson, 2010.) Scrumiin ja XP:n verrattuna Kanbanin prosessimalli ei ole iteratiivinen.

Julkaisujen suunnittelupalavereissa käydään läpi toimitettava ohjelmisto, ja niissä tulisi olla läsnä projektipäällikön lisäksi muita sidosryhmiä. Tuloksena syntyvä julkaisusuunnitelma pitää sisällään muun muassa sen, mitä valmiita osia julkaisussa tulee olemaan ja mitä riskejä julkaisuun liittyy. Jokaiseen ohjelmiston julkaisuun liittyy kustannuksia, kuten markkinointi- ja koulutuskustannukset. Säännöllisin väliajoin tulevat versiot lisäävät luottamusta ja vähentävät koordinointikuluja. (Anderson, 2010.)

Taulukossa 3 on esitetty tiivistelmä Kanbanin piirteistä Andersonin (2010) esityksen mukaisesti.

TAULUKKO 3 Kanban jaettuna osiin

Menetelmän osa	**Kanban**
Tausta	2000-luvulla teollisuustuotantoympäristöstä ohjelmistokehitykseen tuotu menetelmä, jonka juuret ovat lean-periaatteissa
Lähestymistapa	Kevyt (lean) lähestymistapa Empiirinen lähestymistapa
Periaatteet	Kaizen-kulttuurin periaatteet, työnkulun visualisointi, keskeneräisen työmäärän rajoittaminen, virtauksen hallitseminen ja mittaus, prosessikäytäntöjen selkeyttäminen, parantamismahdollisuuksien havaitseminen
Soveltaminen	Soveltuu erilaisille organisaatioille, mutta vaatii räätälöintiä ja optimointia
Prosessi	Ei esitä omaa prosessimallia, vaan suosittelee Kanbanin soveltamista jonkun muun prosessin mukaisesti
Roolit	Olemassa olevan organisaation ja kehitysprosessien mukaiset roolit
Käytäntöjä	Kanban-taulu, keskeneräisen työmäärän rajoittaminen, syöttö- ja kehitysjono, palveluluokka, mittarit

Kanbanin mukaista toimintaa voidaan mitata erilaisilla mittareilla. Kehitysjonon kokoa, keskeneräistä työtä sekä julkaistuja tehtäviä voidaan mitata, jolloin voidaan varmistua siitä, että Kanbanin käyttö sujuu hyvin. Keskimääräinen läpimenoaika kertoo, kuinka hyvin organisaatio keskimäärin suorittaa tehdyt työtehtävät. Läpimenoaikaa voidaan erikseen mitata esimerkiksi ominaisuuksille ja virheiden korjauksille. *Erääntyneen päivämäärän* (engl. *due date*) avulla voidaan selvittää, kuinka suuri osa tehtävistä saadaan valmiiksi niille oletetussa ajassa. Suoritustehoa voidaan mitata sillä, kuinka monta tehtävää tiimi saa valmiiksi per kuukausi. Alustavaa laatua voidaan mitata jakamalla esiintyneiden vikojen määrä kehitetyillä ominaisuuksia. Ongelmien ja estyneiden työtehtävi-

en määrää mittaamalla voidaan tutkia, kuinka hyvin ja nopeasti ne pystytään ratkaisemaan niiden esiinnyttyä. *Virtauksen tehokkuuden* (engl. *flow efficiency*) mittausta voidaan tehdä katsomalla, kuinka paljon aikaa työntekijät käyttävät yksittäisen tehtävän suorittamiseen suhteessa siihen, mitä se viettää estyneenä tai jonoissa. *Virhekuormaa* (engl. *failure load*) voidaan taas mitata tutkimalla, kuinka suuri osuus työtehtävistä johtuu aiemmasta huonosta laadusta. Näitä ovat esimerkiksi vikojen korjaaminen tai ominaisuuden uudelleen tekeminen huonon käytettävyyden vuoksi.

3.5 Yhteenveto

Perinteisesti ohjelmistojen kehittämismenetelmät ovat olleet suunnitelmavetoisia, kun taas ketterät menetelmät ovat enemmän muutosvetoisia menetelmiä. Agile-manifestin (Agile-Alliance, 2001a) julkaisun jälkeen ketterien menetelmien käyttö on tullut 2000-luvulla yhä yleisemmäksi. Ketteryyden ja ketterän kehittämisen määritteleminen on kuitenkin ollut haastavaa. Yhteistä näkemyksille on muiden muassa se, että ketterä ohjelmistokehitys on inkrementaalista ja iteratiivista, pyrkii toimittamaan lyhyiden iteraatioiden tuloksena toimivaa koodia ja edistää eri sidosryhmien yhteistyöstä.

Tässä luvussa esiteltiin myös kolme ketterää menetelmää. Scrum keskittyy tarjoamaan tukea erityisesti ohjelmistoprojektin johtamiseen. Scrumin mukaan kehitystyö tapahtuu iteratiivisesti sprinteissä, joiden pituus on yleensä 2–4 viikkoa. Sprintin aikana tehtävät työtehtävät kootaan sprintin työlistaan, johon ne valitaan tuotteen kehitysjonosta. XP taas tarjoaa toimintaohjeita erityisesti ohjelmiston kehittäjille. Keskeisenä osana menetelmää ovat käytänteet kuten jatkuva integrointi, pariohjelmointi, yksinkertainen suunnittelu ja suunnittelupeli. Menetelmän prosessimalli on Scrumin tapaan iteratiivinen. Kanbanin periaatteet perustuvat Lean-ajattelutapaan, jonka juuret ovat autoteollisuudessa. Poiketen XP:stä ja Scrumista, Kanbanissa ei käytetä määrämittaisia kehittämissyklejä. Roolien ja käytänteiden osalta se pyritään sopeuttamaan organisaatiossa jo olemassa olevaan prosessiin. Tärkeitä periaatteita Kanbanissa ovat muun muassa työn visualisointi Kanban-taulua apuna käyttäen sekä keskeneräisen työmäärän rajoittaminen.

4 KETTERÄN MENETELMÄN RÄÄTÄLÖINTIÄ KOSKEVIA EHDOTUKSIA JA OHJEITA

Edellisissä luvuissa on tarkasteltu yhtäältä menetelmiä ja niiden räätälöinti yleisesti ja toisaalta ketteriä menetelmiä. Tässä luvussa yhdistetään tarkastelukohteet kuvaamalla ketterien menetelmien räätälöintiä koskevia ehdotuksia ja ohjeita. Ensiksi kerrotaan ketterien menetelmien räätälöinnistä yleisesti ja valitaan tarkempaa selvitykseen otettavat tutkimukset. Tämän jälkeen kuvataan näitä tutkimuksia. Lopuksi esitetään yhteenveto ja vertailu ketterien menetelmien räätälöintiä koskevista esityksistä.

4.1 Ketterien menetelmien räätälöimisestä yleisesti

Ketterien menetelmien alkuvaiheessa 2000-luvulla tutkimuksen ja keskustelun kohteena olivat lähinnä menetelmien määritteleminen sekä niiden käytöstä saadut empiiriset tulokset (Mnkandla & Dwolatzky, 2007). Sen sijaan ketterien menetelmien räätälöintiä koskevia tutkimuksia oli vähän (Aydin ym., 2005; Fitzgerald, Hartnett & Conboy, 2006). Kuitenkin vain harva organisaatio voi ottaa ketterän menetelmän, kuten XP:n, käyttöön sellaisenaan (Pikkarainen & Passoja, 2005), vaikka alkuvaiheessa jotkut menetelmistä vaativatkin, että niitä tulisi käyttää kokonaisuudessaan, jotta eri käytänteistä syntyvät synergiahyödyt realisoituisivat. Esimerkiksi Beck (1999) sanoo XP:stä, että sen käytänteet eivät toimi yksinään hyvin, vaan vaativat muutkin käytänteet pitämään sen tasapainossa. Tämä on kuitenkin ketteryyteen liittyvän joustavuuden vastaista. (Ågerfalk & Fitzgerald, 2006.)

Cockburnin ja Highsmithin (2001) mukaan täsmälliset (strict) prosessit on suunniteltu standardoimaan toimintaa, kun taas ketterät prosessit on suunniteltu hyödyntämään jokaisen yksilön ja kehitystiimin yksilöllisiä voimavaroja. Heidän mukaansa prosessi tulee valita, räätälöidä ja sopeuttaa kunkin projektitiimin tarpeita silmällä pitäen. Ketteriä menetelmiä onkin usein tarve räätälöidä samalla lailla kuten perinteisiä menetelmiä, ja joskus ketterien menetelmien rää-

tälöimiselle on nähty olevan jopa suurempi tarve. Muun muassa Keenan (2004) toteaa, että räätälöinti on edellytys ketterien menetelmien käytölle.

Ketteriä menetelmiä on kuitenkin kritisoitu siitä, että ne tarjoavat harvoin itse apua siihen, miten räätälöintiä tulisi suorittaa (Abrahamsson ym., 2003). Conboy ja Fitzgerald (2010) toteavat, että ketteristä menetelmistä ainoastaan Crystal (Cockburn, 2001) ottaa tilannetekijöitä huomioon ja tarjoaa apua räätälöintiin. Anderson (2010) kuitenkin määrittelee Kanbanin yleisesti räätälöitäväksi, sillä Kanban ottaa huomioon organisaation nykyiset prosessit, ja Anderson (2010) tarjoaa apua Kanbanin räätälöimiselle muun muassa isoille organisaatioille. On kuitenkin jopa sanottu, että on tarve uusille ketterille menetelmille, jotka tarjoaisivat joustavuutta menetelmän soveltamisessa tietyn projektin käyttöön eli projektikohtaisessa räätälöinnissä. Projektin alussa tehtävä päätös käytettävästä ketterästä menetelmästä tekee muuten mahdolliset isot muutokset projektin edetessä mahdottomiksi. (Henderson-Sellers & Serour, 2005.) Henninger ym. (2002) kuitenkin toteaa, että lähestymistapaa voi muuttaa myös projektin aikana. Esimerkiksi projektin alussa voidaan käyttää Scrumia hyödyksi, mutta myöhemmin voidaan vaihtaa käyttämään XP:ä ja sen käytänteitä, tai tarvittaessa myös perinteisiä menetelmiä (Henninger ym., 2002).

Ketterän menetelmän räätälöinti on todettu ongelmalliseksi myös käytännön ohjelmistokehityksessä. Kyselyssä irlantilaisille organisaatioille ketterän menetelmän räätälöinti paljastui ongelmaksi. Sen syiksi mainittiin muun muassa tietämyksen puute räätälöinnistä ja aiemmat epäonnistuneet kokemukset. (Conboy & Fitzgerald, 2010.)

Ketteriä menetelmiä ottaessa käyttöön ja räätälöitäessä on hyvä huomata, että niiden omaksumisessa organisaatiossa voi mennä pitkä aika useista tekijöistä johtuen (Qumer & Henderson-Sellers, 2008). Organisaatio voi laajemmin muodostaa tähän myös muutosstrategian, joka ottaa huomioon kyseiset tilannetekijät (Gandomani, Zulzazil, Abd Ghani & Sultan, 2013). Ketteriä käytänteitä voi myös reflektoida organisaatiossa tietyin väliajoin, jolla voidaan parantaa kehittämisprosesseja ja kehittäjien kykyjä (Salo & Abrahamsson, 2007). Pitkällä tähtäimellä ketterien menetelmien käyttämiseen vaikuttavat useat tekijät, kuten kouluttautuminen, ketterä ajattelutapa, asenne, motivaatio ja tekninen kompetenssi (Senapathi & Srinivasan, 2013).

Ketterien menetelmien räätälöimisen helpottamiseksi on viime vuosina tehty joitakin ehdotuksia ja ohjeita. Tutkimuksia ei kuitenkaan ole erityisen paljoa, ja ne kärsivät osittain laatuongelmista. Akbar, Hassan ja Abdullah (2011) toteavat, että tutkimusten tapa käsitellä ketteryyttä, prosessien parantamista ja räätälöimistä liikkuu yleisellä tasolla. Tutkimuksia on kritisoitu myös puutteellisista tutkimusmenetelmistä sekä siitä, ettei kehitettyjen viitekehyksien ja mallien soveltuvuutta ole osoitettu käytännössä. Räätälöinnin lisätutkimuksille niin käytännön soveltajien kuin tutkijoiden taholta onkin nähty tarve, samoin kuin uusille formaaleille lähestymistavoille, viitekehyksille, menetelmille ja standardeille. (Akbar ym., 2011.)

Ketterien menetelmien räätälöintiä koskevien tutkimusten kartoittamiseksi suoritettiin tätä tutkimusta varten kirjallisuushakuja erilaisiin tietokantoihin.

Kirjallisuutta haettiin pääasiassa Googlen ja Google Scholarin avulla sekä Nelliportaalin kautta käyttämällä eri tietokantoja, joita olivat muun muassa Springer Link, IEEE Xplore, ScienceDirect, IGI Global ja ACM Digital Library. Hakutermeinä käytettiin muun muassa sanoja ”situational”, ”agile”, ”method”, ”methodology”, ”tailoring”, ”customization”, ”adaptation”, ”configuration” ja “deploying”. Löydetyistä lähteistä suoritettiin valinta tietyin kriteerein. Tutkimuksen laatu ja laajuus huomioitiin kuten myös julkaisun ajankohtaisuus. Tutkimuksia on pyritty käsittelemään etenkin niiden käytännön ohjelmistokehitykseen tarjoamien ohjeiden näkökulmasta. Hakutuloksien joukosta valikoitui myös ketterän menetelmän räätälöintiä käsitteleviä tapaustutkimuksia, joita käsitellään luvussa 5.

Tässä luvussa keskitytään sellaisiin tutkimuksiin, joissa annetaan yleisiä ohjeita ja ehdotuksia ketterien menetelmien räätälöintiin. Näissä tutkimuksissa aiheet, tutkimusotteet ja -menetelmät vaihtelevat jonkin verran. Esimerkiksi Boehmin ja Turnerin (2004) esityksen voidaan katsoa olevan ohjeellinen neuvo myös ketterän menetelmän räätälöintiin, vaikka tutkimuksessa käsiteltiin myös perinteisiä menetelmiä. Myös Henninger ym. (2002) esittelevät ns. ”hybridin” lähestymistavan ketterän menetelmän räätälöintiin. Muita tutkimuksia viitekehyksistä, malleista ja menetelmistä ketterän menetelmän räätälöintiin ovat tehneet muun muassa Keenan (2004), Aydin ym. (2005), Mnkandla ja Dwolatzky (2007), El-Said, Hana ja Eldin (2008), Mirakhorli ym. (2008), Qumer ja Henderson-Sellers (2008), Cao ym. (2009), Karlsson ja Ågerfalk (2009), Conboy ja Fitzgerald (2010), Mikulenas ja Kapocius (2011), Mikulėnas, Butleris ja Nemuraitė, Ayed ym. (2012) sekä Baskerville ja Pries-Heje (2013). Myös käytännön työhön suunnattuja oppaita ketterän menetelmän käyttöönottamisesta ja räätälöinnistä on tehty, kuten muun muassa Koch (2005) ja Moreira (2013).

Tässä luvussa tarkasteltaviksi valittiin edellä mainituista yhdeksän esitystä. Seuraavassa kerrotaan valintaperusteet kunkin osalta. Aydin ym. (2005) esitys jakaa räätälöinnin ennen projektia tapahtuvaan staattiseen sekä projektin aikana tapahtuvaan dynaamiseen räätälöintiin. Se valittiin, sillä esitys käsittelee hallitusti tilannetekijöitä, jotka ovat selvillä ennen projektia tai ilmenevät vasta projektin aikana. Ayedin ym. (2012) tutkimus valittiin sen vuoksi, että se tarjoaa ylemmän tason metamallin ketterän menetelmän räätälöinnin tueksi, vaikkei se suoranaisesti tarjoa ohjeita menetelmän räätälöintiin.

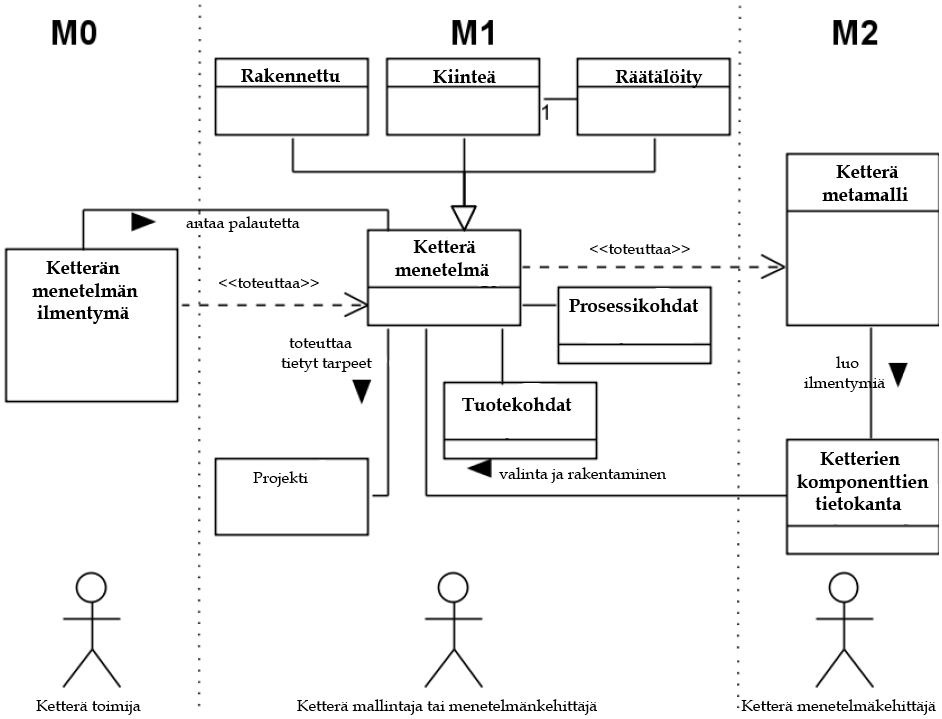
Cao ym. (2009) käsittelevät ketterän menetelmän räätälöintiä AST-teorian pohjalta ja poikkeaa siten muista ketterän menetelmän räätälöintiä käsittelevistä tutkimuksista. Conboy ja Fitzgerald (2010) taas antavat konkreettisia ohjeita XP:n räätälöintiin, joita voidaan hyödyntää räätälöintiprosessin aikana. Ohjeita voi soveltaa myös muita ketteriä menetelmiä räätälöitäessä. Karlsson ja Ågerfalk (2009b) sekä Mikulenas ja Kapocius (2011) käsittelevät ketterän menetelmän räätälöintiä tilannekohtaisen menetelmäkehityksen näkökulmasta. Heidän esityksensä tarjoavat konkreettisia ohjeita ja prosessin räätälöintiin.

Luin ja Chanin (2005) sekä Mirakhorlin ym. (2008) esitykset käsittelevät XP:n räätälöintiä yksityiskohtaisella tasolla. Ne täydentävät sitä yleistä kuvaa ketterien menetelmien räätälöinnistä, jota muut tutkimukset tarjoavat. Qumerin

ja Henderson-Sellersin (2008) tutkimus ei suoranaisesti käsittele ketterän menetelmän räätälöintiä tarjoamalla siihen prosessia, jota olisi mahdollista käyttää. Tutkimus kuitenkin valittiin käsiteltäväksi, sillä se on kypsyysmalliajatteluun pyrkivän lähestymistavan johdosta mielenkiintoinen, ja se tarjoaa näkökulman pitkäaikaiseen ja hallittuun ketterän menetelmän omaksumiseen organisaatiotasolla.

4.2 Räätälöinti käyttämällä metamallia hyväksi

Ayed ym. (2012) ovat tutkineet ketterän menetelmän räätälöintiä metamallintamista käyttäen. He toteavat, että vaikka ketteriä menetelmiä on paljon, niillä on yhteinen paradigma. Niitä voidaan myös kuvailla metamallin avulla, ja vielä yleisemmällä tasolla geneerisen prosessimetamallin avulla. Ayed ym. (2012) ovat esittäneet tilannelähtöistä menetelmäkehitystä toteuttavan lähestymistavan (kuvio 10), jonka avulla voidaan tarkastella ketterän menetelmän räätälöintiä.



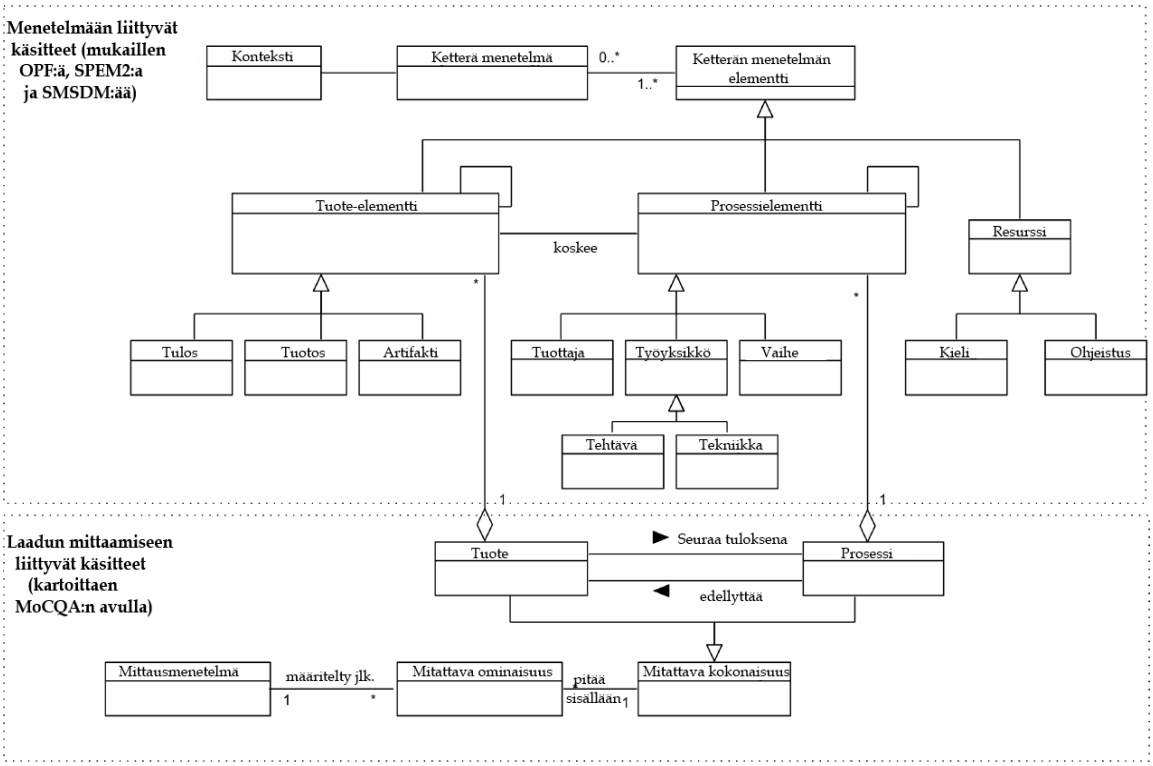
KUVIO 10 Lähestymistapa ketterän menetelmän räätälöintiin (Ayed ym., 2012, 68)

Ayed ym. huomauttavat, että vaikka on tehty paljon tutkimusta yleisesti menetelmien tai perinteisten menetelmien metamallintamisesta, tutkimusta ketterien menetelmien metamallintamisesta on tehty paljon vähemmän. Esimerkki tällaisesta on Damianin, Colombon, Fratin sekä Bellettinin (2007) tutkimus, jossa käsitellään Scrumia metamallintamisen näkökulmasta. Mikulėnas, Butleris ja Nemuraitė (2011) taas esittävät metamallin osittaiseen ketterän menetelmän räätä-

löintiin, jonka avulla osia voidaan jakaa elementteihin ja niitä voidaan yhdistää ja yleistää. Heidän mallinsa tukee räätälöintiä ja päätöksentekoa menetelmää mallintaessa, mutta ei parannettaessa itse kehitystyön aikana.

Ayed ym. (2012) ovat käyttäneet hyväkseen näitä esityksiä ketterän menetelmän räätälöinnin lähestymistavan luomisessa. Lähestymistavassa erotetaan kolme tasoa, M0, M1 ja M2. M0 vastaa ketterää menetelmää käyttötilanteessa eli ketterän menetelmän ilmentymää (instance). M1 vastaa ketterän menetelmän kuvausta. Kuvaus osoittaa, mille projektille se on tarkoitettu, mistä prosesseista se koostuu ja mitä tuotoksia sen mukaan tuotetaan. Menetelmä voi olla kiinteä (fixed), rakennettu tietyistä osista tai räätälöity jostakin olemassa olevasta menetelmästä. M2 vastaa metamallitasoista esitystä menetelmästä. Metamallia suunnittelee menetelmäkehittäjä. Alemmat tasot toteuttavat ylemmän tason määrittelyjä sekä tarjoavat palautetta kehittämiseen.

Ayedin ym. (2012) ketterä metamalli on esitetty tarkemmin kuviossa 11. Se mukailee muita prosessimetamalleja, joita on tehty. Näitä ovat Open Process Framework (OPF, OPEN Process Framework Repository Organization, 2009), Software & Systems Process Engineering Metamodel (SPEM, Object Management Group, 2008) ja Standard Metamodel for Software Development Methodologies (SMSDM, Henderson-Sellers & Gonzalez-Perez, 2005).



KUVIO 11 Ketterän menetelmän räätälöinnin metamalli (Ayed ym., 2012, 69)

Kuvion mukaan ketterä menetelmä koostuu useista elementeistä, jotka voivat olla tuote-elementtejä, prosessielementtejä ja resursseja. Tuote-elementti on jotakin, joka tuotetaan projektin aikana. Se voi olla tulos (outcome), artifakti tai tuotos (deliverable). Tuote-elementtejä ovat muun muassa tekniset dokumentit,

suunnitelmat ja julkaisut. Prosessielementti taas voi olla työn tekijä, eli tuottaja, vaihe tai työyksikkö (work unit). Prosessielementtejä ovat muun muassa roolit (esim. testaaja), iteraatiot, tehtävät ja tekniikat. Resurssi taas viittaa johonkin, jota käytetään projektin loppuunsaattamiseen, kuten kieliä (esim. ohjelmointikieliä) ja ohjeistuksia.

Menetelmään liittyvien käsitteiden lisäksi metamalli pitää sisällään laadun mittaamiseen liittyviä käsitteitä, jotka perustuvat Model-Centric Quality Assessment -viitekehykseen (MoCQA, Vanderose, Kamseu & Habra, 2010). Käytettävää menetelmää ja sen sisältämien tuote- ja prosessielementtien laatua tulee jatkuvasti arvioida. Mitattava kokonaisuus on esimerkiksi elementti, ja se pitää sisällään mitattavia ominaisuuksia. Mittaus suoritetaan jotain mittausmenetelmää käyttäen. (Ayed ym., 2012.)

4.3 Räätälöinti ketterien käytäntöjen kustannus-arvo -suhteita vertaamalla

Mikulenas ja Kapocius (2011) ovat tutkineet erilaisia priorisointimenetelmiä ketterien menetelmien räätälöimisen tukemiseksi. Tutkimuksen lähestymistapa on deduktiivinen. Heidän mukaansa ketterät menetelmät on rakennettu neljänlaisista elementeistä: vaiheista (esim. jaksot, virstanpylväät ja versiot), työyksiköistä (esim. prosessit, tekniikat ja toimet), tuottajista (esim. ihmiset, tiimit ja työkalut) sekä työn tuloksista (esim. dokumentit, mallit ja ohjelmiston osat). Heidän mielestään menetelmätasoisen vertailun sijasta tulisi kiinnittää huomiota siihen, mitkä elementit sopivat parhaiten räätälöitävään menetelmään. Siinä tarvitaan priorisointia.

Tutkimuksessa tarkastellaan sitä, miten tavallisesti vaatimusmäärittelyn apuna käytetyt priorisointitekniikat soveltuvat ketterien menetelmien räätälöintiin yhteyteen tehtäessä valintoja siitä, mitä käytäntöjä (elementtejä) menetelmään sisällytetään. Esimerkiksi Top Ten -vaatimukset -tekniikka (Lauesen, 2002) toimii siten, että eri viiteryhmät valitsevat joukosta kymmenen sopivinta käytäntöä. Se sopii etenkin silloin, jos sidosryhmien välillä on erimielisyyksiä käytänteistä, mutta ovat muuten tasa-arvoisia toisiinsa nähden. Toisessa tavassa numeroidaan käytännöt yhdestä alkaen käytäntöjen määrään asti (Karlsson, Wohlin & Regnell, 1998), jolloin ensimmäinen on tärkein ja viimeinen on vähiten tärkeä käytäntö. Numerointiin voi liittää halutessaan prioriteetin sanallisesti arvioituna, kuten kriittinen, tavallinen tai valinnainen. Kyseisten käytäntöjen kanssa on ongelma, etteivät niiden väliset suhteelliset erot ole selviä, ja sanalliset kuvaukset voivat hämmentää sidosryhmiä. (Mikulenas & Kapocius, 2011.)

Toisenlaiset tekniikat lähestyvät aihetta formaalisemmin keinoin, ja ne ottavat myös käytänteiden suhteellisen eron huomioon. Niin sanotussa Sadan dollarin testissä (Leffingwell & Widrig, 1999) sidosryhmät saavat sata kuvitteellista yksikköä, kuten rahaa tai tunteja, käyttöönsä jaettavaksi eri vaihtoehtojen välille. Tässä tekniikassa on ongelmana se, että siinä käytetään ainoastaan yhtä

kriteeriä arviointiin. AHP-tekniikassa (Analytic Hierarchy Process) (Regnell, Höst, Natt och Dag, Beremark & Hjelm, 2001; Saaty, 2007) arvioidaan eri vaihtoehtoja käyttämällä hyväksi useampia kriteerejä. Sen etuna on se, että käytänteiden välistä tehokkuutta olemassa olevien ristiriitaisten tavoitteiden ratkaisemiseksi voidaan arvioida. AHP-tekniikan ongelmana on kuitenkin se, että käytäntöjen määrän kasvaessa suureksi se on kovin työläs käyttää. (Mikulenas & Kapocius, 2011.)

Koska ketterät menetelmät korostavat projektitiimien työtä, tulisi Mikulenaksen ja Kapociuksen (2011) mielestä tiimin kaikkien jäsenten myös osallistua käytänteiden priorisoimiseen. He esittävät oman ehdotuksensa priorisointitekniikaksi, joka pitää sisällään kuusi vaihetta. Ensimmäisessä vaiheessa tunnistetaan kohdealueen ongelma-alueet ja ongelmat, joiden ratkaisemiseen myöhemmissä vaiheissa tullaan keskittymään. Toisessa vaiheessa etsitään joukkoa sopivia ketteriä käytänteitä erilaisista ketteristä menetelmistä. Etsiminen voidaan suorittaa esim. aivoriihi-tekniikalla (brainstorm). Kolmas vaihe pitää sisällään monia alavaiheita. Siinä keskitytään käytänteiden arviointiin sen jälkeen, kun lista erilaisista käytänteistä on saatu valmiiksi. Tämän jälkeen tekniikoita verrataan toisiinsa niiden tuottaman arvon mukaisesti kokemukseen perustuen. Kahden käytänteen ollessa samanarvoisia tulee niiden väliseksi numeroksi yksi, kun taas mikäli toinen tekniikka on todella paljon arvokkaampi kuin toinen, tulee arvoksi yhdeksän. Arvosteluasteikko toimii numeerisesti siis välillä 1–9. Tämän jälkeen tulokset taulukoidaan matriisiin, jolloin jokainen rivi sekä sarake vastaavat yhtä käytännettä. Esimerkki matriisista on esitetty taulukossa 4.

TAULUKKO 4 Käytänteiden arvojen vertaaminen matriisilla

	K1	K2	K3
K1	1	1/3	1/2
K2	3/1	1	4/1
K3	2/1	1/4	1

Kun käytänteiden vertailusta on saatu taulukko, lasketaan rivien perusteella vertailuarvot kullekin käytänteelle. Taulukossa 5 on esimerkki yllä mainituilla arvoilla lasketuista vertailtavista tuloksista.

TAULUKKO 5 Vertailukelpoiset tulokset käytänteiden arvoille suhteessa toisiinsa

	K1	K2	K3
K1	0.166	0.211	0.09
K2	0.5	0.638	0.72
K3	0.333	0.158	0.181

Laskeminen tapahtuu laskemalla ensin jokaisen sarakkeen summa. Yllä mainitusta taulukon neljä arvoilla K1:lle tulee arvo 6, K2:lle 1,583 ja K3:lle 5,5. Tämän

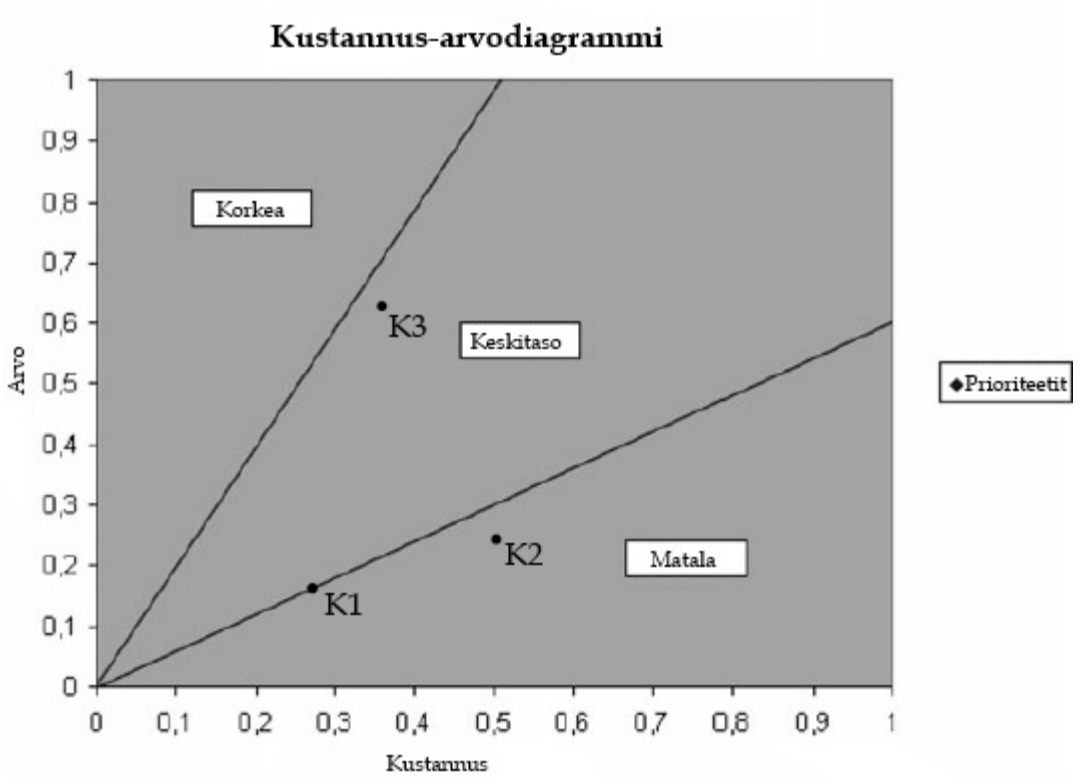
jälkeen jokainen yksittäinen matriisin solu jaetaan kyseisen sarakkeen summalla, johon solu kuuluu.

Lopullinen priorisointimatriisi saadaan laskemalla jokaisen rivin alkiot yhteen ja jakamalla se alkion määrällä. Taulukko 6 pitää sisällään lopulliset priorisointimatriisin. Lopuksi käytänteet asetetaan järjestykseen niiden tuottaman arvon mukaisesti.

TAULUKKO 6 Lopulliset vertailukelpoiset tulokset käytänteiden arvoille

K1	K2	K3
0.156	0.619	0.224

Neljäs vaihe pitää sisällään samat toimenpiteet kuin kolmas vaihe, mutta sen sijaan että keskityttäisiin arvioimaan käytäntöjen arvoa, arvioidaan käytänteitä toisiinsa nähden niiden käytön aiheuttamia kustannuksia vertailemalla. Tämän jälkeen jokaiselle käytännölle on saatu muodostettua arvo väliltä 0 ja 1, jonka jälkeen ne voidaan asentaa kustannus-arvodiagrammille. Olettaen, että aiemmin mainittujen käytänteiden summaksi olisi saatu K1:lle 0,275, K2:lle 0,365 sekä K3:lle 0,502., käytänteet näyttäisivät kustannus-arvodiagrammilla kuvion 12 osoittamalla tavalla.



KUVIO 12 Kustannus–arvodiagrammi (Mikulenas & Kapocius, 2011, 491)

Arvojen määrittämisen jälkeen voidaan käydä keskustelua ketteristä käytänteistä ja valita ne, joita tullaan käyttämään räätälöitynä menetelmänä. Kustannus-

arvodiagrammi voi toimia keskustelun pohjana ja ohjata kehitystiimiä valitsemaan käytänteitä, joilla on mahdollisimman hyvä kustannus–arvo -suhde.

4.4 Räätälöinti käyttämällä MMC-menetelmää

Karlsson ja Ågerfalk (2004; 2005; 2009a) ovat esitelleet *menetelmän menetelmän räätälöimiseksi* (engl. *Method for Method Configuration, MMC)*. Heidän ensimmäiset tutkimuksensa ovat keskittyneet perinteisten menetelmien räätälöinnin tarkasteluun. Uusimmassa tutkimuksessa Karlsson ja Ågerfalk (2009a; 2009b) tarkastelevat, kuinka menetelmä sopii tukemaan ketteriä arvoja ja tavoitteita. Karlsonin ja Ågerfalkin tutkimuksen lähestymistapa on deduktiivinen. MMC on itsessään tilannekohtaisen menetelmäkehityksen menetelmä.

MMC:n keskeinen ajatus on ottaa jokin menetelmä niin sanotuksi pohjamenetelmäksi (base method). Esimerkiksi organisaation käyttämä menetelmää voidaan ottaa projektikohtaisten menetelmien räätälöinnin lähtökohdaksi. Pohjamenetelmää räätälöitäessä tulisi pyrkiä saamaan aikaan menetelmä, joka ottaa huomioon tilannekohtaiset ominaispiirteet, mutta samaan aikaan säilyttää pohjamenetelmän perusarvot ja -tavoitteet. Muutoin menetelmän ydinperiaatteet voivat hävitä räätälöinnin yhteydessä. Tämän vuoksi MMC:n avulla räätälöitäessä ketterää menetelmää tulisi pyrkiä säilyttämään menetelmän ja Agile-manifestin periaatteita.

MMC:n tavoite on systematisoida ja suunnitella räätälöinti oman menetelmän kautta tuottaen samalla uudelleenkäytettäviä osia. Ylemmällä tasolla tavoitteena on noudattaa erilaisia suunnitteluperiaatteita, joiden avulla voidaan luoda joustava menetelmä. Ensimmäinen periaate on modularisaation periaate. Se tarkoittaa sitä, että menetelmä koostuu moduuleista, jotka ovat itsenäisiä ja sisäisesti johdonmukaisia. Toinen periaate on perustelun periaate, eli valitut menetelmäosat tulee analysoida ja perustellusti tehdä valinnat niiden käyttämisestä. Kolmas periaate on tukea uudelleenkäyttämistä. Neljäs periaate on vuorovaikutuksen periaate, jolloin räätälöinti tapahtuu yhdessä menetelmäekspertin (method engineer) ja menetelmän käyttäjien kanssa.

MMC:n osat ovat *menetelmäkomponentteja* (engl. *method component)*, joita voidaan pitää samankaltaisina aiemmin luvussa 2 mainittujen menetelmälohkojen ja -palojen kanssa. Menetelmäkomponentti pitää sisällään itse komponentin sisällön sekä komponentin rajapinnan. Sisältö on kooste alemman tason elementeistä, pitäen sisällään syötteen, tuotoksen, näiden muuntoprosessin sekä *menetelmän rationaliteetin* (engl. *method rationale*). Sisällön elementit rakentuvat *määritellyistä toimista* (esim. määrittele käyttäjätarina), *käsitteistä* (esim. tehtävä), *notaatioista* (esim. tekstimuotoinen), *artefakteista* (esim. käyttäjätarina) sekä *aktoreiden rooleista* (esim. asiakas).

Menetelmäkomponentin rajapinnan tarkoituksena on piilottaa ei-olennaiset yksityiskohdat komponentin sisältä menetelmän räätälöinnin aikana. Toimintaperiaate on siten samanlainen kuin ohjelmistokehityksessä käytettävillä komponenttien rajapinnoilla. Räätälöinnin aikana ollaan kiinnostuneita mitä

syötteitä komponentti tarvitsee ja mitä tuloksia se saa aikaan, eikä siitä, miten se toimensa suorittaa. Rajapinta toimii ulkoisena näkymänä komponentista. (Karlsson & Ågerfalk, 2009b.)

Menetelmäkomponenttien valinnat perustuvat menetelmän rationaliteettiin, ja ne on tehty projektin erityispiirteiden mukaisesti. Valinnat voidaan esittää räätälöintipaketteina (configuration package). Räätälöintipakettiin voidaan ottaa osia myös muista menetelmistä, kuin pohjamenetelmästä. Räätälöintimalli (configuration template) on laaja, yhdistetty menetelmä, joka pitää sisällään pohjamenetelmän. Se voi koostua useammasta räätälöintipaketista. Karlsson ja Ågerfalk (2009b) ovat luoneet MMC:n avuksi MC Sandbox –nimisen työkalun, joka auttaa räätälöinnin suorittamisessa. Sen avulla voidaan tallentaa räätälöintipaketteja ja -malleja, joita voidaan käyttää hyödyksi tulevaisuudessa.

MMC ei ole ketterän menetelmän räätälöintiin erityisesti suunniteltu menetelmä. Sitä on kuitenkin sovellettu myös ketterien menetelmien kanssa. Karlsson ja Ågerfalk (2009b) tutkivat sen käyttöä kolmen eri projektin yhteydessä, joissa käytettiin XP:ä pohjamenetelmänä. Käytänteet ja niiden rationaliteetti pystyttiin sovittamaan MMC:n avulla yhteen. Esimerkiksi RUP:n liiketoimintanäkymää (engl. business vision) käytettiin saamaan yleiskuva liiketoiminnasta, ja se sopi hyvin XP:n tapaan toimia ollen sopiva tervetulleiden muuttuvien vaatimuksien tavoitteeseen (periaate 3). MMC:n ja sen käsitteellisen viitekehyksen huomattiin olevan mahdollista sovittaa XP:n sekä ketterien arvojen ja tavoitteiden kanssa hyvin yhteen. Koko projektitiimi osallistui räätälöintiin ja samalla pystyttiin tekemään laadunvarmistusta menetelmästä. Näin myös menetelmän rationaliteetti tuli täsmällisesti ilmi. Projektissa huomattiin tarve räätälöidä XP-menetelmää lisäämällä siihen erilaisia komponentteja sen sijaan, että niitä otettaisiin pois. Tämä johtuu osin siitä, että XP on varsin kevyt menetelmä, eikä se tarjonnut tukea tarvittaville osille.

4.5 Räätälöinti AAIM:n avulla

Qumer ja Henderson-Sellers (2008) ovat esitelleet ketterän ohjelmistokehityksen viitekehyksen. Tutkimuksen lähestymistapa perustuu kypsyysmalliajatteluun. Viitekehys pitää sisällään ylemmällä abstraktiotasolla käsitteellisen mallin ketterän menetelmän ytimestä, hallinnasta, tietämyksestä sekä näiden abstraktiosta sekä yhteydestä liiketoimintaan. Tämän lisäksi viitekehykseen kuuluu liiketoiminta, ohjelmistoteknologia, ketterä työkalusarja sekä neljäkohtainen analysointityökalu (4-DAT), jolla voidaan arvioida menetelmiä eri näkökulmista. Viitekehyksen yhteyteen on kehitetty ketterä omaksumis- ja kehittämismalli (Agile Adoption and Improvement Model, AAIM), joka kokoaa yhteen olemassa olevaa tietämystä ja käsitteitä niin teoriasta kuin käytännön ohjelmistokehityksestä. AAIM on räätälöivästä ketterästä menetelmästä riippumaton malli ja tarjoaa tiekartan (roadmap) siirtymiselle ketterään organisaatioon. (Qumer & Henderson-Sellers, 2008.)

AAIM on eräänlainen kypsyysmalli, joka koostuu kuudesta vaiheesta, jotka on jaettu kolmeen lohkoon. Jokaisessa lohkossa voidaan käyttää 4-DAT -työkalua senhetkisen ketteryyden arvioimiseen (Qumer & Henderson-Sellers, 2006a; Qumer & Henderson-Sellers, 2006b). Kun on saavutettu vaiheessa määritelty ketteryyden taso, voidaan siirtyä seuraavalle tasolle, mikäli kyseisen portaan ketteryys on tarpeeksi korkealla tasolla. Mallin tarkoituksena on helpottaa ketterän menetelmän käyttöönotto, arviointia ja parantamista. (Qumer & Henderson-Sellers, 2008.)

Ensimmäinen lohko, *johdattelu (prompt)*, koostuu yhdestä vaiheesta, eli *alkuajasta (infancy)*. Kyseisessä vaiheessa ei oteta käyttöön mitään varsinaista ketterää menetelmää, vaan esitellään ja vahvistetaan ketteriä ominaisuuksia, kuten nopeutta, reagoivuutta ja joustavuutta, olemassa olevassa ohjelmistonkehitysprosessissa.

Toinen lohko, *ydinkohta (crux)*, koostuu kolmesta vaiheesta, jotka ovat *varhaisvaihe (initial), realisaatiovaihe (realization)* ja *arvovaihe (value)*. Lohkossa keskitytään vakiinnuttamaan valitun ketterän menetelmän käytänteitä ja ominaisuuksia, jotka erottavat ne perinteisistä menetelmistä. Varhaisvaiheessa keskitytään viestinnän ja yhteistyön mahdollistamiseen luomalla hyvät mahdollisuudet viestintään ja yhteistyöhön niin organisaation sisällä kuin myös ulkopuolelle asiakkaiden ja muiden keskeisten sidosryhmien kanssa. Realisaatiovaiheessa pääpaino on toimivien artefaktien luomisessa minimaalisella dokumentaatiolla. Dokumentaation sijaan pääasiallinen viestinnän keino on kasvokkain ja verbaalisesti tapahtuva kommunikointi. Ydinkohdan viimeisessä vaiheessa käytänteet ovat vakiintuneet ja keskittyvät arvostamaan niin kehittäjiä kuin myös asiakkaita. Kehittäjillä tulisi olla myös vapautta tehdä työtään ja päätöksiä tavalla, jolla saadaan luotua haluttua liiketoiminta-arvoa.

Viimeinen lohko, *huippulohko (apex)*, pitää sisällään kaksi vaihetta: *älykkään vaiheen (smart)* ja *kehittymisvaiheen (progress)*. Lohkon tarkoitus on keskittyä oppimiseen ja laadukkaaseen tuotantoympäristöön pyrkien samalla minimoimaan resurssien käyttö. Prosessi pyrkii jatkuvasti parantamaan resurssien käyttöä, mutta laatu ei saa vaarantua. Älykkään vaiheen tarkoituksena on vakiinnuttaa oppiva ympäristö organisaatioon. Oppimista tulisi tapahtua niin ihmisissä, jotka ovat ohjelmistokehityksessä mukana, prosessissa, tuotteessa ja työkaluissa. Kehittymisvaiheessa käytänteet on keskitetty tavoitteeseen päästä kevyeen (lean) tuotantoympäristöön. Tällöin pyritään pitämään prosessi ketteränä ja tuottamaan mahdollisimman lyhyellä aikajänteellä ja mahdollisimman vähin resurssein, tinkimättä kuitenkaan laadusta.

4.6 Räätälöinti jakamalla se staattiseen ja dynaamiseen räätälöintiin

Aydin ym. (2005) ovat tutkineet ketterien menetelmien räätälöintiä tulkitsevalla pitkäaikaisella kenttätutkimuksella isossa eurooppalaisessa finanssialan yrityk-

sessä. Tutkimuksen lähestymistapa on induktiivinen. Organisaation käyttämä menetelmä oli DSDM, mutta tutkimuksen avulla saatiin tietoa, joka hyödyttää myös muiden ketterien menetelmien sekä menetelmien yleisesti räätälöintiä. Räätälöintiä voidaan heidän mukaansa tarkastella kahdesta näkökulmasta, suunnittelunäkökulmasta (engineering perspective) sekä sosio-organisatorisesta näkökulmasta (socio-organizational perspective). *Suunnittelunäkökulmassa* räätälöintiä tekevät menetelmäekspertit (method engineer) perustuen projektikontekstista olevan tietämyksen pohjalta, kun taas sosio-organisatorisessa näkökulmassa räätälöinti on jatkuvaa toimintaa, jolla mukautetaan menetelmää aina ohjelmistokehityksen tilanteen mukaisesti (evolving). Suunnittelunäkökulman mukaan menetelmäosat ovat koherentteja ja rakenteisia ja prosessia ohjailevat tietyt tarkoitukset, kun taas sosio-organisatorisesta näkökulmasta osat ovat erillisiä ja niitä kehitetään projektin aikana. Prosessi nähdään huonommin jäsentyneenä.

Aydin ym. (2005) jakavat havaintojensa perusteella menetelmän räätälöinnin staattiseen ja dynaamiseen. *Staattinen menetelmän räätälöinti* tapahtuu ennen projektin alkua menetelmäeksperttien, toimesta. Se kohdistuu menetelmän rakenteisiin osiin, kuten vaiheisiin, aktiviteetteihin tai tekniikoihin. Menetelmä pyritään mukauttamaan projektin tiedossa oleviin ominaispiirteisiin. *Dynaaminen menetelmän räätälöinti* tapahtuu ohjelmistokehityksen aikana, kun menetelmää käytetään. Tällöin menetelmäeksperttien lisäksi räätälöintiä suorittavat projektipäälliköt. Kun staattinen räätälöinti liikkuu käsitteellisellä tasolla, dynaaminen räätälöinti on luonteeltaan enemmän empiiristä. Menetelmää räätälöidään tällöin yksityiskohtaisemmalla tasolla muuttamalla käytössä olevia menetelmän osia, tuomalla niitä lisää tai innovoimalla uusia, riippuen siitä, mitä käsillä oleva tilanne vaatii. Menetelmän sopeuttamisen lisäksi voidaan tarvittaessa tehdä räätälöintiä myös itse projektin tilannetekijöihin, jotta tilanne tukee paremmin menetelmän osa-alueita. Tätä voidaan tehdä esimerkiksi sen takia, että halutaan noudattaa menetelmää tarkemmin.

Dynaamisessa räätälöinnissä voidaan käyttää hyödyksi taulukkomuotoista ESRL-työkalua (Extended Suitability and Risk List), jonka avulla tilannetekijöitä voidaan kartoittaa. Tekniikassa on sarakkeet tilannetekijöille (esim. loppukäyttäjien oikeutus), niiden kuvaukselle (esim. käyttäjillä oikeus tehdä päätöksiä, mutta he eivät välttämättä käytä sitä) sekä toimille, joilla tekijää pystytään estämään (esim. kerro käyttäjille, että he voivat tehdä päätöksiä) ja korjaamaan (esim. tee sopimuksia työntekijöiden saavutettavuudesta).

Aydin ym. (2005) esittävät staattiseen ja dynaamiseen menetelmän räätälöintiin myös päätelmiä ja ohjeita. Niitä voidaan hyödyntää etenkin isoissa organisaatioissa. ESRL-tekniikan tyylistä työkalua voidaan käyttää tallentamaan aiemmista projekteista historiatietoja, joita voidaan hyödyntää myöhemmin dynaamisessa menetelmän räätälöinnissä.

4.7 XP:n räätälöinti tutkimalla menetelmän ja kehittäjien erityispiirteitä

Conboy ja Fitzgerald (2010) ovat esittäneet viitekehyksen, jolla voidaan analysoida ketterien menetelmiä räätälöintiä tukevia ominaispiirteitä ja ohjelmistokehittäjien räätälöintiin liittyviä kykyjä (vrt. alaluku 2.3.). Heidän tutkimuksensa lähestymistapa on induktiivinen. He ovat käyttäneet viitekehystä XP-menetelmän analysointiin. He toteavat, että XP:stä (Beck, 1999) on esitetty, millaisiin tilanteisiin se ei sovellu. Toisaalta XP:stä on tehty lukuisia tapaustutkimuksia ympäristöissä, mihin sitä ei ole alun perin ajateltu käytettäväksi. Muiden ketterien menetelmien yhteydessä rajauksia ei ole selvästi sanottu. Esimerkiksi Schwaber ja Beedle (2002) toteavat Scrumin sopivan kaikkiin projekteihin. Tilannetekijöiden huomioiminen on niin XP:n kuin monien muiden ketterien menetelmien yhteydessä ongelma. Ne eivät tarjoa esimerkkejä siitä, miten ne voitaisiin ottaa huomioon räätälöintiprosessin aikana. Lisäksi XP:n käytänteet ovat hyvin binäärisiä: ne joko ovat mukana, tai ne jätetään käyttämättä kokonaan. (Conboy & Fitzgerald, 2010.)

Toinen puoli viitekehyksestä käsittelee ohjelmistokehittäjien räätälöintiin liittyviä kykyjä. Conboy ym (2010) käyttivät viitekehystä ketterien menetelmien räätälöintiä koskevien tapaustutkimusten tarkasteluun. He toteavat, että tutkimusten raportoinnissa on monia puutteellisuuksia koskien ohjelmistokehittäjien tilannetekijöiden tunnistamisessa, sekä siinä, onko XP valittu joukosta menetelmiä, ja tunsivatko kehittäjät muita menetelmiä tai menetelmänosia kuinka hyvin. XP:ä koskevat tapaustutkimukset eivät usein kerro myöskään sitä, kuinka kurinalaisesti ja millä tavalla räätälöintiä on suoritettu, tai miten sen käytänteitä on arvioitu ennen sen käyttämisestä päättämistä. (Conboy & Fitzgerald, 2010.)

Viitekehystä on käytetty myös XP:ä käyttäjien haastattelemiseksi sen selvittämiseksi, kuinka he menetelmää räätälöidessään kokivat XP:n ominaispiirteet ja kuinka he noudattavat viitekehyksessä mainittuja keinoja. Suurimmalle osalle kehittäjistä oli epäselvää, millaisiin tilanteisiin XP soveltuu. Monet olivat lukeneet tai kuulleet, ettei XP:tä tulisi räätälöidä vaan sen periaatteita ja käytäntöjä tulisi noudattaa tarkasti. XP:ä kuitenkin käytännössä räätälöitiin tavalla tai toisella. Usein kehittäjät olivat pyrkineet hakemaan tietoa tilannetekijöiden huomioimisesta, mutta eivät olleet löytäneet sopivaa materiaalia. Räätälöinti tapahtui usein ad hoc -tavalla ja virheistä oppimalla. (Conboy & Fitzgerald, 2010.)

Perustuen aiempien tutkimusten analysointiin ja ohjelmistokehittäjien haastatteluihin Conboy ym. (2005) päätyvät esittämään kymmenen ketterien menetelmien räätälöintiä koskevaa ohjetta ohjelmistokehittäjille (taulukko 7). Ensiksikin, kehittäjien kannattaa tehdä formaali analyysi menetelmän sopivuudesta tiettyyn projektiympäristöön. Tähän formaaliin analyysiin voidaan käyttää esimerkiksi Boehmin ja Turnerin (2003) viiden akselin kaaviota (vrt. kuvio 7). Kaikki kehittäjät tulisi pitää mukana räätälöintiprosessissa ja sen eri vaiheis-

sa. Tämä tarkoittaa, että kehittäjiä tulisi kuunnella niin tehtäessä päätöstä XP:n käyttämisestä, räätälöintiprosessin aikana kuin myös XP:n implementoinnin aikana.

TAULUKKO 7 Kymmenen ohjetta XP:n räätälöintiin

1.	Toteuta formaali analyysi menetelmän sopivuudesta projektiympäristöön
2.	Ota kehittäjät mukaan päätöksentekoon XP:n käyttämisestä, räätälöinnistä sekä implementoinnista
3.	Tunnista rajat, joihin voidaan mennä, äläkä vie XP:ä pidemmälle
4.	Kouluta kehittäjiä XP:stä
5.	Toteuta käytännön harjoituksia, kun kehittäjiä koulutetaan XP:stä
6.	Rohkaise kehittäjiä opettelemaan ja hankkimaan tietämystä muista menetelmistä
7.	Kartoita kehittäjien tietämys muista menetelmistä ja käytänteistä ja käytä sitä hyväksi räätälöinnissä
8.	Seuraa, kuinka kehittäjät pitäytyvät käytänteiden käytössä esim. palaverien avulla
9.	Viesti tiimin sisällä räätälöinnistä myös implementoinnin jälkeen
10.	Jos päätös räätälöinnistä tulee yhdeltä henkilöltä, sen vaikutukset muihin työntekijöihin tulee arvioida

Jokainen kehittäjä tulisi opettaa tuntemaan XP ja sen käytänteet ja mahdollisuudet niiden käyttöön. Tämä voidaan tehdä esimerkiksi organisaation sisällä (esim. 1 päivän kurssi) tai kannustamalla tutkimaan asiaa internet-lähteistä. Tämän lisäksi tulisi olla käytännöllistä koulutusta XP:n käytöstä, kuten työnkulun miettiminen, pelit, mentorointi tai XP:n roolien pelaaminen. XP:n lisäksi kehittäjiä tulisi kannustaa oppimaan ja hankkimaan kokemusta muistakin kuin XP-menetelmästä. Yhdessä tapauksessa kehittäjät olivat tehneet tämän niin, että tiimin kaikkein kehittäjien tuli valita yksi ketterä menetelmä ja oppia siitä lisää sekä arvioida sitä sen perusteella. Opettelun lisäksi tulisi kartoittaa kehittäjien tietämystä ja kokemusta muista menetelmistä tiimin sisällä ja hyödyntää tätä XP:tä räätälöitäessä ja mahdollisesti lisättäessä siihen uusia osia.

XP:n käytänteissä pitäytymistä tulisi seurata koko projektin ajan, jotta käytänteitä ei jää pois turhaan esimerkiksi laiskuuden tai huolimattomuuden takia. Tätä voidaan tarkastella esimerkiksi palavereissa, jonka aikana samalla pystyy tarkastelemaan käytänteestä saatavia hyötyjä ja haittoja, jolloin tarvittaessa sen voi jättää harkitusti pois. Tiimin sisällä tulisi myös laajemmin keskustella itse implementoinnin jälkeen tapahtuvasta räätälöintityöstä. Keskustelua voidaan käydä tiimin ollessa koolla esimerkiksi päivittäisissä palaverissa tai retrospektiivisissä palavereissa. Lisäksi mikäli jonkin räätälöintipäätöksen tekee yksittäi-

nen kehittäjä, sen vaikutusta muihin tiimin jäseniin tulisi arvioida ja siitä keskustella.

Tutkimuksen tarkoituksena oli selvittää, kuinka sopiva XP oli räätälöintiin ja tehdä ehdotuksia siitä, miten sitä voisi parantaa. Toiseksi tarkoituksena oli tutkia, kuinka kehittäjät räätälöivät XP:tä, sekä tuottaa parhaita käytäntöjä, ketterän menetelmän räätälöintiin. Vaikka monet tuloksista koskivat ainoastaan XP:ä, niiden voi katsoa sopivan myös yleisesti ketteriin menetelmiin tai kehittämismenetelmiin.

Ohjeet toimivat enemmänkin suosituksina, mutta ne eivät neuvo tallentamaan kokemuksia räätälöinnistä seuraavia projekteja varten. Ohjeistoa voisi täydentää tätä koskevalla ohjeella.

4.8 Ohjeita räätälöintiin tutkimalla tapaustutkimuksia AST-teorian avulla

Cao ym. (2009) ovat tutkineet ketterien menetelmien käyttöönottoa ja räätälöintiä eri konteksteissa käyttämällä hyväkseen adaptiivista strukturaatioteoriaa (engl. adaptive structuration theory, AST, Poole & DeSanctis, 1990; DeSanctis & Poole, 1994). Heidän lähestymistapansa on induktiivinen. AST tarkastelee organisaatiossa tapahtuvaa muutosta, joka voi johtua erilaisista rakenteista, kuten teknologiasta, toimista, organisaatioympäristöstä tai sosiaalisista teoista. Ketterät menetelmät on tässä yhteydessä nähty tuovan rakenteen ohjelmistokehitykseen, ja käyttöönotettu rakenne näkyy sosiaalisten tekojen muodossa. Rakenteistaminen on sääntöjen, resurssien ja muiden rakenteiden tuomista käyttöön. Omaksuminen (appropriation) taas tarkoitti näitä rakenteita itse käytössä.

Tarkemmin teoriaa testattiin neljän tapaustutkimuksen yhteydessä, joissa organisaatio oli räätälöinyt XP:n käyttöönsä. XP toimi tällöin rakenteena, mutta rakenteita muokattiin usein muun muassa turvallisuuskriittisten tai kompleksisten projektin kohdalla. Myös organisaatiokulttuuri ja johdon asenne vaikuttivat siihen, miten XP:n käytänteitä otettiin käyttöön. Lisäksi kehittäjätiimin asenne, kokemus ja tietämys XP:stä vaikuttivat osaltaan käytäntöjen käyttöön. Esimerkiksi tiimin uskon XP:hen ollessa vähäistä kehittäjät eivät edes halunneet kokeilla erilaisia käytäntöjä.

XP:n käytäntöjen omaksuminen vaihteli paljon projektikohtaisesti. Tilannetekijöistä johtuen eri käytänteisiin tehtiin muutoksia, joiden takia myös alkuperäiset käytänteet erosivat omaksutuista käytänteistä. Esimerkiksi sovellusalueen takia oli tarve arkkitehtuurille, joka ei enää noudattanut XP:n periaatetta yksinkertaisesta suunnittelusta. Valittu refaktorointitapa erosi merkittävästi XP:n refaktorointi-käytänteestä. Kompleksisten ja laajojen sovellusten kanssa läsnä olevien asiakkaiden (on-site customer) käyttö on vaikeampaa, jonka johdosta esimerkiksi testien kirjoittamisen jälkeen kehittäjät varmistivat niiden paikkansapitävyyden liiketoiminnasta vastaavilta henkilöiltä. Vaatimusten jäljitettävyys pidettiin pienenä.

Myös kehittäjistä johtuvista syistä joitakin käytänteitä räätälöitiin. Pariohjelmointia suoritettiin osittain, yhteisomistajuus rohkaisi kehittäjiä tutustumaan kohdealueeseen ja ohjelmistojen laadusta tehtiin sopimuksia, jolloin ohjelmiston laadusta puhuttaessa oli selvää, mitä tarkoitetaan. Organisaatioon liittyvänä seikkana ylimmän johdon toiveet olivat osittain ristiriidassa ketterien arvojen kanssa, jonka johdosta formaalisuuden ja ketteryyden kanssa jouduttiin hakemaan tasapainoa.

Cao ym. (2009) esittelevät tapaustutkimuksien tulosten perusteella kolme ohjetta ketterän menetelmän käyttöönotolle ja räätälöinnille. Ensimmäinen ohje on ylimmälle johdolle, jonka tulee ymmärtää roolinsa ja tukensa tärkeys ketteriä menetelmiä räätälöitäessä. Tuen puuttuessa menetelmien käyttö huononee ja se vaikuttaa pahimmillaan projektin lopputulokseen. Ketterien menetelmien käytänteitä voidaan joutua räätälöimään, jotta ylemmän johdon, organisaatiokulttuurin ja kehitystiimin kulttuurin välille saadaan tasapaino. Toisena ohjeena projektipäälliköiden tulee huomata, ettei menetelmiä voida ottaa käyttöön ilman niiden vaikutuksen arviointia kehitystiimiin ja prosessin tuotoksiin. Heidän tulee myös ottaa huomioon kehitystiimien tyylit ja tilannetekijät sekä tunnistaa tekijät, jotka voivat aiheuttaa kitkaa ylemmän johdon kanssa. Kolmas ohje on tarkoitettu kehittäjille, joiden tulee ymmärtää käytänteiden vaatima itsenäisyys, jota heiltä odotetaan.

4.9 XP:n räätälöinti RDP-tekniikalla

Mirakhorli ym. (2008) esittelevät XP:n räätälöintitekniikan. joka perustuu käytäntöjen sijasta sääntöihin. Mirakhorlin ym. (2008) lähestymistapa on deduktiivinen. XP:n säännöt jaetaan kahteen kategoriaan, sitoutumissääntöihin (Rules of Engagement), jotka liittyvät ketteryyteen, sekä pelin sääntöihin (Rules of Play), jotka liittyvät seikkoihin, mitkä tekevät XP:stä erityisen (unique).

Sitoutumisen sääntöihin kuuluu kuusi sääntöä. Ensimmäisen säännön mukaan liiketoiminnasta vastaavien ihmisten tulee työskennellä yhdessä kehittäjien kanssa päivittäin koko projektin ajan. Toisena sääntönä on, että tärkeimpänä asiana on asiakkaan tyytyväisyys. Tämä tarkoittaa, että asiakkaan tulee asettaa tavoitteet ja tarvittaessa muuttaa tavoitteita ja prioriteetteja kehittäjien antamien arvioiden perusteella. Kolmas sitoutumisen sääntö on toimittaa toimivaa ohjelmistoa jatkuvasti, mieluummin mahdollisimman pienin väliajoin. Neljännen säännön mukaan toimiva ohjelmisto on edistymisen mittari. Viidenneksi tulee globaali tietoisuus, mikä tarkoittaa, että missä tahansa projektin vaiheessa tulee voida arvioida tiimin edistymistä asiakkaitten tavoitteiden saavuttamisessa ja tiimin tulee reflektoida, kuinka se voi tehostaa toimintaansa. Viimeinen sitoutumisen sääntö liittyy tiimin toimimiseen sosiaalisena verkostona, jossa korostuu muiden muassa henkilöiden välinen kommunikaatio ja vastuullisuus.

Pelin sääntöjä on viisi, ja ne perustuvat XP:n periaatteisiin. Ensimmäinen sääntö on jatkuva testaaminen, jolla ohjelmistoa validoidaan jatkuvasti. Toinen sääntö on selkeys ja ohjelmakoodin laatu, jonka tulee olla selkeästi ilmaistua ja

yksikkötestattua. Kolmas sääntö koskee yhteistä sanastoa, jolloin yhteinen yksinkertainen tarina kertoo, miten järjestelmän tulisi toimia. Neljäs sääntö on se, että kaikilla on oikeus ja vähintään kahdella kehittäjällä on ymmärrys tehdä mikä tahansa tehtävä. Viimeinen sääntö koskee sitä, että kehityksen tulisi lähteä testivetoisesti pareissa.

Sääntöjen tukemiseksi Mirakhorli ym. (2008) esittelevät RDP-tekniikan, jonka avulla muodostetaan RDP-kortteja, jotka auttavat tunnistamaan käytänteitä, mitkä tukevat aiemmin esitettyjä sääntöjä. Kirjaimet RDP tulevat sanoista ”Rule” (sääntö), ”Description” (kuvaus) ja ”Practice” (käytäntö). Sääntöjä tukemaan voi ottaa niin XP:n omia käytänteitä kuin myös muita käytänteitä. Jokaisesta säännöstä tehdään oma kortti CRC-korttien tapaan, johon merkitään säännön otsikko, säännön kuvaus sekä ne käytänteet, jotka on valittu tukemaan kyseistä sääntöä.

Tekniikka pitää sisällään neljä erilaista roolia, joihin tulee valita sopivat henkilöt. *Johtaja* järjestää tapaamiset tiimin kesken ja johtaa keskustelua. *Asiakas* kertoo yleiskuvan järjestelmästä omasta perspektiivistään ja vastaa esitettyihin kysymyksiin. *Seuraaja* (*tracker*) auttaa hiomaan ja tarkastelemaan valittuja käytänteitä, jotta varmistutaan, että ne täyttävät menetelmän säännöt. *Tarkkailija* (*observer*) taas havainnoi RDP-tekniikan käyttämistä ja miettii sitä, kuinka sitä voitaisiin parantaa, sekä ilmoittaa prosessin jälkeen, kuinka hyvin se toimi.

RDP-tekniikkaa käytettiin tapaustutkimuksessa, jossa räätälöitiin XP tukemaan sääntöjä ja projektin olosuhteita. Tällöin käytänteiksi valittiin osa-aikaisesti paikalla oleva asiakas, toimialueen asiantuntijoiden käyttäminen, suunnittelupeli, pienet julkaisut, iso näkyvillä olevilla taulu, yhteisomistajuus, osittainen pari-ohjelmointi, testaajarooli, refaktorointi, koodausstandardit, arkkitehtuuriin keskittyminen ja arkkitehtuurin jatkuva refaktorointi.

RDP-tekniikkaa käyttäessä räätälöinnin onnistuminen voi kuitenkin vaatia hyvää tietämystä XP:stä ja siitä, miten tietyt käytänteet voivat tukea erilaisia sääntöjä ja projektissa vallitsevia ominaispiirteitä. Myös arvoa tuottavia käytänteitä voi mahdollisesti jäädä käyttämättä.

4.10 Tiekartta XP:n asteittaiseksi käyttöönotoksi

Lui ja Chan (2005) esittävät tiekartan, jonka mukaan XP:n käytänteitä voidaan ottaa asteittain käyttöön. Heidän lähestymistapansa on kypsyysmalliajatteluun perustuva. Tästä on hyötyä etenkin kokemattomille kehitystiimeille tai opiskelijoille, joiden tapauksessa kaikki käytänteet kerralla ottava lähestymistapa ei toimisi.

Lähtökohtana tiekartalle käytetään Beckin (1999) graafista esitystä XP:n käytänteiden välisistä vaikutussuhteista. Graafin perusteella Lui ja Chan (2005) ovat tehneet visuaalista tiedon louhintaa, jossa aluksi kaikki käytänteet on ryhmitelty matriisille, johon on merkitty miten käytänteet tukevat toisiaan. Tämän perusteella on saatu erilaisia osajoukkoja käytänteistä, jotka selvästi ovat kytköksissä toisiinsa. (Lui & Chan, 2005.) Yhteenveto käytänteiden kytköksistä toi-

siinsa visuaalisesti matriisilla esitettynä Luin ja Chanin (2005) esityksen mukaisesti on kuviossa 13.

	PO	R	T	YS	JI	YO	M	40	PJ	PA	SP	KS
Koodausstandardit	·				·	·						
Suunnittelupeli			·						·	·		
Paikalla oleva asiakas			·				·				·	
Pienet julkaisut			·	·	·						·	
40 tunnin työviikko	·	·		·								
Metafora	·	·		·						·		
Yhteisomistajuus	·	·	·		·							·
Jatkuva integrointi	·	·	·			·			·			·
Yksink. suunnittelu	·	·	·				·	·	·			
Testaus	·	·		·	·	·			·	·	·	
Refaktorointi	·		·	·	·	·	·	·				
Pariohjelmointi		·	·	·	·	·	·	·				·

KUVIO 13 XP:n käytänteiden kytkökset toisiin käytäntöihin (Lui & Chan, 2005, 478)

Matriisista on pääteltävissä, että ensimmäisessä vaiheessa kannattaa valita käytänteiksi testaus ja testilähtöisyys, jonka lisäksi otetaan käyttöön siihen läheisesti kytköksissä olevat koodin refaktorointi ja yksinkertainen suunnittelu. Tämän lisäksi ensimmäisessä vaiheessa käytettäväksi on suositeltu valittavaksi matriisin reunalta koodausstandardit sen vuoksi, että se voi auttaa kokemattomia kehittäjiä pitämään itsekuria yllä. Toisessa vaiheessa otetaan XP:stä käyttöön jatkuva integrointi. Kolmas vaihe pitää sisällään pariohjelmoinnin ja yhteisen omistajuuden käyttöönoton, jotka ovat selvästi toisiinsa liittyviä käytänteitä. Viimeisessä eli neljännessä vaiheessa otetaan käyttöön loput XP:n käytänteet, eli metafora, 40 tunnin työviikko, pienet julkaisut, paikalla olevan asiakkaan sekä suunnittelupelin.

4.11 Vertailu ja yhteenveto

Tässä luvussa verrataan edellä esitettyjä ehdotuksia ja ohjeistoja ketterien menetelmien räätälöintiin taulukossa 8 esitetyn yhteenvedon pohjalta. Taulukossa on esitetty lähde, räätälöintistrategia ja -lähtökohta, räätälöinnin kohde sekä räätälöintiprosessi. Räätälöinnin lähtökohtana voi toimia jokin olemassa olevista menetelmistä, eli pohjamenetelmä, olemassa olevat menetelmät tai ketterät käytänteet. Räätälöinnin kohteella tarkoitetaan niitä osia ketteristä menetelmistä, joita räätälöinnillä muokataan. Räätälöintiprosessi-sarake osoittaa, miten ko. lähteessä on jäsennetty räätälöinti vaiheisiin tai aktiviteetteihin. Taulukossa esitettyjen piirteiden lisäksi vertailussa kiinnitetään huomiota siihen, ketkä räätälöintiä tekevät sekä mahdollisiin käytännön ohjeisiin, joita voidaan käytännön työssä käyttää hyödyksi.

TAULUKKO 8 Yhteenveto ehdotuksia ja ohjeita tarjoavista tutkimuksista ketterän menetelmän räätälöintiin

Lähde	**Räätälöintistrategia ja -lähtökohta**	**Räätälöinnin kohde**	**Räätälöintiprosessi**
Aydin ym. (2005)	Räätälöinti jaettu staattiseen ja dynaamiseen räätälöintiin. Pohjamenetelmänä DSDM, mutta soveltuu muihinkin ketteriin menetelmiin	Prosessiorientoituneet piirteet ja sosio-organisatoriset piirteet	Räätälöinti ennen projektia staattisesti tiedossa oleviin piirteisiin ja projektin aikana dynaamisesti ilmeneviin tekijöihin ja riskeihin.
Ayed ym. (2012)	Ketterän menetelmän räätälöinnin pohjana on metamalli ja laadun mittaamisen malli	Ketterän menetelmän mukainen tuote, prosessi, resurssi	Ei käsitelty
Cao ym. (2009)	AST-teorian hyväksikäyttäminen räätälöimisessä ja räätälöinti erilaisten tilannetekijöiden mukaisesti	Organisaatioiden eri rakenteet, tekniset ja sosiaaliset	Ei suoranaista prosessia
Conboy & Fitzgerald (2010)	Yleisten ohjeiden soveltaminen XP:n räätälöimisessä, mutta myös muiden ketterien menetelmien räätälöimisessä	Räätälöintiohjeiden huomioiminen prosessissa mm. kehittäjien, prosessin ja viestinnän kannalta	Räätälöintiprosessin aikana olevien ongelmakohtien huomioiminen ja ratkaiseminen ohjeita soveltamalla
Karlsson & Åkerfalk (2009)	Jonkin pohjamenetelmän käyttäminen ja sovittaminen	Pohjamenetelmä ja osia muista menetelmistä	Erilaisten menetelmäkomponenttien yhdistäminen kokonaisuudeksi systemaattisesti ja uudelleenkäytettävien osien luonti
Lui & Chan (2005)	Pohjamenetelmänä XP	XP-käytänteiden joukko	XP:n käytänteiden ottaminen käyttöön vaiheittain
Mikulenas & Kapocius (2011)	Menetelmän integrointi joukosta käytänteitä	Ketterät käytänteet eri menetelmistä otettuina	Käytänteiden ja menetelmäosien valitseminen niiden kustannuksia ja arvoja vertailemalla
Mirakrakhorli ym. (2008)	XP-menetelmän sovittaminen	XP:n säännöt ja käytänteet	RDP-korttien muodostaminen ja valitseminen aivoriihessä projektin erityispiirteiden perusteella
Qumer & Henderson-Sellers (2010)	Ketteryyden vaiheittainen omaksuminen ja sen mittaaminen eri tasoilla	Organisaation käyttöönottoon keskittyminen mm. käytänteiden puolesta	Ei suoranaista prosessia

Aydinin ym. (2005) esittämää jakoa menetelmän staattiseen ja dynaamiseen räätälöintiin ei ole muissa tutkimuksissa ollut esillä, vaikka räätälöintiä voidaan usein tilannetekijöiden pakottamana joutua tekemään itse projektin aikana. Esitys ottaa kuitenkin kantaa ainoastaan laajoihin organisaatioihin, joilla on enemmän kokemusta räätälöinnistä esimerkiksi menetelmäeksperttien toimesta. Se ei ota kantaa esityksen soveltuvuuteen pienempien ja kypsyysasteeltaan alempien organisaatioiden ketterän menetelmän räätälöintiin. Aydin ym. (2005) käsittelee räätälöintiä projektitasolla, ja on tarkastelutasoltaan yleinen.

Ayedin ym. (2012) jäsentävät räätälöinnin kohteen yleisellä tasolla metamallin avulla. Metamalli tarjoaa keinon visualisoida menetelmäkomponentteja, jotka ovat räätälöidyssä menetelmässä käytössä. Se myös tarjoaa tietoa laadun mittaamisen menetelmistä, jotka voivat auttaa räätälöidyn menetelmän parantamisessa. Toisin kuin muut tässä tutkielmassa ketterän menetelmän räätälöintiä käsittelevät tutkimukset, se ei tarjoa konkreettisia ohjeita sille, kuinka räätälöintiä tulisi suorittaa. Esitys on näkökulmaltaan selvästi yleisin.

Caon ym. (2009) lähestymistapa ketterien menetelmien räätälöinnin tutkimiseen AST:n avulla poikkeaa lähestymistavaltaan muista, mutta sitä on mahdollista käyttää muissakin ketterien menetelmien räätälöintiä koskevissa tutkimuksissa hyödyksi. Tutkimuksen tuloksena on huomio ylimmän johdon roolin tärkeydestä räätälöinnin yhteydessä. Jos ketterien menetelmien käyttöönotolle ei ole ylemmän johdon tukea, on niitä vaikea saada jalkautettua. Cao ym. (2009) antavat ohjeita myös projektipäälliköille ja kehittäjille. Räätälöintiä käsitellään organisaatio- ja projektitasolla, ja tarkastelutasoltaan esitys on yleinen.

Conboyn ja Fitzgeraldin (2010) tutkimus antaa konkreettisia ohjeita räätälöintiin. He ovat osoittaneet kohtia, jotka on nähty ongelmalliseksi. Sen jälkeen he tarjoavat niihin mahdollisia ratkaisukeinoja. Ohjeet eivät poissulje muiden keinojen hyväksikäyttämistä ketteriä menetelmiä räätälöidessä. Räätälöintiprosessiin kantaa ottavat lähestymistavat voivat sopia käytettäväksi Conboyn ja Fitzgeraldin (2010) ohjeiden kanssa. Räätälöintiä he käsittelevät projektikohtaisella tasolla, ja esitys antaa yksityiskohtaisia ohjeita.

Karlsonin ja Ågerfalkin (2009b) esittelemä MMC on tilannekohtaista menetelmäkehitystä soveltava menetelmä, jossa projektille tulee oma, yksilöllinen menetelmä. MMC on jo varsin kypsä ja pitkälle kehittynyt. Tämä tulee ilmi muun muassa tuesta eri työkaluille ja siinä, että menetelmän käyttämisen aikana sitä voidaan tarvittaessa muuttaa. Saadut räätälöintikokemukset tulisi ottaa talteen ja hyötykäyttöön tulevaisuudessa. Työkalu pitää sisällään mahdollisuuden menetelmätietokantaan, joka auttaa kokemuksien haltuun ottamisessa. Kokemusten haltuunotto osana räätälöintiprosessia on mainittu jo aiemmin alaluvussa 2.3 Patelin ym. (2004) esittelemässä viitekehyksessä. Erilaisten osien uudelleenkäyttämisen voi katsoa olevan ketterien arvojen mukaista. Räätälöinnissä on mukana myös erillinen menetelmäekspertti. Räätälöintiä Karlsson ja Ågerfalk (2009b) käsittelevät projektitasolla, mutta menetelmätietokanta on organisaatiotasoinen. Yksityiskohdiltaan esitys on yleinen.

Luin ja Chanin (2005) tutkimus XP:n käytänteiden kytköksistä toisiinsa antaa konkreettisen lähtökohdan, jonka avulla sen räätälöintiä voidaan suorittaa. Se ottaa kantaa kuitenkin ainoastaan XP:n käytäntöihin, eikä esimerkiksi XP:n prosessimalliin tai rooleihin. Käytänteitä otetaan tiekartan mukaisesti asteittain kehitystiimin käyttöön. Vaiheiden välinen jako ei ole kuitenkaan selvää, eivätkä Lui ja Chan (2005) kerro sitä, milloin kehitystiimi on valmis etenemään seuraavaan vaiheeseen. Käytänteiden ym. menetelmien osien välisiä suhteita ja riippuvuuksia on kuitenkin tärkeää pohtia räätälöidessä, ja XP:n kohdalla Luin ja Chanin (2005) kuvio antaa visuaalisen esityksen siitä, miten eri käytänteet ovat kytköksissä toisiinsa. Heidän esityksensä käsittelee räätälöintiä organisaatio- ja projektitasolla, ja tarkastelutasoltaan se on yksityiskohtainen.

Mikulenasin ja Kapociuksen (2011) esittelemä ketterien käytäntöjen tutkiminen ja valitseminen kustannuksia ja arvoja vertailemalla muistuttaa aiemmin tutkielmassa mainittua tilannelähtöistä menetelmäkehitystä. Tässä lähestymistavassa käytänteet valitaan tutkimalla käytänteitä, joista valitsemalla muodostetaan tilannekohtainen menetelmä. Vaikkeivät Mikulenas ja Kapocius (2011) sitä mainitse, olisi tällä tavalla tunnistetuista käytänteistä mahdollisuus muodostaa organisaation oma menetelmätietokanta. Toisin kuin tilannelähtöisessä menetelmäkehityksessä usein käy, tässä koko projektitiimi osallistuu menetelmän räätälöintiin sen sijaan, että sitä tekisi esimerkiksi yksi menetelmäekspertti, kuten esimerkiksi Kumar ja Welke (1992) Karlssonin ja Ågerfalkin (2009b) mukaan suosittelevat. Tämä on linjassa myös Agile-manifestin (Agile-allianssi, 2001a; Agile-allianssi, 2001b) periaatteiden kanssa. Räätälöintiä käsitellään projektitasolla. Mikulenaan ja Kapociuksen (2011) esitys käsittelee räätälöintiä yksityiskohtaisella tasolla, mutta vain käytänteiden valinnan osalta.

Mirakhorlin ym. (2008) esittelemä RDP-tekniikka keskittyy räätälöimiseen sääntöjen perusteella, eroten siten monista muista lähestymistavoista ketterien menetelmien räätälöimiseksi. Räätälöinti ottaa huomioon myös ketterän manifestin erityispiirteet ja tarjoaa konkreettisen tiekartan sille, miten räätälöintiä voi suorittaa. Räätälöintiä käsitellään projektitasolla, ja esitys käy läpi räätälöintiä yksityiskohtaisella tasolla.

Qumerin ja Henderson-Sellersin (2008) malli on hyvin kokonaisvaltainen ja se tarjoaa tasovaiheisiin perustuvan tiekarttamaisen ketterän menetelmän parantamiseen. Ketterän ohjelmistokehityksen viitekehyksen avulla voi laajemmin tarkastella ketteriä menetelmiä, ja AAIM-mallia voidaan käyttää hyväksi itse räätälöinnin aikana. AAIM-mallin avulla voidaan analysoida organisaation nykytilaa ja pohtia tulevaisuuden kehittämismahdollisuuksia. Tasovaiheisesti toimivissa lohkoissa ja vaiheissa voi edetä sen mukaisesti, kuinka hyvin menetelmä ja ketteryys on pystytty sopeuttamaan organisaation käyttöön. Malli on organisaatiotasoinen, ja esitys tarkastelutasoltaan yleinen.

Ketterän menetelmän räätälöintiä koskevia ehdotuksia ja ohjeita on esitetty kirjallisuudessa erilaisia. Aiheen käsittelyssä ne keskittyvät eri asioihin. Osa tutkimuksista esittelee hyvin yleisen mallin räätälöinnille, kun taas toisissa tutkimuksissa on esitelty tarkka prosessi tai yksityiskohtaisia ohjeita sille, kuinka räätälöintiä tulisi suorittaa. Räätälöintiä on käsitelty usein projektitasolla, mutta

joskus myös organisaatiotasolla. Osassa tutkimuksista räätälöinti suoritetaan räätälöimällä yhtä ketterää menetelmää tai käyttämällä sitä pohjamenetelmänä, mutta osassa menetelmään ei ole otettu kantaa, tai käytänteitä haetaan tasa-arvoisesti useista ketteristä menetelmistä. Monet esitykset neuvovat suorittamaan räätälöintiä tiekarttamaisesti, jolloin räätälöinti ja ketterän menetelmän käyttöönotto tapahtuu vaiheittain. Näiden lisäksi muun muassa tutkimusmenetelmät (esim. induktiivinen vs. deduktiivinen), kohdeorganisaatioiden koot ja konkreettiset ohjeet kehittäjille vaihtelevat tutkimuksittain.

5 KETTERÄN MENETELMÄN RÄÄTÄLÖINTIÄ KOSKEVIA TAPAUSTUTKIMUKSIA

Tässä luvussa tarkastellaan ketterän menetelmien räätälöintiä koskevia tapaustutkimuksia. Ensiksi kerrotaan, miten tapaustutkimuksia on etsitty, mitkä ovat olleet valintakriteerit ja mitkä tapaustutkimukset ovat tulleet valituiksi. Toiseksi esitetään viitekehys, jota käytetään ketterien menetelmien räätälöintiä koskevien tapaustutkimusten analysoimiseksi ja vertaamiseksi. Tämän jälkeen kuvataan valittuja tapaustutkimuksia ja lopuksi niitä tarkastellaan yhdessä analysoiden ja vertaillen viitekehyksen avulla.

5.1 Tapaustutkimusten valinta

Alaluvussa 4.1. kerrottiin, miten ketterien menetelmien räätälöintiä koskevaa kirjallisuutta on tässä tutkimuksessa etsitty. Etsinnässä käytettiin tutkimustietokantoja ja aihetta kuvaavia asiasanoja. Osa tutkimuksista käsitteli ketterien menetelmien räätälöintiä yleisellä tasolla perustuen käsitteellis-teoreettiseen tarkasteluun. Näitä tutkimuksia kuvattiin ja analysointiin edellisessä luvussa. Tässä luvussa keskitytään löydettyihin tapaustutkimuksiin. Näitä tutkimuksia oli 39 kpl. Ne on esitetty taulukossa 9. Ensimmäisessä sarakkeessa on mainittu lähde. Toisessa sarakkeessa on tutkimuksen erityispiirteet, joiden mukaan räätälöintiä on suoritettu. Kolmannessa sarakkeessa on mainittu menetelmät, joita on käytetty räätälöinnin pohjana. Esitysten yksityiskohtaisuus vaihtelee. Monissa tutkimuksissa on esimerkiksi kerrottu, miltä osilta menetelmää räätälöintiin, muttei tähän johtaneita tilannetekijöitä ja syitä. Erityispiirteinä on mainittu muun muassa kulttuurillisia (esim. tietyn maan ympäristö), sosio-organisatorisia (esim. iso organisaatio) sekä teknisiä (esim. sovellusalueen vaativuus) seikkoja.

Menetelmien räätälöintiä käsittelevistä tapaustutkimuksista on valittu seuraavassa alaluvussa tarkasteltaviksi seuraavin kriteerein, jotka on esitetty taulukon 9 esittelyn jälkeen.

TAULUKKO 9 Ketterän menetelmän räätälöintiä koskevia tapaustutkimuksia

Lähde	**Konteksti ja erityispiirteet**	**Pohjamenetelmä**
Bass (2012)	Yritysjärjestelmien kehittämisprojektit	Scrum & XP
Bass (2013)	Laajat ulkoistetut tai hajautetut projektit, etenkin projektiomistajan roolin räätälöiminen	Scrum
Bowers, May, Melander, Baarman & Ayoob (2002)	Turvallisuuskriittinen ja laaja järjestelmä	XP
Dahlmann, Gregorio & Modigliani (2013)	Yhdysvaltojen puolustusministeriön alaiset projektit (ISO 15288 -standardin täyttävät)	Scrum
Díaz, Perez, Yagüe & Garbajosa (2012)	Ohjelmistotuoteperheiden suunnittelu	Scrum
Diebold, Lampasona & Taibi (2013)	Hajautetut kehitystiimit, joilla vähän päällekkäisiä työtunteja	Scrum
Drobka, Noftz & Raghu (2004)	Tehtäväkriittisten järjestelmien projektit	XP
Fitzgerald, Hartnett & Conboy (2006)	Suorittimien ohjelmistokehitykseen keskittynyt organisaatio	Scrum & XP
Fitzgerald, Stol, O'Sullivan & O'Brien (2013)	Säännellyt ympäristöt (turvallisuus- ja tietoturvallisuuskriittiset)	Scrum
Fruhling & De Vreede (2006)	Hätätilanteessa käytetty järjestelmä	XP
Greaves (2011)	Asiakaspalvelun ongelmajonon hallitseminen	Kanban
Harmsen, van den Brand, Hillegersberg & Aydin (2007)	Ulkoistettu ja hajautettu järjestelmäkehitys suuressa organisaatiossa	DSDM & RUP
Hildenbrand, Geisser, Kude, Bruch & Acker (2008)	Hajautettu yritysjärjestelmien kehittäminen	XP
Hong, Yoo & Sungdeok (2010)	Ulkoistetut elektronisen liiketoiminnan projektit	Scrum
Hossain, Bannerman & Jeffery (2011)	Globaalisti hajautettu ohjelmistokehitys	Scrum & XP
Ingason, Gestsson & Jonasson (2013)	Islantilainen tietoliikenneyhtiö	Scrum & Kanban
Ikonen, Pirinen, Fagerholm, Kettunen & Abrahamsson (2011)	Yliopiston yhteydessä toimiva kokeellinen opiskelijoiden ohjelmistokehityslaboratorio	Kanban
Jayawardena & Ekanayake (2010)	Ketterän projektinhallinnan keinot Sri Lankan kontekstissa	Scrum, XP & FDD
Lagerberg, Skude, Emanuelsson, Sandahl & Ståhl (2013)	Laajat ohjelmistonkehitysprojektit	Scrum
Layman, Williams, Damian & Bures (2006)	Viestintäkäytäntöjen räätälöinti globaalissa ohjelmistokehityksessä	XP

(jatkuu)

TAULUKKO 9 (jatkuu)

Lähde	**Konteksti ja erityispiirteet**	**Pohjamenetelmä**
Layman, Williams & Cunningham (2004)	Ketterille menetelmille sopiva pieni tiimi, joka kehittää graafisia käyttöliittymiä	XP
Li, Moe & Dybå (2010)	Siirtyminen suunnitelmavetoisista prosessista ketterään menetelmään	Scrum
Little & Karaj (2013)	Julkinen sektori (Kanadan työministeriön alla toimiva yksikkö)	Kanban
Maassen & Sonnevelt (2010)	Vakuutusyhtiön järjestelmät	Kanban
Murru, Deias & Mughuddu (2003)	Eurooppalainen web-kehitystä tekevä yritys	XP
Mishra & Mishra (2011)	Monimutkaiset ohjelmistoprojektit	XP, DSDM & FDD
Nielsen & McMunn (2004)	Iso finanssipalveluja tuottava organisaatio	XP
Nikitina, Kajko-Mattson & Stråle (2012)	Scrumin käytöstä siirtyminen Scrumin ja Kanbanin yhteyskäyttöön (Scrumban)	Scrum & Kanban
Paasivaara, Durasiewicz & Lassenius (2008)	Hajautettu kehitys yhdistettynä laajan ohjelmiston kehitykseen	Scrum
Pikkarainen & Passoja (2005)	Mobiililaitteisiin videoteknologiaa kehittävä organisaatio (muuttuva ympäristö ja nopea markkinoille tuloaika)	Scrum & XP
Polk (2011)	Pelikoneiden ja kasinoiden ohjelmistoja valmistava, eiketterä organisaatio	Scrum & Kanban
Poole & Huisman (2001)	Ylläpitoympäristö	XP
Rodriguez, Partanen, Kuvaja & Oivo (2014)	Suomalainen langattomia sulautettuja järjestelmiä kehittävä organisaatio	Scrum & Kanban
Rottier & Rodrigues (2008)	Lääketieteellisiä laitteita kehittävä organisaatio	Scrum
Salo & Abrahamsson (2008)	Eurooppalaiset sulautettuja järjestelmiä kehittävät organisaatiot	Scrum & XP
Scott, Johnson & McCullough (2008)	Julkinen sektori (Calgaryn kaupunki)	Scrum & XP
Terlecka (2012)	Järjestelmäylläpitäjistä koostuva tiimi	Scrum & Kanban
Wang, Conboy & Pikkarainen (2012)	Useampia erilaisiin tilannetekijöihin perustuvia tapaustutkimuksia	Scrum & XP
Özcan, Kocak & Brune (2013)	Avoimen lähdekoodin ohjelmisto sairaalaympäristössä	Scrum & Crystal

Ensimmäisenä kriteerinä tapaustutkimuksen valinnalle edellytetään, että tutkimuksessa on raportoitu riittävän seikkaperäisesti ketterien menetelmien räätälöinnistä. Toiseksi tarkoituksena on saada joukkoon erilaisia tapauksia. Kolmanneksi edellytetään, että räätälöinnin kohteena on ollut joko Scrum, XP, Kanban tai näiden yhdistelmä siksi, että näitä menetelmiä on tässä tutkielmassa aiemmin esitelty. Valittavien joukon rajaamiseen vaikutti tarve pitää tutkielman laajuus kohtuullisena.

Näiden kriteerien mukaisesti on valittu kuusi tapaustutkimusta: Fitzgerald ym. (2006), Díaz ym. (2011), Hong ym. (2010), Maassen ja Sonnevelt (2010), Scott ym. (2008) ja Wang ym. (2012). Fitzgeraldin ym. (2006) käsittelevät Scrumin ja XP:n räätälöintiä suuressa suorittimien ohjelmistokehitykseen keskittyvässä organisaatiossa. Díaz ym. (2011) käsittelevät Scrumin räätälöintiä ohjelmistotuoteperheiden tapauksessa. Hong ym. (2010) käsittelevät Scrumin räätälöintiä ulkoistettujen elektronisen liiketoiminnan projektien yhteydessä. Maassen ja Sonnevelt (2010) käsittelevät Kanbanin räätälöintiä vakuutusyhtiön tapauksessa. Scott ym. (2008) käsittelevät XP:n räätälöintiä julkisella sektorilla. Wang ym. (2012) taas esittelevät useita erilaisia tilannetekijöitä sisältäviä tapaustutkimuksia. Tässä tutkimuksessa käsitellään näistä tarkemmin keskisuurta tietoturvallisuuden tuotteisiin keskittyvää organisaatiota, joka räätälöi Scrumia ja XP:ä.

5.2 Ketterän menetelmän räätälöinnin viitekehys

Patelin ym. (2004) esittämän ketterän menetelmän räätälöinnin viitekehyksen mukaisesti (vrt. kuvio 3) voidaan erottaa viisi keskeistä seikkaa, jotka tulee ottaa huomioon ketterän menetelmän räätälöintiä tarkasteltaessa: Mikä tai mitkä menetelmät tai niiden osat ovat räätälöinnin kohteena? Millainen on konteksti? Millä strategialla ja prosessilla räätälöintiä tehdään? Millainen on räätälöinnin tulos? Otettiinko räätälöintiprosessin aikana ja sen jälkeen kokemus talteen?

Räätälöinnin *kohteena* voi olla jokin ketterä menetelmä (esim. Scrum, XP tai Kanban), menetelmät tai niiden osat (esim. jatkuva integraatio, testilähtöinen kehittäminen, kanban-taulu). *Kontekstilla* tarkoitetaan sitä organisaatiota tai kehittämisprojektia, jonka käyttöön kohdetta ollaan räätälöimässä. Räätälöinti tehdään kyseisen kontekstin tilannetekijöiden mukaisesti. Kuviossa 4 esitettiin esimerkki kattavasta tilannetekijöiden jäsennyksestä. Näitä ovat muun muassa organisaation erityispiirteet ja minkälaisia ohjelmistoja se kehittää. *Räätälöintistrategiana* voi olla pilotoinnin ja kokeilun kautta räätälöinti organisaatiolle sopivan menetelmän kehittämiseksi, tai perusteellinen menetelmän suunnittelu, joka edeltää projektien käynnistämistä. *Räätälöintiprosessi* koostuu askeleista, jotka etenevät useimmiten kontekstin tilannetekijöiden tunnistamisesta ja räätälöitävän menetelmän valinnasta menetelmän osien (esim. käytänteiden) valintaan ja ohjelmistokehitysprosessin ja roolien kiinnittämiseen. Viimeinen viitekehyksen osa on räätälöinnin *tulos*, jolla tarkoitetaan lähtökohtana olleesta menetelmästä muokattua menetelmää ja prosessia. Tässä yhteydessä voidaan il-

maista, millä tavalla se eroaa olennaisilta osiltaan lähtökohtana olleesta menetelmästä, mitä prosesseja, rooleja ja käytänteitä räätälöinnin kohteesta on säilytetty, mitä niistä on muutettu ja mitä niistä on jätetty pois.

5.3 Tapaustutkimusten kuvaaminen

Tässä alaluvussa kuvataan valitut tapaustutkimukset yksityiskohtaisemmin. Tapaustutkimuksista käsitellään räätälöinnin pohjana käytetty menetelmä, kohdealueen erityispiirteet, räätälöintiprosessi ja -strategia, räätälöidyn tulos ja se, otettiinko räätälöinnistä kokemukset haltuun tulevaisuutta varten, mikäli näitä asioita on tutkimuksessa käsitelty. Alakohdat on otsikoitu kyseisten kontekstien mukaisesti.

5.3.1 Ulkoistetut elektronisen liiketoiminnan projektit

Hong ym. (2010) tutkivat ketterän menetelmän räätälöintiä ulkoistettujen elektronisen liiketoiminnan projektien tapauksessa. Tapaustutkimuksen kohteena on ollut eteläkorealainen elektronisen liiketoiminnan yritys, joka on käyttänyt kolmansien osapuolten palveluita kehitystyössä saadakseen uudet palvelut ajoissa valmiiksi sekä vähentääkseen kustannuksia. Menetelmänä yrityksellä oli vesiputousmalli. Ulkoistetut projektit eivät kuitenkaan saavuttaneet niille asetettuja tavoitteita, onnistumisprosentin ollessa alhaisempi ja laadun heikompi. Yritys analysoi ulkoistettuja projekteja ja huomasi vesiputousmallin olevan suurin syy epäonnistumiseen. Scrumin käyttämisessä sellaisenaan yritys näki kuitenkin ongelmia. Yrityksen tavoitteena olikin räätälöidä Scrum yrityksen sisäiseksi menetelmäksi ja käytettäväksi ulkoistettujen projektien yhteydessä. Räätälöintiprosessia suoritettiin yrityksen toimesta lähtökohdat huomioon ottaen.

Scrumia muokattiin kolmelta eri osa–alueelta. Scrum–menetelmästä roolien ja vastuiden jakautuminen koettiin liian epäselväksi. Niiden käyttäminen olisi ollut vaikeaa etenkin henkilöille, joille menetelmä ei ollut valmiiksi tuttu ja jotka olivat tottuneet vesiputousmallin mukaisiin rooleihin ja vastuisiin. Ratkaisuna tähän projektitiimin henkilöille annettiin samankaltaiset roolit, joissa he olivat aiemmin toimineet. Scrumin tuoteomistajana toimi suunnittelijatiimi, Scrum–mestarina toimi aiempi projektipäällikkö ja ohjelmiston kehittäjät toimivat tavallisina kehitystiimin jäseninä. Jokaiselle tehtiin selväksi, mitä työtehtäviä tulee kehitysvaiheen aikana tehdä. Tämän havainnollistamiseksi luotiin taulukko, josta ilmenee jäsenten roolit ja vastuut jokaisen sprintin aikana.

Toiseksi projektien ollessa pitkiä Scrumin lyhyisiin tuotejulkaisuihin tähtäävää prosessimallia pidettiin liian epäselvänä, josta syystä kokonaiskuvan luominen oli vaikeaa. Myös kehittäjien *seisonta–aikaa* (engl. *stand–by time)* pidettiin ongelmana. Kokonaiskuvan saamiseksi otettiin käyttöön projektin alussa *pääsprintti (*engl. *master sprint)*, jonka aikana määritellään koko projektin virs-

tanpylväät ja tavoite. Eri yksiköiden johtajat arvioivat tuotteen kehitysjonon sisällön ja ominaisuuksien kehittämiseen tarvittavan ajan. Tämän tiedon avulla pystytään kokonaisuutena arvioimaan, kuinka monta sprinttiä koko projektin aikana tullaan tarvitsemaan ja kuinka pitkiä ne tulevat olemaan. Jokaisen yksittäisen sprintin alkaessa tiimin jäsenten työtehtävät jaetaan kortteihin, jossa näkyy työtehtävän lisäksi vastuuhenkilö, kehitysjonon id–numero ja arvioitu kehittämisaika. Suunnittelijoiden, kehittäjien, koodaajien ja laadunvarmistajien työt pyritään rytmittämään mahdollisimman hyvin niin, ettei turhaa seisonta–aikaa tule.

Kolmantena räätälöinnin kohteena olivat ohjelmiston arviointiin liittyvät kriteerit. Sprinttikatselmoinnissa arviointipisteiden ja edistymiskäyrän sijaan käytettiin mittarina valmiita nettisivuja, ja nämä laitettiin erilliselle taululle tarkasteltavaksi. Nämä sopivat hyvin mittariksi elektronisen liiketoiminnan yritykselle, sillä nettisivujen monimutkaisuus on suurimmalle osalle sivuja samantasoinen. Sprintin aikana keskeneräiseksi jääneet osiot siirrettiin seuraavaan sprinttiin. Projektin arvioitu ja oikea etenemistaso pystyttiin laskemaan tehtyjen ja tekemättömien nettisivujen suhteesta.

Kokonaisuutena yritys sai mitattavia hyötyjä Scrumin räätälöinnistä ulkoistettuihin projekteihin. Vesiputousmallia käyttämällä sisäisten projektin onnistumisprosentti oli vuonna 2009 89 %, mutta ulkoistetuista projekteista 60 % onnistui, 20 % myöhästyi ja 20 % epäonnistui. Ulkoistettujen projektien testien virhesuhde oli merkittävästi korkeampi kuin sisäisten projektien. Räätälöidyn Scrum–menetelmän käyttöönoton jälkeen onnistumisprosentti ulkoistetuilla projekteilla nousi ja virheiden määrä laski. Myös henkilöstön seisonta–aika kehitystyössä väheni, ja tyytyväisyys menetelmän käyttöön oli henkilöstölle tehdyn kyselyn perusteella kasvanut. Kokemuksien talteenotosta tai niiden hyödyntämisestä räätälöinnin aikana ei kerrottu.

5.3.2 Ohjelmistotuoteperheiden suunnittelu

Ohjelmistoalustan tekeminen ohjelmistotuoteperheen pohjalle vaatii pitkäaikaisia investointeja, jotka eivät välttämättä osoittaudu kannattaviksi liiketoimintaympäristössä tapahtuvien muutosten johdosta. Ohjelmistotuoteperheitä kehittävät etenkin isot organisaatiot. Ketterää kehittämistä onkin tämän johdosta ehdotettu käytettäväksi myös ohjelmistotuoteperheiden suunnittelun tapauksessa (Díaz ym., 2011). Díaz ym. (2011) ovat tutkineet Scrumin räätälöintiä ohjelmistotuoteperheiden kontekstissa. Strategiana menetelmäkehittämisessä on ollut konfigurointi, jolla ohjelmistotuoteperheiden konsepteja ja ketterää menetelmää räätälöimällä on pyritty luomaan organisaatioon sopiva menetelmä. Lopputulosta on kutsuttu *ketteräksi ohjelmistotuoteperheiden suunnitteluksi* (engl. *Agile Product Line Engineering*, *APLE*). Tämän jälkeen sitä on sovellettu käytännön työssä testauksen kehitysympäristössä.

Ohjelmistotuoteperheen ja ketterän kehittämisen välillä voidaan nähdä joitain ristiriitoja. Niiden sovittaminen on kuitenkin mahdollista sellaisella tavalla, jossa molempien olennaiset periaatteet säilyvät. Ongelmana on etenkin ohjel-

mistotuoteperheen vaatima arkkitehtuuri- ja pitkäaikaissuunnittelu ketterän kehittämisen korostaessa puolestaan asiakkaalle tuotettua arvoa ja inkrementeissä julkaistua ohjelmistoa. Diáz ym. (2011) ehdottavat kolmea konseptia, joiden avulla Scrum voidaan räätälöidä vastamaan näitä tarpeita. Ensimmäinen konsepti on *mukautuva osakomponentti (*engl. *plastic partial component, PPC).* Kyseiset komponentit ovat erityisen mukautuvia. Ne tukevat niin ulkoista evoluutiota, eli muutoksia arkkitehtuurin konfiguraatiossa, kuin sisäistäkin evoluutiota, eli muutoksia itse komponentin sisällä. Arkkitehtuurin konfiguraation muuttaminen tapahtuu lisäämällä tai poistamalla muita komponentteja ja yhteyksiä. PPC:llä on erilaisia *muunneltavuusskohtia (*engl. *variability points*), jotka määrittelevät sen muunneltavuuden, sekä *variantteja (*engl. *variant)*, jotka liittävät ohjelmakoodin osia siihen. Muunneltavuuskohdat auttavat joustavasti mukautumaan lisäämällä tai poistamalla paloja ohjelmakoodia. Varianttien avulla voidaan määrittää missä ja miten PPC:ä voidaan laajentaa. Näin komponentteja voidaan muokata ja laajentaa milloin vain, ja niitä pystytään kehittämään Scrumin mukaisesti inkrementeissä tehden sprintissä ainoastaan vaadittu toiminnallisuus sille hetkelle sopivaksi.

Toinen konsepti Scrumin räätälöimiseksi on toimivan arkkitehtuurin luominen ohjelmistotuoteperheelle. Tärkeä osa tätä on käyttää arkkitehtuurin suunnittelussa PPC–komponentteja, joiden avulla arkkitehtuuri voi nopeasti vastata muuttuviin olosuhteisiin ja sidosryhmien vaatimuksiin. Arkkitehtuuria kehitetään inkrementaalisesti ja iteratiivisesti, ja se on valmis ottamaan vastaan odottamattomia muutoksia koko ohjelmistonkehityksen ajan.

Kolmas konsepti on *reflektiivinen uudelleenkäyttö (*engl. *reflective reuse)*. Käytäntö soveltaa systemaattista uudelleenkäyttöä tuoteperheiden suunnittelussa sekä opportunistista uudelleenkäyttöä, jota käytetään ketterien menetelmien yhteydessä. Reflektiivisessä uudelleenkäytössä uudelleenkäyttö suunnitellaan aina lyhyeksi aikaa kerrallaan. Scrumin tapauksessa luonteva aika tälle on yksi sprintti. Suunnittelun jälkeen koodia kirjoitettaessa tarvittavat osat tallennetaan uudelleenkäytettävälle ohjelmakoodille luotuun säilöön, mistä niitä voidaan tarpeen mukaan käyttää. Uudelleenkäyttöön liittyviä riskejä voidaan näin pienentää ja välttää suuria investointeja alkuvaiheessa, mutta sitä voidaan silti käyttää ohjelmistoperheiden kehittämisen kanssa strategisesti hyödyksi.

Scrumin prosessimallia on myös muokattu sopimaan paremmin ohjelmistotuoteperheiden suunnittelun erikoispiirteisiin. Esipelivaiheessa sidosryhmät tapaavat ja ohjelmistotuoteperheen kehitysjono muodostetaan, jonka jälkeen kehitysjonon ominaisuuksista tehdään joko yleisiä tai vaihtelevia. Kehitysjonon sisältö kehittyy sprinttien mukana. Toisessa vaiheessa suoritetaan ohjelmistotuoteperheen julkaisunsuunnittelu, jolloin julkaisut ja vaatimukset määritellään ja jaetaan sprintteihin. Yleisten ominaisuuksien priorisoinnissa tulisi käyttää hyödyksi reflektiivisen uudelleenkäytön mahdollisuutta. Seuraavaksi sprintin suunnittelupalaverissa määritetään yksittäisen sprintin tavoite ja työlista.

Suunnittelupalaveria seuraava sprintti kestää 2–4 viikkoa, ja se koostuu kahdesta osasta. Ensimmäisessä osassa suoritetaan sovellusaluekohtaista kehitystyötä. Tämä pitää sisällään PPC–komponenttien suunnittelua ja toteutusta,

ominaisuuksien mallinnusta ja muuttuvien ominaisuuksien määritystä sekä toimivan tuoteperhearkkitehtuurin luomista. Toinen osa pitää sisällään sovelluksen kehitystä ja toteutusta, jolloin muunneltavuuskohdat valitaan ja liitetään tiettyihin variantteihin ja variantit yhdistetään toimiviin tuotteisiin. Tehtävät tämän saavuttamiseksi ovat arkkitehtuurin uudelleenkonfigurointi lisäämällä tai poistamalla vaihtelevia komponentteja ja liitoksia sekä PPC–komponenttien lopullinen määritteleminen liittämällä ominaisuuksia sen toiminnallisuuteen. Sprintin lopuksi on katselmus ja retrospektiivi, joiden jälkeen palataan julkaisusuunnitteluvaiheeseen. Samalla tässä vaiheessa voidaan käyttää sprinteistä saatua palautetta hyödyksi reflektiivisen uudelleenkäytön suunnittelussa.

Ketterää ohjelmistotuoteperheiden suunnittelua on hyödynnetty käytännössä järjestelmien testausympäristöjen TOPEN–ohjelmistoperheen (Test and OPeration ENvirnonment) kehitystyössä. Järjestelmällä testataan erilaisia *testauksen alla olevia järjestelmiä (*engl. *systems under test, SUT).* Testauksen alla olevat järjestelmät vaativat testausympäristöltään erilaisia ominaisuuksia ja järjestelmät kehittyvät jatkuvasti, jolloin ketterä kehittäminen sopi lähestymistavaksi siihen. Ohjelmistotuoteperhearkkitehtuuri kehitettiin iteratiivisesti, ja se tuki erilaisia muunneltavuuskohtia ja niiden variantteja. Ketterällä ohjelmistotuoteperheiden suunnittelulla saatiin joustavuutta kehitykseen. Kyseisen projektin aikana tapahtuneesta kokemuksien haltuunotosta ei kuitenkaan mainittu, vaan räätälöinti tapahtui ennen projektia.

5.3.3 Julkinen sektori

Calgary on Kanadan neljänneksi suurin kaupunki, jossa on yhteensä noin miljoona asukasta. Sillä on 28 liiketoimintayksikköä ja 14 000 työntekijää, joista 400 on informaatioteknologian ammattilaisia työtehtävien vaihdellessa IT–infrastruktuurin ylläpidosta ohjelmistokehitykseen. Organisaation kulttuurille on useita laajoille ja vakiintuneille organisaatioille ominaisia piirteitä. Niitä ovat rajoittavat käytännöt, menettelytavat ja prosessit, tarkasti määritellyt roolit ja vastuut sekä hierarkkiset valtuuttamis- ja raportointirakenteet. Ympäristössä on useita ongelmia, jotka voivat haitata ketterien menetelmien käyttämistä. Näitä ovat muun muassa asiakkaiden heikko osallistuminen kehitykseen sekä päätöksenteon ja valtuutuksen hitaus. (Scott ym., 2008.)

Ketterän menetelmän räätälöintistrategiaksi oli valittu pilottiprojektin tekeminen Calgaryn kaupungille kustomisoiden projektin menetelmä valikoimalla prosesseja ja käytänteitä pääasiassa Scrumista ja XP:stä. Samalla tarkoitus oli kehittää organisaatiokulttuuria kohti ketterän kehittämisen periaatteita ja arvoja. Tutkimuksessa ei mainittu perusteita, joilla käytännöt valittiin Scrumista ja XP:stä. Organisaatio osti myös mentorointipalveluita ulkopuoliselta taholta menetelmän käyttöönotossa. (Scott ym., 2008.)

Pilottiprojektin jäsenet koottiin yhteen tilaan eri puolelta organisaatiota olleista ohjelmistokehityksen ja ohjelmistotuen osastoilta. Onnistumista kohtaan oli myös paljon ennakkoluuloja ja pelkoja. Viestintä oli alkuun vaikeaa, sillä projekti käynnistettiin varsin nopeasti eikä tiimin jäsenillä ollut paljoa koke-

musta asiakkaiden vaatimusten kirjauksesta. Kehitystiimi oli tottunut tulemaan kehitystyöhön mukaan vasta myöhemmin projektissa, mutta pilottihankkeessa käyttäjätarinoita keräävä työpaja valjastettiin käyttöön aikaisin. Sisäisten tapaamisten ja asiakastapaamisten lisääntyessä myös viestintä parani ja käyttäjätarinoiden keräämisestä tuli luonnollisempaa. Samalla käyttäjätarinoiden kanssa otettiin käyttöön tuotteen kehitysjono, johon vaatimukset kirjattiin. Tiimillä oli myös käytössään erillinen huone, jossa palavereita Scrumin tapaan järjestettiin. Projektilla oli myös erillinen seinä, jossa työtehtäviä ja niiden edistymistä seurattiin, testattavia tehtäviä määritettiin ja projektiin liittyviä riskejä hallittiin.

Kehitystyö oli iteratiivista. Iteraatiopalavereissa järjestettiin työtehtävät ennen iteraation aloittamista. Samaan tapaan sprintin jälkeen kehittäjätiimi teki iteraation sisällöstä arvion ja teki muutoksia palautteen perusteella. Alkuun iteraatioiden suunnittelu oli hidasta johtuen osittain kulttuurimuutoksesta, sillä kehittäjät organisoituivat nyt itse itsensä eivätkä työtehtävät tulleet enää ylemmältä osapuolelta. XP:n käytännöistä otettiin käyttöön testivetoinen kehittäminen ja pariohjelmointi. Testivetoisuus omaksuttiin varsin nopeasti organisaatiossa, ja kolmannen iteraation lopussa tiimin kaikki jäsenet olivat huomanneet sen arvon ja kehittivät ohjelmistoa käytänteestä huolimatta yhtä nopeasti kuin aiemmin.

Projektin edetessä ja iteraatioiden jatkuessa tiimin tuottavuus kohosi merkittävästi ja kehitystyön laatu oli korkealla. Mentoroinnin määrä väheni, mutta silti palautetta oli mahdollista saada organisaation ulkopuolelta. Organisaatiokulttuuri muuttui huomattavasti, eikä ihmisten rooleja enää määritelty yhtä tiukasti kuin aiemmin. Kehittäjät pystyivät itse vaikuttamaan työtehtäviinsä, tunsivat asiakkaansa, hallitsivat ja keskittyivät työtehtäviinsä paremmin ja ohjelmiston laadulle annettiin suurempi painoarvo. Myös asiakkaat olivat tyytyväisiä, koska he kokivat omistavansa kehitetyn ohjelmiston ja olevansa osa tiimiä. He näkivät toimivan ohjelmiston aikaisin ja pystyivät vaikuttamaan projektisuunnitelmaan ja projektin edistymiseen.

Pilottiprojektissa otettiin käytänteitä ja prosesseja niin Scrumista kuin XP:stä. Huolimatta onnistuneesta projektista koko organisaation kulttuurin ja kaiken kehitystyön muuttamiseksi ketteräksi oli monenlaisia haasteita. Projekti kuitenkin osoitti, että ketterä kehittäminen voidaan räätälöidä myös hierarkkiaallisiin organisaatioihin. Projekteja toteuttaessa tulee ottaa huomioon riittävä organisaation tuki uuden menetelmän käyttöönotolle.

Kokemusten haltuunotto tapahtui vaiheittain. Räätälöinnin aikana mentorointi auttoi kehittäjiä räätälöityjen käytänteiden omaksumisessa. Tällöin tiimi sai räätälöintiprosessin aikana otettua talteen hyödyllistä tietoa, jonka avulla räätälöintiä pystyttiin jatkamaan. Kehittäjät olivat kokeneempia menetelmien käytössä, mikä auttoi prosessissa. Pilottiprojektin aikana saatiin kokemusta, jota voidaan hyödyntää jatkossa, kun ketterien menetelmien käyttö jatkuu pilottiprojektin ulkopuolella. Näitä olivat muun muassa huomio siitä, että tietyt roolit ja vastuut kaipaavat selventämistä ja testaamisen tulee olla läpinäkyvämpää.

5.3.4 Tietoturvallisuuspalveluihin keskittynyt kaupallinen tuote

Wang ym. (2012) ovat tehneet useita tapaustutkimuksia ketterän kehittämiseen liittyen etenkin *prosessi–innovaatioiden* (engl. *process innovation)* sulauttamisen näkökulmasta. Yksi näistä on tietoturvallisuuspalveluihin keskittynyt keskikokoinen yritys, jolla on toimintaa 90 maassa. Ennen ketterien menetelmien käyttöönottoa yritys käytti vesiputousmallia. Ohjelmistojen laatua ja luotettavuutta pidettiin hyvänä, mutta johto päätti kuitenkin ottaa ketteriä menetelmiä käyttöön, jotta markkinoilla tapahtuviin muutoksiin pystyttäisiin vastaamaan nopeasti. Tiimi, joka toimi yhtenä pilottiprojektina, oli kuusihenkinen, ja sille järjestettiin koulutusta niin Scrumista kuin XP:stäkin. Pilottiprojektin tekemisestä oli kulunut kirjoitushetkellä 1,5 vuotta ja tiimi oli toiminut ketterästi sen jälkeen, joten käytännöt ovat muotoutuneet vielä räätälöinnin jälkeenkin. Strategiana räätälöinnillä olikin konfigurointi kehittäjätiimin käyttämäksi menetelmäksi.

Projektitiimi toimi 2–4 viikon mittaisissa sprinteissä. Suunnittelupalaverin jälkeen vaatimukset sijoitettiin tuotteen kehitysjonoon. Tiimi ei kuitenkaan itse valikoinut tai määritellyt vaatimuksia sprintin ajaksi, mutta organisoi itsensä jokaisen sprintin ajaksi. Tiimi piti päivittäisiä palavereita, mistä tuli tiimin sisällä tehokas ongelmienratkaisun väline. 40 tunnin työviikon käytäntöä tiimi piti yllä ja ajoitti työtehtävät sprintin ajalle niin, ettei ylitöiden tekeminen ollut tarpeellista. Paikalla oleva (on–site) asiakkaan käytäntöä taas ei pystytty täysin soveltamaan. Yksinkertaisen suunnittelun periaate tuli käyttöön etenkin alun jälkeen kehittäjien tullessa paremmiksi ketterien menetelmien käytössä, jolloin monimutkaisten ratkaisujen sijaan nojattiin yksinkertaiseen lähestymistapaan. Teknisistä esteistä johtuen käytäntöä ei kuitenkaan pystytty aina soveltamaan.

Pariohjelmointia tiimi ei noudattanut, mutta koodikatselmukset ja uusien työntekijöiden perehdytys tehtiin pareissa. Testivetoista kehitystä noudatettiin tiimin sisällä voimakkaasti. Lisäksi tiimi käytti jatkuvaa integrointia huolimatta joistain teknisistä ongelmista, joita se kohtasi ottaessaan käytäntöä toimeen. Saatuaan ongelmat ratkaistua tiimi alkoi tehdä automaattista *koodinkokoamista ja testausta* (engl. *build)* tunneittain. Jatkuvan integraation avulla testausta saatiin tehtyä merkittävästi helpotettua.

Koodin yhteisomistajuus oli myös käytössä vielä pidemmälle vietynä kuin XP:ssä on alun perin määritelty. Se päti niin ohjelmakoodiin kuin myös muihin työskentelyresursseihin ja tuloksiin, eikä kukaan yksittäinen kehittäjä saanut omistusta jonkun ratkaisun kehittämisestä. Refaktorointia tiimi suoritti ainoastaan järjestelmän monimutkaisten osien yhteydessä, sillä refaktoroinnin suorittaminen aiheutti korkeita kustannuksia. Tiimi käytti myös avointa työtilaa.

Tapaustutkimuksen kehittäjätiimi käytti monia Scrumin ja XP:n käytänteistä, ja se oli vienyt monet käytänteistä pidemmälle kuin mitä virallisissa ohjeissa on sanottu eri käytännöistä. Tiimi oli huomannut käytänteiden käyttämisestä selvää hyötyä. Tämä näkyi muun muassa jatkuvan integraation edistyneen käytön helpottaessa testausta merkittävästi. Räätälöinti on tällöin jatkuva prosessi, jossa tiimi jatkuvasti arvioi ja muuttaa käytänteitä. Kokemuksien hal-

tuunoton takana olevaa prosessia ei kuitenkaan Wangin ym. (2012) tutkimuksessa esitelty kyseisen tapauksen kohdalla.

5.3.5 Suorittimien ohjelmistokehitykseen keskittynyt organisaatio

Fitzgerald ym. (2006) ovat tutkineet Intel Shannonia, joka on Intelin suorittimiin keskittynyt osasto Irlannissa. Intel Shannonilla oli tutkimuksen aikaan töissä 125 henkilöä, joista 90 oli ohjelmistokehitystöissä. Yleisesti sen kehittämien tuotteiden vaatimusmäärittely oli valmiiksi tehty Yhdysvalloissa, ja ohjelmistosekä piisirujen suunnittelu tehtiin Irlannissa. Tutkimuksella selvitettiin miten Scrumia ja XP:tä räätälöitiin käyttöönotossa organisaatiossa. Yritys oli käyttänyt sisäisesti *prosessien kypsyysmallin (*engl. *capability maturity model, CMM)* tason 2 mukaista prosessimallia, mutta se oli siirtynyt markkinoiden vaatiessa käyttämään myös ketteriä menetelmiä. Tutkimuksen teon aikaan Intel Shannon oli käyttänyt XP:tä viisi vuotta ja Scrumia kolme vuotta, Scrumia projektin hallintaan ja XP:ä varsinaiseen ohjelmistokehitykseen. Menetelmän räätälöintiä organisaatiossa vei eteenpäin joukko asiaan sitoutuneita menetelmäeksperttejä.

XP:stä oli valikoitu käytettäväksi joitain käytänteitä, mutta ei läheskään kaikkia. Pariohjelmoinnin käytöstä oli saatu erilaisia hyötyjä, kuten vaaditun ohjelmakoodin laatutason saavuttamista kehitystyössä aiemmin ja koodivirheiden vähenemistä. Se olikin käytänteenä yleisesti käytössä, mutta joihinkin soveltumattomiin vaiheisiin sitä ei käytetty. Testivetoista kehitystä oli sovellettu niin, että yksikkötestit kirjoitettiin samaan aikaan kun toteutus tehtiin. Tämän oli huomattu antavan suunnan kehitykselle ja ymmärrystä kehitettävästä kohteesta. Refaktorointi oli käytössä, ja sen oli havaittu vähentävän virheitä. Yksinkertainen suunnittelu oli refaktorointiin yhteydessä, ja sitä sovellettiin tekemällä suunnittelu valkotaululle jokaisen ohjelmakoodilohkon tapauksessa, jolloin toteutusta voitiin tehdä suunnittelun kanssa samanaikaisesti. Yhteisomistajuuden käytäntö toteutui Intel Shannonilla, mutta se rajoittui ainoastaan kehitystiimien sisälle, ei tiimien välille. Koodausstandardit taas olivat vahvasti käytössä jo ennen ketterien menetelmien tuloa.

Intel Shannonilla jätettiin käyttämättä monia XP:n käytäntöjä. Suunnittelupeliä ei käytetty, vaan sen sijasta käytettiin Scrumin käytäntöjä. Pienten julkaisujen käyttäminen ei soveltunut, sillä suorittimien tapauksessa ohjelmistojulkaisut on sidottu niihin. Jatkuvaa integrointia sovellettiin jokaiselle komponentille erikseen, mutta kokonaisvaltaista integrointia ei tehty kuin vasta ennen julkaisua. 40 tunnin työviikko taas ei soveltunut käyttöön kehitysympäristön takia, koska Yhdysvaltojen ja Euroopan välinen aikaero saattoi pidentää viikkotunteja. Paikalla olevaa asiakasta taas ei ollut saatavilla, joten kyseisen käytännön soveltaminen ei myöskään ollut mahdollista.

Scrum oli ollut Intel Shannonilla käytössä kolme vuotta, ja osana melkein jokaista kehitystiimiä. Esipelivaiheessa suunnittelu oli yksinkertaista eikä töistä tehty aika–arvioita tai mahdollisia kytköksiä muihin työtehtäviin. Projektin alussa tehtiin alustavat arviot sprinteistä, jolloin ne arvioitiin yleisellä tasolla. Projektiin ja sprintteihin liittyviä epävarmuustekijöitä otettiin myös huomioon,

ja pitkiä työtehtäviä jaksotettiin useamman sprintin ajalle. Ohjelmistoarkkitehtuuri oli pitkälle ennalta määrättyä piisirujen tapauksessa. Jokaisen sprintin alussa tiimi päätti, mitä he sen aikana tulevat tekemään käyttäen lähtökohtana projektin sprinttisuunnitelmaa ja mahdollisia tuotteen kehitysjonoon tulleita kohtia. Sprintti ”suojelee” tiimiä ympäristön vaikutuksilta sen aikana. Projektin lopussa listattiin ylimääräiset työtehtävät, mitä ei ollut alkuperäiseen suunnitelmaan merkitty, luotiin mahdollinen demonstraatio kehitystyön kohteesta ja kirjoitettiin yhteenveto projektista.

Ketterien menetelmien hyötyinä todennettiin Intel Shannonilla parantunut ohjelmakoodin laatu ja virheiden väheneminen tuotteissa. Ketteriä menetelmiä ei sovellettu sellaisenaan, vaan pragmaattisesti valittiin sopivimmat käytännöt käyttöön. Räätälöinti oli edistynyttä, sillä sitä sovellettiin kahteen menetelmään samanaikaisesti. Kokemuksien haltuunottamista suoritettiin Intel Shannonilla. Käytäntöjen valitsemisen jälkeen niitä valvottiin ja niiden vaikutusta mitattiin. Lisäksi projektien jälkeen XP:ä ja Scrumia verrattiin keskenään, ja kehittäjät myös kertoivat, millaiseksi he uskoisivat ohjelmiston tulleen, mikäli ketterien menetelmien sijaan olisi käytetty perinteisiä menetelmiä.

5.3.6 Vakuutusyhtiön IT-osasto

ASR on yksi kolmesta suuresta alankomaalaisesta vakuutusyhtiöstä. Se toimii riskejä välttävällä ja konservatiivisella teollisuuden alalla. Sen IT-osasto koostuu noin 200 työntekijästä, ja kehitystiimit ovat jakautuneet projekteittain käytettävän teknologian ympärille. Kehityksen lisäksi tiimit ovat vastuussa ohjelmiston ylläpidosta ja toiminnasta. Ohjelmistojen tuotantoversioita ei yleensä päivitetty kuin muutaman kerran vuodessa. (Maassen & Sonnevelt, 2010.)

Vuosina 2006 ja 2007 muutospyyntöjen määrä kasvoi, jolloin ylläpidon vaatimusten priorisointi vaikeutui ja läpimenoaika pidentyi merkittävästi. Tämän johdosta organisaatio kokeili käyttää Scrumia kehittämismenetelmänä. Suurin osa projekteista, jotka käyttivät Scrumia, olivat onnistuneita, mutta sen käytössä nähtiin myös ongelmia. Toiminta ja ylläpidon pienet muutokset tuottivat vaikeuksia, sillä niitä oli vaikea pilkkoa osiin lyhyiden sprinttien aikajaksoille. Asiakkaat halusivat enemmän joustavuutta, ja organisaatio näki tarpeen visuaalisille keinoille johtamisessa ja jatkuvalle parantamiselle. Muun muassa näiden syiden takia ASR päätyi kokeilemaan Kanbania, pilotoiden sitä ensin yhdellä projektilla.

Organisaatio keskittyi kolmeen Kanbanin periaatteeseen, jotka olivat työn visualisoiminen, keskeneräisen työmäärän rajoittaminen ja työnkulun helpottaminen. Kanbanin räätälöintiä johdettiin valmentajien ja päälliköiden johdolla. Työn visualisoimiseksi käytettiin kanban-taulua, jonka sarakkeet räätälöitiin sopimaan kehitystiimin prosessia. Tauluna käytettiin yksinkertaista valkotaulua, ja töille oli erivärisiä lappuja viisi kappaletta. Visualisoinnin lisäämiseksi lappuun laitettiin piste jokaiselle päivälle, kun työ ei edennyt.

Keskeneräistä työmäärää rajoitettiin Kanbanin ohjeiden mukaisesti asettamalla vaihekohtaiset rajat. Tämän lisäksi jokaisella tiimin jäsenellä oli ainoas-

taan kaksi magneettia, jonka johdosta he pystyivät aloittamaan uuden työn ainoastaan, mikäli vanha valmistui. Näiden avulla prosessista paljastui pullonkauloja. Päivittäiset tapaamiset olivat nopeita, sillä kanban-tauluun saatiin visualisoitua muiden työvaiheet, ja retrospektiiviset tapaamiset auttoivat jatkuvassa parantamisessa. Pilotoinnin jälkeen ASR laajensi Kanbanin käyttöä useampaan kehitystiimiin, jotka hiljalleen ottivat sen käyttöönsä. Jokainen tiimi räätälöi oman prosessin ja oman kanban-taulun, mutta taulun lappujen värit on pidetty tiimeissä samoina sekaannusten välttämiseksi. Kanbanin käytöstä nähtiin erityistä hyötyä syntyvän ylläpitoympäristössä.

ASR kohtasi myös haasteita Kanbanin räätälöimisessä. Tiimin kohdistuvina haasteina olivat muun muassa vastustus keskeneräisen työmäärän rajoittamista vastaan, henkilökohtaisesta tuottavuudesta huolehtiminen ja osa-aikaisesti tiimin kanssa työskentelevät henkilöt. Organisaatioon kohdistuvina haasteina olivat muun muassa hajautetut tiimit ja organisaation yleiset käytänteet, jotka olivat Kanbanin käytänteitä jäykemmät. Haasteiden aiheuttamia ongelmia kuitenkin pystyttiin osittain ratkaisemaan ja tekemään niistä näkyviä.

Maassen ja Sonnevelt (2010) kokosivat kokemuksiensa perusteella räätälöintiohjeita, jotka on esitetty taulukossa 10. Ohjeista on hyötyä muille organisaatioille ja projekteille, jotka ovat räätälöimässä Kanbania käyttöönsä.

TAULUKKO 10 Kanbanin räätälöintiä koskevia ohjeita (Maassen & Sonnevelt, 2010)

Aloita yhdellä kehitystiimillä ja laajenna kertyneen kokemuksen pohjalta, tehden etenemisestä mahdollisimman helppoa. Jatka periaatteiden tarkkailemista.
Aloita hitaasti ja esittele uusia käytänteitä vasta, kun tiimi ja organisaatio ovat valmiita niihin.
Anna tiimin omistaa prosessi ja kanban-taulu
Etene päivittäisissä palavereissa asioiden käsittelyssä kanban-taulun mukaisesti oikealta vasemmalle, ja kysy, onko käsiteltävän sarakkeen työtehtävissä mitään päivitettävää
Yhdenmukaista työtehtävien värit kanban-tauluissa eri kehitystiimissä
Valkotaulu ja post-it -laput ovat riittäviä työkaluja alkuun, sillä elektronisten työkalujen käyttämisellä on omat ongelmansa
Viesti jatkuvasti (haastattelut, kahvipöydän keskustelut, esitykset)
Luo ja käytä johdon tarjoamaa tukea
Jaa onnistumiset tiimin ja johdon kanssa

5.4 Yhteenveto tapaustutkimuksista

Tässä luvussa esiteltiin ketterän menetelmän räätälöintiä koskevia tapaustutkimuksia. Seuraavaksi esitetään yhteenveto tutkimuksista. Esitys on jäsennetty aiemmin tässä luvussa esitetyn viitekehyksen mukaisesti.

Taulukossa 11 on yhteenveto käsiteltyjen tapaustutkimusten pohjamenetelmistä, kontekstista ja erityispiirteistä. Kahdessa tapaustutkimuksessa oli räätälöinnin kohteena Scrum, kolmessa Scrumin ja XP:n yhdistelmä ja yhdessä tapaustutkimuksessa Kanban. Hongin ym. (2010) tapaustutkimus käsitteli ulkois-

tettuja elektronisen liiketoiminnan projekteja, Díaz ym. (2011) käsittelivät ketterää ohjelmistotuoteperheiden suunnittelua, Scott ym. (2008) julkista sektoria ja Wang ym. (2012) monikansallisen yrityksen tietoturvapalveluihin keskittynyttä projektitiimiä. Fitzgerald ym. (2006) käsittelivät suorittimien ohjelmistokehitykseen keskittynyttä organisaatiota ja Maassen sekä Sonnevelt (2010) käsittelivät alankomaalaista vakuutusyhtiötä. Díazin ym. (2011) tutkimuksessa räätälöintiä suoritettiin yhtä projektia varten, mutta muissa tutkimuksissa räätälöintiä suoritettiin organisaatiota varten. Räätälöintiprosessia ja -strategiaa selventäviä kuvauksia sisältyi tutkimuksiin vähän. Prosessin eri osia saatettiin kertoa, mutta koko prosessin avaavaa tarkkaa kuvausta ei ollut. Yksittäisiä prosessin osia saattoivat olla esimerkiksi tuen hakeminen organisaation ulkopuolelta tai räätälöinnin suorittaminen projektin aikana ilmenneiden tilannetekijöiden mukaisesti.

TAULUKKO 11 Yhteenveto ketterän menetelmän räätälöintiä käsittelevien tapaustutkimuksien pohjamenetelmästä, kontekstista ja räätälöintiprosessista

Lähde	**Pohjamenetelmä**	**Konteksti ja erityispiirteet**	**Räätälöintiprosessi ja -strategia**
Hong ym. (2010)	Scrum	Ulkoistetut elektronisen liiketoiminnan projektit	Räätälöity Scrum vastaamaan organisaation tarpeita ja otettu käyttöön
Díaz ym. (2011)	Scrum	Ohjelmistotuoteperheiden suunnittelu	Ketterän kehittämisen ja ohjelmistotuoteperheiden käsitteiden sovittaminen yhteen ennen projektin käyttöä
Scott ym. (2008)	Scrum & XP	Julkinen sektori (Calgaryn kaupunki)	Pilottiprojekti, jossa räätälöinti tapahtui ulkoista konsultointia hyödyntämällä
Wang ym. (2012)	Scrum & XP	Tietoturvallisuuspalveluihin keskittynyt keskisuuri organisaatio	Pilottiprojekti yksittäiselle kehitystiimille, ulkopuolisen konsultaation suuri hyödyntäminen
Fitzgerald ym. (2006)	Scrum & XP	Suorittimien ohjelmistokehitykseen keskittyvä organisaatio	Räätälöity organisaatiolle ottamalla soveltuvat osat käyttöön ja jättämällä loput pois
Maassen & Sonnevelt (2010)	Kanban	Vakuutusyhtiö	Pilottiprojektin jälkeen usealle tiimille käyttöön, tiimit ovat räätälöineet oman prosessinsa

Taulukossa 12 on yhteenveto käsiteltyjen tapaustutkimusten räätälöidyistä menetelmistä ja niiden eroista asianomaisiin pohjamenetelmiin sekä se, mainittiinko tutkimuksessa sitä, tapahtuiko räätälöinnin aikana kokemusten haltuunottoa.

Ulkoistettujen elektronisen liiketoiminnan projektien tapauksessa (Hong ym., 2010) Scrum räätälöitiin, jotta projektien onnistumista voitaisiin parantaa ja laatua parantaa. Räätälöity menetelmä eroaa Scrumista siinä, että projektin

henkilöiden roolit ja vastuut oli taulukoitu, projektin alkuun lisättiin pääsprintti ja mittarina käytettiin valmiita nettisivuja. Ohjelmistotuoteperheen tapauksessa (Diaz ym., 2011) tehtiin muokkauksia niin ohjelmistotuoteperheiden käsitteisiin kuin myös ketterään menetelmään. Toteutuksessa käytettiin muokkautuvia osakomponentteja, arkkitehtuuri luotiin toimivaksi vastaamaan muuttuvia tekijöitä ja sidosryhmien vaatimuksia sekä käytettiin reflektiivistä uudelleenkäyttöä hyödyksi. Samalla Scrumin prosessimalli muokattiin tukemaan ohjelmistotuoteperheiden suunnittelua. Julkisen sektorin tapauksessa (Scott ym., 2008) tehtiin pilottiprojekti Calgaryn kaupungille. Tällöin kustomisoitiin Scrumin ja XP:n käytäntöjä projektin menetelmäksi käyttäen konsultointia hyödyksi ja käytänteet valittiin sopivuuden mukaan.

TAULUKKO 12 Yhteenveto ketterän menetelmän räätälöintiä käsittelevien tapaustutkimuksien räätälöidyn menetelmän eroavaisuuksista pohjamenetelmään ja kokemusten haltuunotosta

Lähde	**Räätälöity menetelmä ja sen eroavaisuudet pohjamenetelmään**	**Kokemusten haltuunotto**
Hong ym. (2010)	Henkilöiden roolit ja vastuut taulukoidaan, projektin alkuun pääsprintti, mittarina käytetään valmiita nettisivuja	Ei erikseen kerrottu
Díaz ym. (2011)	Mukautuvien ohjelmistokomponenttien luominen, niiden käyttäminen arkkitehtuurin luomisessa, reflektiivinen uudelleenkäyttö, Scrumin prosessimalli sovitettu ohjelmistotuoteperheiden mukaiseksi	Ei erikseen kerrottu
Scott ym. (2008)	Organisaatioon sopivien Scrumin ja XP:n käytänteiden käyttöönotto ja omaksuminen hitaammalla aikataululla	Projektin aikana, ja sen jälkeen tulevia projekteja varten
Wang ym. (2012)	Suurin osa käytänteistä otettu käyttöön, osa käytänteistä viety alkuperäistä ohjetta pidemmälle. Muutama XP:n käytäntö jätetty käyttämättä	”Jatkuva” räätälöinti, jolloin käytänteitä vietiin pidemmälle
Fitzgerald ym. (2006)	Noin puolet XP:n käytänteistä käytössä ja puolet ei, Scrumin prosessimallia muokattu hieman tukemaan organisaation erityispiirteitä	Käytänteiden vaikutuksen mittaus, projektien jälkeinen pohdinta
Maassen & Sonnevelt (2010)	Tiimeittäin kanban-taulun räätälöiminen, yhteinen värikoodi työtehtävillä, työn visualisoiminen, henkilökohtaiset rajat työlle	Pilottiprojektin kokemusten hyödyntäminen, ohjelista räätälöinnille

Tietoturvapalveluihin keskittyneen yrityksen tapauksessa (Wang ym., 2012) Scrum ja XP oli otettu käyttöön 1,5 vuotta aiemmin ja niiden käytännöt oli valittu tai jätetty valitsematta sopivuuden mukaan. Osa käytännöistä oli viety ohjekirjan mallia pidemmälle. Suorittimien ohjelmistokehityksen tapauksessa (Fitzgerald ym., 2006) Scrum ja XP oli räätälöity organisaation käyttöön, Scrumin ollessa käytössä projektin hallinnassa kolme vuotta ja XP käytännön työssä viisi vuotta. Scrumin prosessimallia oli muokattu sopivaksi ja XP:n käytännöistä osa oli otettu käyttöön ja osa jätetty käyttämättä sovellusalueeseen sopimattomana.

Vakuutusyhtiön tapauksessa (Maassen & Sonnevelt, 2010) Scrum oli nähty puutteelliseksi etenkin ohjelmiston ylläpito- ja toimintaympäristössä, jonka johdosta organisaatio räätälöi Kanbanin ensiksi yhdelle kehitystiimille pilottihankkeena, mutta tämän jälkeen useammalle kehitystiimille. Jokainen tiimi räätälöi Kanbanin prosessiltaan vastaamaan omaa työtapaansa.

6 POHDINTA

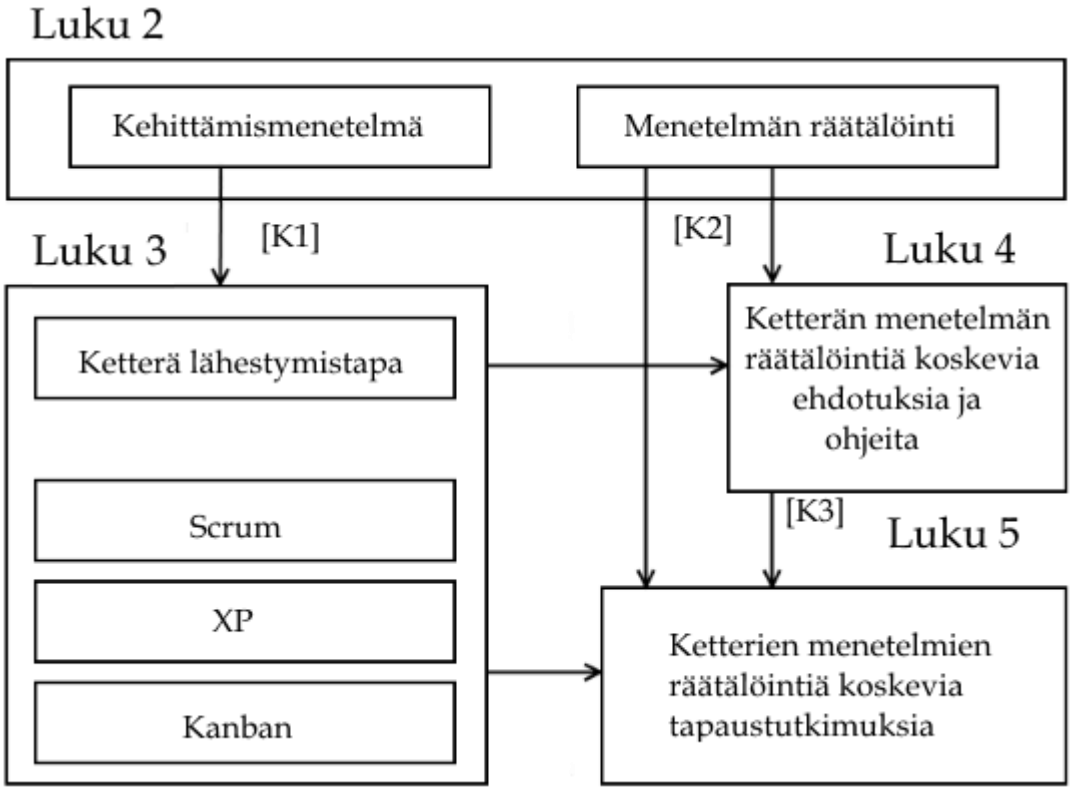
Tämän tutkimuksen tavoitteena on ollut selvittää, millä tavalla ketteriä menetelmiä on esitetty räätälöitäväksi, millä tavalla tätä räätälöintiä on tehty käytännössä ja millaisin kokemuksin. Näihin kysymyksiin vastaamiseksi on työssä ensin selvitetty, mitä menetelmällä tarkoitetaan tietojärjestelmien kehittämisen yhteydessä, mistä menetelmä koostuu ja miten sitä voidaan kehittää ja erityisesti räätälöidä. Toiseksi työssä on kuvattu ketterää lähestymistapaa ja kolmea ketterää menetelmää, Scrumia, XP:tä ja Kanbania. Kolmanneksi tutkielmassa on valittu perusteelliseen kirjallisuushakuun perustuen joukko tutkimuksia ja kuvattu niiden esityksiä ja ohjeistuksia ketterien menetelmien räätälöinnistä. Neljänneksi on samaan kirjallisuushakuun nojaten kuvattu kuusi tapaustutkimusta, jotka koskevat ketterän menetelmän räätälöintiä joko tiettyä organisaatiota tai projektia varten, ja verrattu niitä keskenään.

Yllä mainitut asiat antavat mahdollisuuden monenlaisiin pohdintoihin, joiden jäsentämisessä käytetään kuviota 14. Pohdinta keskittyy erityisesti kolmeen aiheeseen, jotka liittyvät seuraaviin tutkimuskysymyksiin:

- K1: Millaisista osista ketterät menetelmät koostuvat?
- K2: Millä tavalla yleisestä menetelmäkehityksestä ja -räätälöinnistä esitetyt ajatukset ovat nähtävissä ketterien menetelmien räätälöinnille annetuissa jäsennyksissä ja ohjeissa?
- K3: Millä tavalla ketterien menetelmien räätälöinnille esitettyjä jäsennyksiä ja yleisiä ohjeita on noudatettu tapaustutkimuksen kohteena olleissa tapauksissa?

Ensimmäinen kysymys on relevantti siksi, että yleisimmän käsityksen mukaan (esim. Brinkemper ym., 1999; Karlsson & Ågerfalk, 2004) menetelmän räätälöinti perustuu menetelmäkomponenttiajatteluun. Tämän mukaan räätälöinnissä jätetään joitakin osia menetelmästä pois, lisätään joitakin osia ja/tai muokataan joitakin osia paremmin tilanteeseen sopivaksi. Tästä syystä on olennaista tietää, millaisia ovat ketterien menetelmiä osat. Toinen kysymys on mielenkiintoinen siksi, että yleistä menetelmäkehitystä ja räätälöintiä koskevaa tutkimusta on

tehty muutama vuosikymmen ennen ketterien menetelmien ilmaantumista. Tällöin räätälöinnin kohteina olivat usein laajat ja kankeat menetelmät, joita tulikin räätälöidä ennen kuin ne voidaan ottaa käyttöön. Kolmas kysymys liittyy ketterän menetelmän räätälöintiä koskevien yleisten jäsennysten ja ohjeistojen ja käytännön suhteeseen. Käytäntöä edustavat tässä työssä kuvatut tapaustutkimukset. On mielenkiintoista selvittää, mitä jäsennyksistä ja ohjeistoista voidaan nähdä sovelletun käytännön tilanteissa.



KUVIO 14 Tutkielman lukujen sisältö ja pohdinnan aiheet jäsennettynä

Seuraavaksi vastataan ensin kyseisiin kysymyksiin ja sen jälkeen esitetään johtopäätöksiä.

K1: Millaisista osista ketterät menetelmät koostuvat?

Luvussa 3 käsiteltiin ketterää kehittämistä. Eri ketterillä menetelmillä on erilainen fokus ja ne rakentuvat erilaisista osista. Scrum (Schwaber & Sutherland, 2013) tarjoaa käytänteitä, jotka auttavat etenkin ohjelmistoprojektin läpiviemisessä ja tiimityössä. Tällaiset käytänteet kattavat esimerkiksi erilaiset palaverit ja tuotteen kehitysjonon. XP:n (Beck, 1999) fokus on itse kehitystyössä, ja monet sen käytännöt, kuten jatkuva integraatio ja pariohjelmointi, keskittyvät ohjelmistoprojektin teknisiin seikkoihin. Kanbanin (Anderson, 2010) fokus on työkulun sujuvoittamisessa. Se ei esitä teknisiä käytänteitä. Rakenteeltaan Scrum ja XP ovat samantapaisia niin periaatteiltaan, lähestymistavaltaan ja taustaltaan. Kanbanin lähestymistapa ja periaatteet taas perustuvat Lean-ajatteluun, ja sen

taustat ovat teollisuustuotantoympäristössä. Scrum ja XP tarjoavat prosessimallin ohjelmistokehitykseen, ja prosessia tuetaan erilaisilla rooleilla. XP:n prosessimalli ja roolit eivät ole kuitenkaan laajasti käytössä, eikä yksikään luvussa 5 tutkittu tapaustutkimus käyttänyt niitä. Kanbania käytettäessä suositellaan, että organisaatio käyttää jo olemassa olevia rooleja ja prosesseja.

Yllä olevasta voidaan todeta, että Scrum ja XP tarjoavat mahdollisuuksia integroivaan räätälöintiin, jossa tietyn projektin tarpeiden mukaisesti voidaan valita Scrumista prosessimalli ja projektinhallinnallisia käytänteitä ja XP:stä teknisempiä käytänteitä. Toinen mahdollisuus on integroida Scrum ja Kanban siten, että valitaan Scrumista sellaisia piirteitä, jotka puuttuvat Kanbanista, mutteivät ole ristiriidassa Kanbanin perusperiaatteiden kanssa. Tällaisia ovat esimerkiksi jotkut Scrumin sisältämät kokoukset ja roolit. Sen sijaan aikarajoitettua (time-boxed) iteraatiota sprintin muodossa ei oteta mukaan. Myös tässä tapauksessa XP:stä voidaan valita teknisempiä käytänteitä.

K2: Millä tavalla yleisestä menetelmäkehityksestä ja -räätälöinnistä esitetyt ajatukset ovat nähtävissä ketterien menetelmien räätälöinnille an-netuissa jäsennyksissä ja ohjeissa?

Luvussa 2 määriteltiin, että menetelmäkehitys tarkoittaa kaikkia niitä aktiviteetteja, joiden avulla menetelmä on kehitetty ja myöhemmin räätälöity vastaamaan organisaation ja/tai projektin tarpeita. Menetelmän räätälöinti on tällöin organisaation menetelmän johtamista yleisestä menetelmästä tai projektin menetelmän johtamista organisaation menetelmästä. Ketterien menetelmien räätälöinnille on esitetty samantapaisia jäsennyksiä. Patelin ym. (2004) viitekehyksessä konteksti tiettyine tilannetekijöineen ajaa menetelmän räätälöinnin prosessia, josta tuloksena muodostuu käytettävä ketterä menetelmä. Kun räätälöityä menetelmää on sovellettu käytännön työssä, siitä voidaan ottaa kokemukset haltuun, ja käyttää niitä myöhemmin, kun menetelmää räätälöidään erilaiseen kontekstiin. Ketterien menetelmien räätälöintiä koskevissa tutkimuksissa menetelmän räätälöinnin prosessi, mikäli sellainen oli esitetty, pääasiallisesti noudatti Patelin ym. (2004) viitekehystä keskeisiltä osiltaan. Tutkimuksissa oli pääasiassa otettu jollain tapaa huomioon menetelmän räätälöinnistä saatavan kokemuksen haltuunotto, ja vaikka jotkin tutkimukset käsittelivät ketterän menetelmän räätälöintiä tietyn projektin käyttöön, on tutkimuksissa huomioitu se, että tietoa tulee säilöä myös organisaatiotasolla projektien välillä. Esimerkkinä tästä on Karlssonin ja Ågerfalkin (2009b) esittelemä organisaatiotasolla menetelmäosia keräävä tietokanta.

Kirjallisuushakuun perustuen tutkielmaan valittiin yhdeksän esitystä, jotka käsittelivät ketterän menetelmän räätälöintiä yleisesti. Käsiteltyjä ketterän menetelmän räätälöintiä koskevia ehdotuksia ja ohjeita oli hyvin erilaisia. Niiden lähestymistavat vaihtelivat, eivätkä läheskään kaikki niistä esittäneet prosessia, jolla räätälöintiä voi suorittaa. Tästä huolimatta niistä voi saada myös tietoa, jota voidaan käyttää hyödyksi ketterien menetelmien räätälöimisessä. Kypsyysmalliajattelun käyttämistä voi soveltaa esimerkiksi huomioimalla sen, että räätälöintiä voi soveltaa vaiheittain. Tällöin voi aloittaa pienin askelin ja

siirtyä kattavampaan sekä monimutkaisempaan käyttöön myöhemmin. Joidenkin tarkasteltujen käsitteellis-teoreettisten tutkimuksien arvo käytännön työlle on kuitenkin kyseenalainen, sillä tuloksia ei ole välttämättä sovellettu monissa projekteissa.

Keskeinen ajatus, joka näkyi yleisesti menetelmäkehitys ja -räätälöintitutkimuksissa ja ketterän menetelmän räätälöintiä koskevissa tutkimuksissa, oli komponenttiajatus. Tämän mukaan menetelmä nähdään koostuvan komponenteista ja elementeistä, joihin räätälöinnin aikana tehdään valintoja ja muutoksia. Samantapainen ajatus löytyy esimerkiksi Ayedin ym. (2012) metamallissa, Mikulenaan ja Kapociuksen (2011) priorisointiajattelussa, Karlssonin ja Ågerfalkin (2009b) menetelmäosa-ajattelussa sekä Luin ja Chanin (2005) XP:n käytäntöjen valintajärjestyksessä.

K3: Millä tavalla ketterien menetelmien räätälöinnille esitettyjä jäsennyksiä ja yleisiä ohjeita on noudatettu tapaustutkimuksen kohteena olleissa tapauksissa?

Luvussa 5 käsitellyt tapaustutkimukset valittiin käsiteltäväksi pyrkien siihen, että ne olivat keskenään erilaisia, jolloin räätälöidyt ketterät menetelmät ja tutkimuksien kontekstit vaihtelivat. Tapaustutkimuksissa noudatettiin harvoin mitään ennalta määriteltyä prosessia, tai se ainakin oli jätetty tutkimuksessa kertomatta. Räätälöinti näytti tapahtuvan usein implisiittisesti tilannetekijöiden mukaisesti, ja sitä, kuka tai ketkä räätälöintiä suorittivat, ei aina kerrottu. Implisiittisyys räätälöinnissä näkyi muun muassa siinä, että XP:tä räätälöitiin usein ottamalla tilannetekijöiden mukaisesti joitain sen kahdestatoista käytännöstä käyttöön, mutta sen rooleja ja prosesseja ei käytetty. XP:n roolien ja prosessin sijaan projektinhallintaan ja rooleihin käytettiin usein Scrumia. Tutkimuksissa ei kerrottu, harkittiinko näitä XP:n osia käyttöön vai jätettiinkö ne harkitusti pois. Kuitenkin laajan tietämyksen omaaminen eri menetelmistä ja menetelmäosista auttaa räätälöinnissä, ja siitä on myös hyötyä itse kehittäjille silloin, kun menetelmä on käytössä. Tämän ovat huomanneet muun muassa Fitzgerald ja Conboy (2010). Räätälöintiä käsittelevien tapaustutkimuksissa olisi hyvä käydä läpi sitä, harkittiinko muita menetelmiä, ja se, minkä vuoksi siitä on jätetty osia pois.

Yleinen tilanne ketterän menetelmän räätälöimiseksi oli se, että aiempi menetelmä (esim. vesiputousmalli) ei enää soveltunut käytettäväksi muuttuneessa liiketoimintaympäristössä ja ketterien menetelmien käytöllä haettiin mm. parempaa tuottavuutta. Joissain tutkimuksissa (esim. Maassen & Sonnevelt, 2010) oli aiemmin kokeiltu jo jotain ketterää menetelmää, mutta havaittu se puutteelliseksi.

Yleinen lähestymistapa räätälöintiin oli tapaustutkimuksissa käytäntö, jossa jonkin menetelmän käytänteet käydään läpi yksi kerrallaan ja otetaan se tai jätetään ottamatta käyttöön, riippuen sen sopivuudesta. Esimerkiksi joitain osia pystyttiin karsimaan pois sen takia, etteivät ne soveltuneet käytettäväksi joidenkin tilannetekijöiden, kuten sovellusalueen, takia. Tällaisia tutkimuksia olivat muun muassa Fitzgeraldin ym. (2006) ja Wangin ym. (2012) esittämät ta-

paustutkimukset. Menetelmien osat voidaan tällöin nähdä menetelmäkomponentteina. Lähestymistapa pitää sisällään samoja periaatteita kuin esimerkiksi Mikulenaksen ja Kapociuksen (2011) sekä Karlssonin ja Ågerfalkin (2009b) esityksissä. Mikulenas ja Kapocius (2011) esityksessä menetelmäkomponentit valittiin vertailemalla niiden hyötyjä ja kustannuksia toisiinsa. Tapaustutkimuksissa käytetyt lähestymistavat eivät kuitenkaan olleet yhtä formaaleja kuin mitä Mikulenas ja Kapocius (2011) ovat esittäneet. Karlssonin ja Ågerfalkin (2009b) menetelmäkomponenttiajattelussa oli huomioitu pohjamenetelmänä käytettävän menetelmän arvojen pitäminen mukana räätälöidyssä menetelmässä. Tapaustutkimuksissa tätä aihetta ei käsitelty. Myös Ayedin ym. (2011) tutkimus käsitteli lähestymistapaa ketterien menetelmien räätälöintiin ja ketterien menetelmien räätälöinnin metamallia menetelmäkomponenttien avulla.

Vaikka pilotointi oli käytössä useammassa tapaustutkimuksessa, ei sitä erityisesti käsitelty ketterän menetelmän räätälöintiä koskevissa käsitteellisteoreettisissa tutkimuksissa. Joissain tutkimuksissa (esim. Conboy & Fitzgerald, 2010; Karlsson & Ågerfalk, 2009b) otettiin huomioon projektien välillä tapahtuva kokemuksien haltuunotto, mutta pilotointia ei mainittu keinona kokemuksen kartuttamiseen.

XP:n räätälöintiä koskevissa tapaustutkimuksissa lähestymistapana oli valita olosuhteisiin sopivat käytänteet ja jättää sopimattomat käyttämättä. Fitzgeraldin ym. (2006) sekä Wangin ym. (2012) tutkimuksissa mainittiin myös pois jätetyt käytänteet, mutta Scottin ym. (2008) tutkimuksessa ei mainittu perusteita, joilla käytännöt valittiin tai jätettiin valitsematta. Käytänteiden valitseminen tällä keinolla ei noudattanut Luin ja Chanin (2005) esitystä, jossa käytänteitä otettiin osissa käyttöön. Yksi syy joidenkin käytäntöjen poisjättämiseen oli se, ettei jokin käytäntö soveltunut käyttöön esimerkiksi sovellusalueen vuoksi. Toinen syy voi olla se, että Lui ja Chan (2005) esittelivät tiekartta-mallinsa erityisesti kokemattomille kehitystiimeille, mutta Fitzgeraldin ym. (2006) ja Wangin ym. (2012) tapaustutkimuksissa ei ollut kyse kokemattomista tiimeistä. Mirakhorlin ym. (2008) esittelemän XP:n sääntöihin perustuvan räätälöintiprosessin ideoita ei tapaustutkimuksissa ollut havaittavissa käytössä.

Aydinin ym. (2005) jako staattiseen ja dynaamiseen menetelmän räätälöimiseen pitää sisällään piirteitä ketterän kehittämisen pyrkimyksestä jatkuvaan parantamiseen. Dynaaminen räätälöinti tapahtuu itse projektin aikana, jolloin prosessiin ja menetelmään tehdään muutoksia ja parannuksia tarvittaessa. Iteratiivista prosessimallia soveltavissa ketterissä menetelmissä on tyypillistä pohtia prosessia ja työtapoja sekä niiden parantamista itse projektin aikana, kuten esimerkiksi Scrumissa sprintin retrospektiivissä. Myös Kanbanissa jatkuva parantaminen ja parannusmahdollisuuksien havainnointi on keskeinen periaate. Qumer ja Henderson-Sellers (2008) esittivät vaiheittaisen käyttöönoton, jossa organisaatio tai tiimi arvioi ja parantaa omaa tekemistä vaiheittain. Tapaustutkimuksissa ei kuitenkaan käytetty tiekarttamaista lähestymistapaa menetelmän räätälöintiin ja käyttöönottoon.

Conboyn ja Fitzgeraldin (2010) ohjeiden takana olevat periaatteet olivat osittain nähtävissä käytännössä. Esimerkiksi ohjeet kehittäjien mukaan ottami-

sesta räätälöintiin ja heidän kouluttamisestaan oli havaittavissa esimerkiksi Scottin ym. (2008) sekä Maassenin ja Sonneveltin (2010) tutkimuksissa. Toisaalta osaa ohjeista, kuten formaalia analyysiä menetelmän sopivuudesta projektiympäristöön tai projektin aikana sen seuraamista, kuinka hyvin kehittäjät pysyvät käytänteiden käytössä, ei tapaustutkimuksissa tullut esille.

Caon ym. (2009) tutkimus käsitteli aihetta tutkimalla XP:n käyttöä AST-teorian valossa tietyssä organisaatiossa. Heidän antamiansa ohjeiden tyylisiä tekijöitä ei tapaustutkimuksissa ollut huomioitu tai sitä ei ollut mainittu, mutta niiden huomioiminen voi olla hyödyllistä. Esimerkiksi Wangin ym. (2012) ja Scottin ym. (2008) tutkimuksissa oli mainittu, että aloite ketterän menetelmän käyttöön tuli ylemmältä johdolta, jolle Caon ym. (2009) tutkimus tarjoaisi konkreettisia ohjeita siitä, miten toimia ketterien menetelmien käyttöönottamisessa ja räätälöimisessä.

Yleisiä johtopäätöksiä

Yleisesti tutkimuksista on havaittavissa useita ketterän menetelmän räätälöintiä koskevia ongelmia. Keskeinen ongelma on tilannetekijöiden tiedostaminen ja niihin reagoiminen oikealla tavalla. Tilannetekijät mainittiin räätälöintiä koskevissa tutkimuksissa miltei poikkeuksetta, mutta räätälöintipäätöksiä ja -ohjeita ei aina perusteltu niiden mukaisesti. Jonkin verran yksittäisiä ohjeita erilaisiin tilannetekijöihin on esitelty (vrt. Luku 5). Kaikkiin tilanteisiin sopivia johtopäätöksiä räätälöinnistä on luonnollisesti tilannetekijöiden kattavuuden takia mahdoton esittää. Yleisesti menetelmätietämyksen kasvattamisen lisäksi kannattaa kiinnittää huomiota myös tilannetekijöiden tiedostamiseen. Siten erityispiirteistä ja sovellusalueesta riippuen räätälöintiä osataan paremmin kohdentaa tiettyihin käytäntöihin, rooleihin tai prosesseihin. Ympäröivän maailman muuttuessa voi tulla esille myös tilannetekijöitä, joiden painoarvo aikaisemmin ei ole ollut niin merkittävä. Näitä voivat olla esimerkiksi muutokset käytettävissä teknologioissa tai projektien aikatauluissa.

Keskeinen tekijä on myös riittävän kokemuksen hankkimisen tarve, joka tuli sekä räätälöintiä koskevissa ehdotuksissa että tapaustutkimuksissa esille. Vaikka menetelmää räätälöivien henkilöiden tietämys menetelmistä olisi hyvällä tasolla, ei kehitystiimien tai organisaation tietämys käytetystä menetelmästä tai ketteristä menetelmistä yleisesti ollut aina riittävä. Myös se, miten räätälöity menetelmä tulisi esitellä ja kouluttaa itse kehittäjille, jotka joutuvat käyttämään sitä käytännön työssä, oli paikoin ongelmana. Ongelmaan ei ole yksiselitteistä vastausta, ja se täytyy huomioida eri konteksteissa erilaisilla tavoilla. Kehittäjien kokemus, taidot, yhtenäisyys, organisaatiorakenne ym. voivat olla tilannetekijöitä, jotka vaikuttavat sidosryhmien huomioimiseen. Esimerkiksi kokemattomien kehittäjien kanssa voi ottaa menetelmän vaiheittain käyttöön. Conboy ja Fitzgerald (2010) tarjoavat kehittäjien huomiointiin erilaisia ohjeita, kuten kehittäjien ottaminen mukaan räätälöintiin jo aikaisessa vaiheessa. Kehittäjille tulee myös perustella menetelmän käyttö. Kehittäjien lisäksi tulee huomioida myös muut sidosryhmät, kuten organisaation ylin johto.

Keskeinen tarve näyttäisi olevan ketterän menetelmän räätälöintiä käsitteleville tutkimuksille, jotka tarjoavat kehittäjille konkreettista apua räätälöinnin suorittamiseksi. Mikäli tutkimus esittelee monimutkaisen ja pitkäkestoisen prosessin, jolla räätälöintiä voi suorittaa, eikä se esitä tuloksia sen käyttämisestä käytännössä tai ole muulla tapaa vakuuttava, voivat menetelmäkehittäjät jättää ne huomioimatta. Useat organisaatiot ovat siirtyneet käyttämään ketteriä menetelmiä tuottavuuden ja tehokkuuden parantamisen toivossa, eikä räätälöinnille välttämättä jää paljoa aikaa tai muita resursseja.

7 YHTEENVETO

Ketterien menetelmien käyttö on kasvanut 2000-luvun alun jälkeen voimakkaasti. Yhä suurempi osa järjestelmäkehityksestä tehdään nykyään jonkin ketterän menetelmän mukaisesti. Kuten aiemmin perinteisten menetelmien yhteydessä, myös ketteriä menetelmiä tulee räätälöidä kulloisenkin tilanteen vaatimusten mukaisesti. Ketterien menetelmien räätälöinnistä on tehty viime vuosina sekä käsitteellistä että empiiristä tutkimusta. Kokonaisvaltaista selvitystä siitä, mitä räätälöinnillä ketterien menetelmien yhteydessä tarkoitetaan ja miten sitä on tehty käytännössä, ei sen sijaan ole olemassa.

Tämän tutkielman tarkoituksena on ollut etsiä vastaukset seuraaviin tutkimusongelmiin:

- Millä perusteilla ja tavoilla ketteriä menetelmiä on esitetty räätälöitäväksi?
- Millaisiin kehittämistilanteisiin ketteriä menetelmiä on räätälöity, miten sitä on tehty ja millaisia kokemuksia räätälöinnistä on saatu?

Tässä tarkoituksessa selvitettiin ensin luvussa 2, mitä menetelmällä tietojärjestelmien kehittämisen yhteydessä tarkoitetaan, mistä se koostuu, millä tavalla menetelmiä kehitetään ja erityisesti miten menetelmiä voidaan räätälöidä erilaisten tilannetekijöiden mukaan. Räätälöinnillä tarkoitetaan organisaatio- tai projektikohtaisen menetelmän johtamista yleensä jostain pohjamenetelmästä, jolloin räätälöity menetelmä on muokattu vastaamaan organisaation tai projektin erityispiirteitä eli tilannetekijöitä. Ohjelmistokehitykseen vaikuttavat tilannetekijät voivat liittyä henkilöstöön, vaatimuksiin, sovellukseen, teknologiaan, organisaatioon, toimintaan, johtamiseen ja liiketoimintaan.

Toiseksi luvussa 3 esiteltiin ketterän ohjelmistokehityksen manifesti sekä kirjallisuudessa esitettyjä määritelmiä ja käsityksiä ketteryydestä. Siinä yhteydessä todettiin, että käsitykset poikkeavat yksityiskohdissa toisistaan, mutta yleisesti ollaan sitä mieltä, että ketteryyteen liittyvät sellaiset ominaispiirteet kuten joustavuus, keveys ja reagoivuus. Luvussa kuvattiin myös kolmea tunnetuinta ketterää menetelmää, Scrumia, XP:tä ja Kanbania. Kahden ensimmäisen

osalta esiteltiin niiden prosesseja, rooleja ja käytänteitä, ja viimeksi mainitusta periaatteita. Scrum ja XP noudattavat iteratiivista prosessimallia, kun taas Kanbanissa ei ole erillistä prosessimallia. Myös roolit ja käytänteet eroavat menetelmissä toisistaan.

Ketterien menetelmien räätälöintiä koskevien tutkimusten löytämiseksi suoritettiin laaja kirjallisuushaku. Valitut tutkimukset jaettiin kahteen osaan, niihin joissa esitellään ketterän menetelmän räätälöintiä koskevia jäsennyksiä ja ohjeita ja niihin joissa raportoidaan tapaustutkimuksista. Edellisiä tutkimuksia käsiteltiin luvussa 4. Näitä tutkimuksia oli yhteensä yhdeksän kappaletta. Tutkimuksista käytiin läpi tarkemmin räätälöintistrategia ja -lähtökohta, räätälöintiprosessi sekä räätälöinnin kohde, eli mihin osiin menetelmiä räätälöinti kohdistuu. Tutkimuksia tarkastelemalla pyrittiin vastaamaan ensimmäiseen tutkimuskysymykseen. Ketterien menetelmien räätälöinnin perusteita ja tapoja strategioiden ja lähestymistapojen muodossa on esitetty useita. Jotkut tutkimukset ovat esittäneet selvän tavan räätälöidä joku tietty ketterä menetelmä. Niissä on esitetty prosessi, jonka avulla menetelmää voidaan räätälöidä. Tätä prosessia on voitu tukea esimerkiksi esittelemällä rooleja ja käytänteitä. Prosessin ja lähestymistavan toimivuutta ei ole kuitenkaan aina testattu käytännössä, tai se on ollut puutteelliselta. Osa tutkimuksista käsittelee aihetta tarjoamatta prosessia räätälöintiin, vaan tutkii aihetta esimerkiksi menetelmän metamallin näkökulmasta tai esittelemällä vaiheittaisen ketterän menetelmän käyttöönoton. Tutkimukset tarjoavat lukuisia ohjeita ketterän menetelmään räätälöintiin, joista voi olla hyötyä ketterää menetelmää räätälöidessä.

Luvussa 5 käsiteltiin ketterän menetelmän räätälöintiä koskevia tapaustutkimuksia. Tapaustutkimuksien käsittely tapahtui viitekehyksen avulla, joka koostuu kehittämisympäristön kontekstista ja sen erityispiirteistä, räätälöintiprosessista ja -strategiasta, räätälöinnin tuloksesta ja kokemuksien haltuunottamisesta. Käsiteltyjä tutkimuksia oli yhteensä kuusi kappaletta. Räätälöinnin kohteena on ollut Scrum, XP, Kanban, tai näiden yhdistelmä. Jokaisessa tutkimuksessa on ollut oma konteksti tietyin ominaispiirtein, joiden suhteen räätälöintiä on suoritettu. Ominaispiirteissä oli tutkimusten välillä eroja muun muassa organisaation koossa, käytetyissä teknologioissa, kehittäjien ominaisuuksissa ja liiketoiminnan luonteessa. Useimmissa tapauksissa ketteriä menetelmiä räätälöitiin organisaation käyttöön, mutta joissain tapauksissa tämä tehtiin räätälöimällä menetelmä suoraan pilottiprojektina tietyn projektin käyttöön, josta saatua tietämystä hyödynnettiin myöhemmin. Kaikki tapaustutkimukset eivät kuitenkaan kerro räätälöintipäätösten taustoja tarpeeksi hyvin, jonka johdosta niiden taustalla oleva rationaliteetti voi jäädä osittain epäselväksi.

Luvussa 6 pohdittiin aiemmin käsiteltyjen asiakokonaisuuksien välisiä yhteyksiä. Pohdinnan tuloksena voidaan todeta, että tapaustutkimuksissa on harvoin noudatettu mitään ennalta määritettyä prosessia ketterän menetelmän räätälöinnissä, vaan räätälöinti on edennyt usein tilannekohtaisesti muotoutunutta toimintatapaa käyttäen. Tapaustutkimuksissa ei myöskään viitattu ketterien menetelmien räätälöinnistä tehtyihin käsitteellis-teoreettisiin tutkimuksiin.

Niissä oli kuitenkin nähtävissä samoja periaatteita kuin aiemmin luvussa 4 esitetyissä käsitteellisteoreettisissa tutkimuksissa.

Ketterän menetelmän räätälöintiä koskevia tutkimuksia on varsin paljon. Tässä tutkielmassa ei ole pystytty niitä kaikkia käsittelemään. Vaikka käsittelyyn otettujen tutkimusten valinnassa pyrittiin noudattamaan edustavuuteen ja kattavuuteen pohjautuvia kriteerejä, huomiotta on voinut jäädä tutkimuksia, jotka sisältäisivät uudenlaisia ajatuksia ja kokemuksia ketterien menetelmien räätälöinnistä. Erityisesti tapaustutkimuksia olisi ollut hyvä käydä lisää sekä tarkemmin lävitse, mutta tätä ei voitu tämän työn rajoissa toteuttaa. Laajempi ja järjestelmällinen tapaustutkimusten läpikäynti voi olla yksi jatkotutkimusaihe. Jatkotutkimukseksi esitetään myös ketterien menetelmien räätälöinnin tutkimista empiirisesti, joko kyselytutkimuksella tai tapaustutkimuksella. Vastaavaa tutkimusta ei ole tehty Suomessa.

Käytänteiden räätälöintiä käsitteleviä tutkimuksia voisi lisäksi tehdä prosessi–innovaatioiden näkökulmasta katsoen, jolloin käytettävien käytänteiden, prosessien ja roolien lisäksi tutkitaan, kuinka syvällisesti organisaatiot ovat omaksuneet nämä käyttöönsä, ja tutkitaan keinoja käytänteiden omaksumiseen. Kypsyysmalliajattelun soveltaminen organisaatiossa ketterien menetelmien käytössä ja räätälöinnissä on myös tutkimisen arvoinen asia, ja sitä on osittain sivuttu myös tässä tutkielmassa. Tässä tutkielmassa on käsitelty ketterän menetelmän räätälöintiä Scrumin, XP:n ja Kanbanin avulla. Ketteriä menetelmiä on kuitenkin käytössä muitakin, kuten DSDM, Crystal ja LSD (Lean). Kyseisten menetelmien räätälöinti voisi olla myös yksi jatkotutkimusaihe.

LÄHTEET

Abrahamsson, P., Conboy, K. & Wong, X. (2009). ´Lots Done, More To Do´: The Current State of Agile Systems Development Research. *European Journal of Information Systems*, 18, 281-284.

Abrahamsson, P., Salo S., Ronkainen J. & Warsta J. (2002). *Agile software development methods – Review and analysis*. (1. painos). Espoo: VTT Publications.

Abrahamsson, P., Warsta, J., Siponen, M. & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. Teoksessa L. A. Clarke, L. Dillon & W. F. Tichy (toim.), *Proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, USA, May 3-10 (*s. 244-254). Washington, DC, USA: IEEE Computer Society.

Agile–allianssi. (2001a). Manifesto for Agile Software Development. Haettu 29.12.2013 osoitteesta http://www.agilemanifesto.org/

Agile–allianssi. (2001b). Principles behind the Agile Manifesto. Haettu 12.1.2014 osoitteesta http://www.agilemanifesto.org/principles.html

Agile–allianssi. (2001c). Julistuksen takana olevat periaatteet. Haettu 6.8.2014 osoitteesta http://www.agilemanifesto.org/iso/fi/principles.html

Akbar, R., Hassan, M. F. & Abdullah, A. (2011). A Review of Prominent Work on Agile Processes Software Process Improvement and Process Tailoring Practices. Teoksessa J. M. Zain, W. M. W. Mohd & E. El-Qawasmeh (toim.), *Software Engineering and Computer Systems: Second International Conference (ICSECS 2011), Kuantan, Pahang, Malaysia, June 27-29* (s. 571-585). Berlin: Springer-Verlag.

Ambler, S. W. 2002. *Agile Modeling: Best Practices for the Unified Process and Extreme Programming*. (1. painos). United States of America: John Wiley & Sons.

Avison, D. & Fitzgerald, G. (2003). Where Now for Development Methodologies. *Communications of the ACM,* 46(1), 79-82.

Avison, D. & Fitzgerald, G. (2006). *Information Systems Development Methodologies, Techniques and Tools*. (4. painos). Mateu Cromo: McGraw-Hill.

Ayed, H., Vanderose, B. & Habra, N. (2012). A Metamodel-Based Approach for Customizing and Assessing Agile Methods. Teoksessa J. P. Faria, A. Silva & R. J. Machado (toim.), *Proceedings of the 8th International Conference on the Quality of Information and Communications Technology (QUATIC 2012), Lisbon, Portugal, September 2-6* (s. 66-74). Los Alamitos, CA: IEEE Computer Society.

Aydin, M., Harmsen, F., van Slooten, K. & Stagwee, R. A. (2005). On the Adaptation of an Agile Information Systems Development Method. *Journal of Database Management*, 16(4), 24-40.

Backlund P., Hallenborg C & Hallgrimsson G. (2003). Transfer of Development Process Knowledge Through Method Adaptation. Teoksessa C. U. Ciborra, R. Mercurio, M. de Marco, M. Martinez & A. Carignani (toim.), *Proceedings of the Eleventh European Conference on Information Systems, Naples, Italy, June 16–21* (s. 122-125). Reading: Academic Conferences Limited.

Baskerville, R & Stage, J. (2001). Accommodating Emergent Work Practices: Ethnographic Choice of Method Fragments. Teoksessa N. Russo, B. Fitzgerald & J. Degross (toim.), *Realigning Research and Practice in Information Systems Development 2001, Boise Idaho, USA, July 27–29, 2001* (s. 11–28). New York: Springer Science+Business Media.

Baskerville, R & Pries-Heje, J. (2013). Discursive Co-development of Agile Systems and Agile Methods. Teoksessa Y. K. Dwivedi, H. Z. Henriksen, D. Wastell & R. De' (toim.), *Proceedings of IFIP WG 8.6 International Working Conference on Transfer and Diffusion of IT, TDIT 2013, Bangalore, India, June 27-29* (s. 279-294). Berlin: Springer-Verlag.

Bass, J. M. (2012). Influences on Agile Practice Tailoring in Enterprise Software Development. Teoksessa J. E. Guerrero (toim.), *Agile India Conference (AGILE 2012), Bangalore, India, February 17–19, 2012* (s. 1-10).

Bass, J. M. (2013). Agile Method Tailoring in Distributed Enterprises: Product Owner Teams. Teoksessa L. O'Connor (toim.), *Proceedings of the IEEE 8th International Conference on Global Software Engineering (ICGSE 2013), Bari, Italy, August 26-29* (s. 154-163). Los Alamitos, CA: IEEE Computer Society.

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. (1. painos). Reading, Massachusetts: Addison–Wesley.

Beck, K. & Andres, C. (2004). *Extreme Programming Explained: Embrace Change*. (2. painos). Reading, Massachusetts: Addison–Wesley.

Beck, K. & Boehm, K. (2003). Agility Through a Discipline: A Debate. *Computer*, 36(6), 44-46.

Bekkers, W., van de Weerd, I., Brinkkemper, S. & Mahieu, A. (2008). The Influence of Situational Factors in Software Product Management: An Empirical Study. Teoksessa C. Ebert, S. Brinkkemper, S. Jansen & G. Heller (toim.), *2nd International Workshop on Software Product Management (ISWPM'08), Barcelona, Spain, September 9* (s. 41-48). Los Alamitos, CA: IEEE Computer Society.

Benediktsson, O., Dalcher, D. & Thorbergsson, H. (2006). Comparison Of Software Development Life Cycles: A Multiproject Experiment. *IEEE Proceedings – Software*, 153 (3), 87-101.

Boehm, B. (2002). Get Ready for Agile Methods, with Care. *Computer*, 35(1), 64–69.

Boehm, B. & Turner, R. (2003). Using Risk to Balance Agile and Plan-driven Methods. *Computer*, 36(6), 57-66.

Booch, G., Rumbaugh, J. & Jacobson, I. (1999). *The Unified Modeling Language User Guide.* (1. painos). Redwood City, California, USA: Addison Wesley Longman.

Bowers, J., May, J., Melander, E., Baarman, M. & Ayoob A. (2002). Tailoring XP for Large System Mission Critical Software Development. Teoksessa D.

Wells & Williams L. A. (toim.), *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods – XP/Agile Universe 2002, Chicago, USA, August 4-7* (s. 100–111). Germany: Springer-Verlag.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38, 275–280.

Brinkkemper, S., Saeki, M. & Harmsen, F. (1999). Meta-modelling Based Assembly Techniques For Situational Method Engineering. *Information Systems,* 24(3), 209-228.

Cao, L., Mohan, K., Xu, P. & Ramesh, B. (2009). A Framework for Adapting Agile Development Methodologies. *European Journal of Information Systems*, 18, 332-343.

Coad, P., de Luca, J & Lefebvre, E. (1999). *Java Modeling In Color With UML: Enterprise Components and Process*. (1. painos). Prentice Hall.

Cockburn, A. (2001). *Crystal Clear: A Human–Powered Software Development Methodology for Small Teams*. (1. painos). Addison–Wesley Professional.

Cockburn, A. (2002). *Agile Software Development*. (1. painos). Boston, Addison-Wesley.

Cockburn, A. & Highsmith, J. (2001). Agile Software Development: The People Factor. *Computer*, 34(11), 131-133.

Cohn, M. (2009). *Succeeding with Agile: Software Development Using Scrum*. (1. painos). Addison–Wesley Professional.

Chang, L-H, Tung-ching, L., & Sheng, W. (2002). The Study of Information System Development (ISD) Process from the Perspectives of Power Development Stage and Organizational Politics. Teoksessa R. H. Sprague Jr. (toim.), *Proceedings of the 35th Hawaii International Conference on System Sciences, Hawaii, USA, January 7-10* (s. 3324-3334). Los Alamitos, CA: IEEE Computer Society.

Clarke P. & O'Connor R. V. (2012). The situational factors that affect the software development process: towards a comprehensive reference framework. *Journal of Information Software and Technology*, 54(5), 433–447.

Conboy, K. (2009). Agility from first principles: reconstructing the concept of agility in information systems development. *Information Systems Research,* 20, 329–354.

Conboy, K. & Fitzgerald, B. (2010). Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Transactions on Software Engineering and Methodology,* 20(1), 1–30.

Dahlmann, J., Gregorio, D. & Modigliani, P. (2013). Systems Engineering Processes for Agile Software Development. Teoksessa *Proceedings of the 7th Annual IEEE International Systems Conference (SysCon 2013), Orlando, Florida, USA, April 15-18* (s. 351-355). Los Alamitos, CA: IEEE Computer Society.

Damiani, E., Colombo, A., Frati, F. & Bellettini, C. (2007). A Metamodel for Modeling and Measuring Scrum Development Process. Teoksessa G. Concas, E. Damiani, M. Scotto & G. Succi (toim.), *Agile Processes in Software*

Engineering and Extreme Programming: 8th International Conference, XP 2007, Como, Italy, June 18-22 (s. 74-83). Germany: Springer-Verlag.

DeSanctis, G. & Poole, M. S. (1994). Capturing the complexity in advanced technology use: adaptive structuration theory. *Organization Science,* 2(5), 121-147.

DeMarco, T. (1982). *Controlling Software Projects: Management Measurement and Estimation*. Englewood Cliffs, NJ: Prenctice–Hall.

Díaz, J., Pérez, J., Yagüe A. & Garbajosa J. (2011). Tailoring the Scrum Development Process to Address Agile Product Line Engineering. Teoksessa C. C. Muñoz & S. A. Places (toim.), *Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2011), A Coruña, Spain, September 5–8, 2011* (s. 457-470). A Coruña: University of A Coruña

Diebold, P., Lampasona, C. & Taibi, D. (2013). Moonlighting Scrum: An Agile Method for Distributed Teams with Part-Time Developers Working during Non-Overlapping Hours. Teoksessa L. Lavazza, R. Oberhauser, A. Martin, J. Hassine, M. Gebhart & M. Jäntti (toim.), *Proceedings of The Eighth International Conferenceon Software Engineering Advances (ICSEA 2013), Venice, Italy, October 27 - November 1* (s. 318-323). International Academy, Research and Industry Association.

Drobka, J., Noftz, D. & Raghu, R. (2004). Piloting XP on four mission-critical projects. *IEEE Software*, 21(6), 70-75.

Dybå, T. & Dingsøyr, T. (2008). Empirical Studies of Agile Software Development: A Systematic Review. *Information and Software Technology,* 50, 833-859.

El-Said, M., Hana, M. & Eldin A. S. (2009). Agile Tailoring Tool (ATT): A Project Specific Agile Method. Teoksessa *2009 IEEE International Advance Computing Conference (IACC 2009),* Patiala, India, March 6–7, 2009 (s. 1659-1663). Los Alamitos, CA: IEEE Computer Society.

Faraj, S. & Sambamurthy, V. (2006). Leadership of Information Systems Development Projects. *IEEE Transactions on Engineering Management, 53*(2), 238–249.

Fitzgerald, B. (1996). Formalised Systems Development Methodologies: A Critical Perspective. *The Information Systems Journal*, 6(1), 3-23.

Fitzgerald, B., Hartnett, G. & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*, 15, 200–213.

Fitzgerald, B., Russo, N. & O'Kane, T. (2003). Software Development Method Tailoring At Motorola. *Communications of the ACM*, 46(4), 64-70.

Fitzgerald, B., Russo, N. & Stolterman, E. (2002). *Information systems development: Methods–in–action.* (1. painos). Great Britain: McGraw–Hill Higher Education.

Fitzgerald, B., Stol, K-J., O'Sullivan, R. & O'Brien, D. (2013). Scaling Agile Methods to Regulated Environments: An Industry Case Study. Teoksessa D. Notkin, B. H. C. Chang & K. Pohl (toim.), *Proceedings of 35th International Conference on Software Engineering (ICSE 2013), San Francisco,*

California, USA, May 18-26 (s. 863-872). Passau, Germany: IEEE Computer Society.

Feiler, P. & Humphrey, W. (1992). *Software Process Development and Enactment: Concepts and Definitions* (raportti CMU/SEI-92-TR-004). Carnegie Mellon University, Software Engineering Institute.

Fruhling, A & De Vreede, G-J. (2006). Field Experiences with eXtreme Programming: Developing an Emergency Response System. *Journal of Management Information Systems,* 22(4), 39-68.

Gandomani, T. J., Zulzazil, H., Ghani, A. A. A. & Sultan, A. B. M. (2013). Towards Comprehensive and Disciplined Change Management Strategy in Agile Transformation Process. *Research Journal of Applied Sciences, Engineering and Technology,* 6(13), 2345-2351.

Greaves, K. (2011). Taming the Customer Support Queue: A Kanban Experience Report. Teoksessa R. Bilof (toim.), *Proceedings of Agile 2011 Conference, Salt Lake City, Utah, USA, August 8-12* (s. 154-160). Los Alamitos, CA: IEEE Computer Society.

Harmsen, A. F. (1996). *Situational Method Engineering.* Väitöskirja, Twenten yliopisto.

Harmsen, F., Brinkkemper, S. & Oei, H. (1994). Situational Method Engineering For Information System Project Approaches. Teoksessa A. A. Verrjin Stuart & T.W. Olle (toim.), *Methods and Associated Tools for the Information Systems Life Cycle. Proceedings of the IFIP WG 8.1 Working Conference, Maastricht, Netherlands, September 26-28* (s. 169-194). New York, USA: Elsevier Science Inc.

Harmsen, F., van den Brand, M., van Hillegersberg, J. & Aydin, M. N. (2007). Agile Methods for Offshore Information Systems Development. Teoksessa *Proceedings of the First Information Systems Workshop on Global Sourcing: Services, Knowledge and Innovation, Val d'Isere, France, March 13-15* (s. 1-20). Great Britain: Global Sourcing.

Henderson-Sellers, B. & Serour, M. K. (2005). Creating a Dual-Agility Method: The Value of Method Engineering. *Journal of Database Management,* 16(4), 1-23.

Henderson-Sellers, B. & Gonzalez-Perez, C. (2005). A comparison of four process metamodels and the creation of a new generic standard. *Information and Software Technology,* 47(1), 49-65.

Henderson-Sellers, B. & Ralyté, J. (2010). Situational Method Engineering: State-of-the-Art Review. *Journal of Universal Computer Science,* 16(3), 424-478.

Henninger, S., Ivaturi, A., Nuli, K & Thirunavukkaras A. (2002). Supporting Adaptable Methodologies to Meet Evolving Project Needs. Teoksessa D. Wells & Williams L. A. (toim.), *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods – XP/Agile Universe 2002, Chicago, USA, August 4-7* (s. 33–44). Germany: Springer-Verlag.

Heym, M & Österle, H. (1992). A Semantic Data Model for Methodology Engineering. Teoksessa G. Forte & N. Madhavji (toim.), *Proceedings of the Fifth International Workshop on Computer-Aided Software Engineering,*

Montreal, Canada, July 6-10 (s. 142-155). Los Alamitos, CA: IEEE Computer Society.

Highsmith, J. (2003). *Agile Project Management: Principles and Tools* (Julkaisusarjan osa 4, numero 2). Arlington, Massachusetts: Cutter Consortium.

Highsmith, J. (2002). *Agile Software Development Ecosystems*. United States of America: Addison-Wesley.

Highsmith, J. & Cockburn, A. (2001). Agile Software Development: The Business of Innovation. *Computer,* 34(9), 120-127.

Hildenbrand, T., Geisser, M., Kude, T., Bruch, D. & Acker T. (2008). Agile Methodologies for Distributed Collaborative Development of Enterprise Applications. Teoksessa F. Xhafa & L. Barolli (toim.), *Proceedings of The Second International Conference on Complex, Intelligent and Software Intensive Systems (CISIS 2008), Barcelona, Spain, March 4-7* (s. 540-545). Los Alamitos, CA: IEEE Computer Society.

Hong, N., Yoo, J & Sungdeok C. (2010). Customization of Scrum Methodology for Outsourced E–commerce Projects. Teoksessa J. Han & T. D. Thu (toim.), *Proceedings of 17th Asia Pacific Software Engineering Conference (APSEC 2010), Sydney, Australia, November 30 – December 3* (s. 310-316). Los Alamitos, CA: IEEE Computer Society.

Hossain, E., Bannerman, P. L. & Jeffery, R. (2011). Towards an Understanding of Tailoring Scrum in Global Software Development: A Multi–case Study. Teoksessa D. Raffo, D. Pfahl & L. Zhang (toim.), *Proceedings of the 2011 International Conference on Software and Systems Process (ICSSP '11), Honolulu, Hawaii, USA, May 21–22* (s. 110-119). New York, USA: ACM.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology*. IEEE Std 610.12-1990.

Iivari J., Hirschheim R. & Klein H.K. (1998). A Paradigmatic Analysis Contrasting Information Systems Development Approaches and Methodologies. *Information Systems Research*, 9(2), 164–193.

Ikonen, M., Pirinen, E., Fagerholm, F., Kettunen, P. & Abrahamsson, P. (2011). On the Impact of Kanban on Software Project Work: An Empirical Case Study Investigation. Teoksessa I. Perseil, K. Breitman & R. Sterritt (toim.), *Proceedings of the Sixteenth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2011), Las Vegas, Nevada, USA, April 27-29* (s. 305-314). Los Alamitos, CA: IEEE Computer Society.

Ingason, H. T., Gestsson, E. & Jonasson, H. I. (2013). The Project Kanban Wall: Combining Kanban and Scrum for Coordinating Software Projects. *PM World Journal,* 2(8), 1-23.

Jayawardena, D. S. & Ekanayake L. L. (2010). Adaptation Analysis of Agile Project Management for managing IT projects in Sri Lanka. *Teoksessa Proceedings of International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, September 29 – October 1, 2010* (s. 1-4). Los Alamitos, CA: IEEE Computer Society.

Jones, C. (2007). Development Practices for Small Software Applications. *Cross-talk, the Journal of Defense Software Engineering*, 21(2), 9-13.

Karlsson, F. & Ågerfalk, P. (2004). Method Configuration: Adapting to Situational Characteristics While Creating Reusable Assets. *Information and Software Technology*, 46(9), 619–633.

Karlsson, F. & Ågerfalk, P. (2005). Method-user-centred method configuration. Teoksessa J. Ralyté, P. Ågerfalk & N. Kraiem (toim.), *In Proceedings of the Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes (SREP'05), Paris, France, August 28-30* (s. 31-43). Los Alamitos, CA: IEEE Computer Society.

Karlsson, F. & Ågerfalk, P. (2009a). Towards Structured Flexibility in Information Systems Development: Devising a Method for Method Configuration. *Journal of Database Management,* 20(3), 51-75.

Karlsson, F. & Ågerfalk, P. (2009b). Exploring Agile Values in Method Configuration. *European Journal of Information Systems*, 18, 300-316.

Karlsson, J., Wohlin, C. & Regnell, B. (1998). An evaluation of methods for prioritizing software requirements. *Information and Software Technology,* 39(14-15), 939-947.

Keenan, F. Agile Process Tailoring and probLem analYsis (APTLY). (2004). Teoksessa *Proceedings of the 26th International Conference on Software Engineering (ICSE'04),* Edinburgh, United Kingdom, May 23–28, 2004 (s. 45-47). Washington, DC, USA: IEEE Computer Society.

Kniberg, H. & Skarin, M. (2010). *Kanban and Scrum - making the most of both*. C4Media, Publisher of InfoQ.com.

Koch, A. S. (2005). *Agile Software Development: Evaluating the Methods for Your Organization*. (1. painos). Artech House Publishers.

Koutonen, J. & Leppänen M. (2013). How Are Agile Methods and Practices Deployed in Video Game Development? A Survey into Finnish Game Studios. Teoksessa H. Baumeister & B. Weber (toim.), *Agile Processes in Software Engineering and Extreme Programming: Proceedings of 14th International Conference, XP 2013, Vienna, Austria, June 3-7* (s. 135-149). Berlin: Springer-Verlag.

Kumar, K. & Welke, R. J. (1992). Methodology Engineering: A Proposal for Situation-specific Methodology Construction. Teoksessa W. W. Cotterman & J. A. Senn (toim.), *Challenges and Strategies for Research in Systems Development* (s. 257-269). New York, USA: John Wiley & Sons.

Laanti, M., Similä, J. & Abrahamsson, P. Definitions of Agile Software Development and Agility. (2013). Teoksessa F. McCaffery, R. V. O'Connor & R. Messnarz (toim.), *Proceedings of 20th European Conference on Systems, Software and Service Process Improvement (EuroSPI), Dundalk, Ireland, June 25-27* (s. 247-258). Berlin: Springer-Verlag.

Ladas, C. (2008). *Scrumban: Essays on Kanban Systems for Lean Software Development*. Seattle, Washington: Modus Cooperandi Press.

Lagerberg, L., Skude, T., Emanuelsson, P, Sandahl, K. & Ståhl, D. (2013). The impact of agile principles and practices on large-scale software development projects. Teoksessa L. O´Conner (toim.), *Proceedings of the 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Meas-*

urement (ESEM 2013), October 10-11, Baltimore, Maryland, USA (s. 348-356). Los Alamitos, CA: IEEE Computer Society.

Langefors, B. (1973). *Theoretical Analysis of Information Systems.* (4. painos). Auerbach, Philadelphia.

Lauesen, S. (2002). *Software Requirements: Styles & Techniques*. (1. painos). Essex, Great Britain: Pearson Education.

Layman, L., Williams, L., Damian, D. & Bures, H. (2006). Essential Communication Practices for Extreme Programming in a Global Software Development Team. *Information Software and Technology,* 48(9), 781-794.

Layman, L., Williams, L. & Cunningham, L. (2004). Exploring Extreme Programming in Context: An Industrial Case Study. Teoksessa *Proceedings of the 2nd Agile Development Conference (ADC '04), Salt Lake City, Utah, USA, June 22-26* (s. 32-41). Los Alamitos, CA: IEEE Computer Society.

Leffingwell, D. & Widrig, D. (1999). *Managing Software Requirements: A Unified Approach*. (1. painos). Upper Saddle River, New Jersey, USA: Addison-Wesley.

Leppänen M. (2005). *An Ontological Framework and a Methodical Skeleton for Method Engineering – A Contextual Approach*. Tietojärjestelmätieteen väitöskirja. Jyväskylän yliopisto.

Li, J., Moe, N. B. & Dybå, T. (2010). Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality. Teoksessa G. Succi, M. Morisio & N. Nagappan (toim.), *Proceedings of the 4th International Symposium on Empirical Software Engineering and Measurement (ESEM 2010), September 16-17, Bolzano-Bozen, Italy* (s. 117-127). Los Alamitos, CA: IEEE Computer Society.

Little, J. & Karaj, A. (2013). Transforming a Public Sector Organization From Stone Age to Agile. Teoksessa J. E. Guerrero (toim.), *Proceedidngs of AGILE 2013, Nashville, Tennessee, USA, August 5-9* (s. 74-81). Los Alamitos, CA: IEEE Computer Society.

Lui, K. M. & Chan, K. C. C. (2005). A Road Map for Implementing eXtreme Programming. Teoksessa M. Li, B. Boehm & L. J. Osterweil (toim.), *Unifying the Software Process Spectrum: Proceedings of International Software Process Workshop, SPW 2005, Beijing, China, May 25-27* (s. 474-481). Berlin: Springer-Verlag.

Lyytinen, K. & Rose G. M. (2006). Information System Development Agility as Organizational Learning. *European Journal of Information Systems*, 15(2), 183-199.

Maassen, O. & Sonnevelt, J. (2010). Kanban at an Insurance Company (Are You Sure?). Teoksessa A. Sillitti, A. Martin, X. Wang & E. Whitworth (toim.), *Agile Processes in Software Engineering and Extreme Programming: Proceedings of 11th International Conference (XP 2010), Trondheim, Norway, June 1-4* (s. 297-306). Germany: Springer-Verlag.

MacCormack, A. & Verganti, R. (2003). Managing the Sources of Uncertainty: Matching Process and Context in Software Development. *The Journal of Product Innovation Management*, 20(3), 217-232.

McAvoy, J. & Butler, T. (2007). The Impact of the Abilene Paradox on Double-loop Learning in an Agile Team. *Information and Software Technology,* 49(6), 552-563.

Mikulėnas G., Butleris, R. & Nemuraitė, L. (2011). An Approach for the Metamodel of the Framework for a Partial Agile Method Adaptation. *Information Technology and Control,* 40(1), 71-82.

Mikulenas G., & Kapocius K. (2011). An Approach for Prioritizing Agile Practices for Adaptation. Teoksessa W. Wei Song, S. Xu, C. Wan, Y. Zhong, W. Wojtkowski, G. Wojtkowski, H. Linger (toim.), *Information Systems Development – Asian Experiences* (s. 485–499).

Mirakhorli, M., Rad, A. K., Aliee, F. S., Pazoki, M. & Mirakhorli, A. (2008). RDP Technique: a Practice to Customize XP. Teoksessa S. Adolph & P. Krichten (toim.), *Compilation E-Proceedings of the Thirteenth International Conference on Software Engineering: Co-located workshops, May 10-18, Leipzig, Germany* (s. 23-32). Los Alamitos, CA: IEEE Computer Society.

Mishra, D. & Mishra, A. (2011). Complex software project development: agile methods adoption. *Journal of Software Maintenance and Evolution: Research and Practice*, 23(8), 549-564.

Mnkandla, E. & Dwolatzky, B. (2007). Agile Methodologies Selection Toolbox. Teoksessa *Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007), August 25-31, Cap Esterel, French Riviera, France* (s. 72). International Academy, Research and Industry Association.

Mordinyi, R., Kühn, E. & Schatten, A. (2010). Towards an Architectural Framework for Agile Software Development. Teoksessa R. Steritt, B. Eames & J. Sprinkle (toim.), *Proceedings of 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, Oxford, England, March 22-26* (s. 276-280). Los Alamitos, CA: IEEE Computer Society.

Moreira, M. (2013). *Being Agile: Your Roadmap to Successful Adoption of Agile*. (1. painos). Apress.

Murru, O., Deias, R. & Mugheddu, G. (2003). Assessing XP at a European Internet company. *IEEE Software*, 20(3), 37-43.

Nerur, S., Mahapatra, R. & Mangalaraj, G. (2005). Challenges of Migrating to Agile Methodologies. *Communications of the ACM*, 48(5), 73-78.

Nielsen, J. & McMunn, D. (2004). The Agile Journey: Adopting XP in a Large Financial Services Organization. Teoksessa H. Baumeister, M. Marchesi & M. Holcombe (toim.), *Extreme Programming and Agile Processes in Software Engineering: Proceedings of 6th International Conference (XP 2005), Sheffield, United Kingdom, June 18-23* (s. 28-37). Germany: Springer-Verlag.

Nikitina, N., Kajko-Mattson, M. & Stråle, M. (2012). From Scrum to Scrumban: A Case Study of a Process Transition. Teoksessa R. Jeffery, D. Raffo, O. Armbrust & L. Huang (toim.), *Proceedings of 2012 International Conference on Software and System Process (ICSSP), June 2-3, Zürich, Switzerland* (s. 140-149). Passau, Germany: IEEE Computer Society.

OPEN Process Framework Repository Organization (OPFRO). (2009). Open Process Framework. Haettu 29.3.2014 osoitteesta http://www.opfro.org/

Object Management Group (OMG). (2008). Software & Systems Process Engineering Metamodel. Haettu 29.3.2014 osoitteesta http://www.omg.org/spec/SPEM/2.0/

Paasivaara, M., Durasiewicz, S. & Lassenius, C. (2008). Teoksessa S. Ceballos (toim.), *Proceedings of the International Conference on Global Software Engineering (ICGSE 2008), Bangalore, India, August 17-20* (s. 87-95). Los Alamitos, CA: IEEE Computer Society.

Palvia, P. & Nosek, J. (1993). A Field Examination of System Life Cycle Techniques and Methodologies. *Information & Management,* 25(2), 73-84.

Patel C., de Cesare S., Iacovelli N. & Merico A. (2004). A framework for method tailoring: a case study. Teoksessa M. Serour (toim.), *Proceedings of 2nd OOPSLA Workshop on Method Engineering for Object–Oriented and Component–Based Development, Vancouver, Canada, October 24-28* (s. 1-15). New York: ACM.

Pedreira O., Piattini M., Luaces M. & Brisaboa N. (2007). A Systematic Review of Software Process Tailoring. *ACM SIGSOFT Software Engineering Notes*, 32(3), 1–6.

Pikkarainen, M. & Passoja, U. (2005). An Approach for Assessing Suitability of Agile Solutions: A Case Study. Teoksessa H. Baumeister, M. Marchesi & M. Holcombe (toim), *Extreme Programming and Agile Processes in Software Engineering: 6th International Conference, XP 2005, Sheffield, United Kingdom, June 18-23* (s. 171-179). Germany: Springer-Verlag.

Polk, R. (2011). Agile & Kanban In Coordination. Teoksessa R. Bilof (toim.), *Proceedings of Agile 2011, Salt Lake City, Utah, USA, August 8-12* (s. 263-268).

Poole, M. S. & DeSanctis, G. (1990). Understanding the use of group decision support systems: the theory of adaptive structuration. Teoksessa J. Fulk & C. Steinfeld (toim.), *In Organizations and Communication Technology (s. 173-193).* Newbury Park, CA: Sage Publications.

Poole, C. & Huisman, J. W. (2001). Using Extreme Programming in a Maintenance Environment. *IEEE Software*, 18(6), 42-50.

Poppendieck, M. (2001). Lean programming. *Software Development Magazine*, 9, 71–75.

Qumer, A. & Henderson–Sellers, B. (2006a). Measuring agility and adoptability of agile methods: a 4–dimensional analytical tool. Teoksessa N. Guimares, P.Isaias A. Goikoetxea (toim.), *Proceedings of the IADIS International Conference on Applied Computing, San Sebastian, Spain, February 25-28* (s. 503–507). IADIS.

Qumer, A. & Henderson–Sellers, B. (2006b). Comparative evaluation of XP and Scrum using the 4D Analytical Tool (4-DAT). Teoksessa Z. Irani, O. D. Sarikas, J. Llopis, R. Gonzalez & J. Gasco (toim.), *Proceedings of the European and Mediterranean Conference on Information Systems (EMCIS), Costa Blanca, Alicente, Spain, July 6-7* (s. 1-8). London: Brunel University.

Qumer, A. & Henderson–Sellers, B. (2008). A framework to support the evaluation, adoption and improvement of agile methods in practice. *Journal of Systems and Software,* 81(11), 1899-1919.

Ramamoorthy, C. V., Garg, V. & Prakash, A. (1986). Programming in the Large. *IEEE Transactions on Software Engineering*, 12(7), 769-783.

Regnell, B., Höst, M., Natt och Dag, J., Beremark, P. & Hjelm, T. (2001). An industrial case study on distributed prioritization in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1), 51-62.

Roberts Jr., T, Gibson, M., Fields, K. & Kelly Rainer Jr., R. (1998). Factors that Impact Implementing a System Development Methodology. *IEEE Transactions On Software Engineering*, 24(8), 640- 649.

Rodriguez P., Markkula J., Oivo M., Turula K. 2012. Survey on agile and lean usage in Finnish software industry. Teoksessa *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM'12), Lund, Sweden, September 19-20* (s. 139-148). Los Alamitos, CA: IEEE Computer Society.

Rodriguez, P., Partanen, J., Kuvaja, P. & Oivo, M. (2014). Combining Lean Thinking and Agile Methods for Software Development: A Case Study of a Finnish Provider of Wireless Embedded Systems. Teoksessa R. H. Sprague Jr. (toim.), *Proceedings of the 47th Annual Hawaii International Conference on System Sciences, January 6-9, Waikoloa, Hawaii, USA* (s. 4770-4779). Los Alamitos, CA: IEEE Computer Society.

Rottier, P. A. & Rodrigues, V. (2008). Agile Development in a Medical Device Company. Teoksessa G. Melnik, P. Kruchten & M. Poppendieck (toim.), *Proceedings of the Agile 2008 Conference, August 4-8, Toronto, Canada* (s. 218-223). Los Alamitos, CA: IEEE Computer Society.

Royce, W. (1970). Managing the Development of Large Software Systems. *Proceedings of IEEE Wescon 26, August, Los Angeles, California, USA* (s. 1-9).

Saaty, T. L. (2007). Multi-decisions decision-making: In addition to wheeling and dealing, our national political bodies need a formal approach for prioritization. *Mathematical and Computer Modelling,* 46(7-8), 1001-1016.

Salo, O. & Abrahamsson, P. (2007). An Iterative Improvement Process for Agile Software Development. *Software Process Improvement and Practice*, 12, 81-100.

Salo, O & Abrahamsson, P. (2008). Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. *IET Software,* 2(1), 58–64.

Sauer, C. & Lau, C. (1997). Trying to adopt systems development methodologies – a case-based exploration of business users' interests. *Information Systems Journal,* 7, 255-275.

Schwaber, K. (1995). SCRUM Development Process. Teoksessa R. Wirfs-Brock (toim.), *Proceedings of the Tenth Annual Conference on Object-Oriented Programming Systems, Languages, and Applications, Austin, Texas, USA, October 15-19* (s. 1-23). New York, USA: ACM.

Schwaber, K. & Beedle, M. (2002). *Agile Software Development with Scrum*. (1. painos). Upper Saddle River, New Jersey, USA: Prentice Hall.

Schwaber, K. & Sutherland, J. (2013). *The Official Scrum Rulebook*. Haettu 30.12.2013 osoitteesta http://www.scrum.org/Scrum-Guides

Scott, J., Johnson R. & McCullough M. (2008). Executing Agile in a Structured Organization: Government. Teoksessa G. Melnik, P. Kruchten & Poppendieck, M (toim.), *Agile Conference (AGILE), 2008*, Toronto, Canada, August 4–8, 2008 (s. 166-170). Los Alamitos, CA: IEEE Computer Society.

Senapathi, M. & Srinavasan, A. (2013). Sustained Agile Usage: A Systematic Literature Review. Teoksessa A. Almeida & E. Barreiros (toim.), *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE 2013), Porto de Galinhas, Brazil, April 14-16 (*s. 119-124). New York, USA: ACM.

Schönström M. & Carlsson S. (2003). Methods as Knowledge Enablers in Software Development Organizations. Teoksessa C. U. Ciborra, R. Mercurio, M. de Marco, M. Martinez & A. Carignani (toim.), *Proceedings of the Eleventh European Conference on Information Systems, Naples, Italy, June 16–21* (s. 1707-1718). Reading: Academic Conferences Limited.

Shalloway, A., Beaver, G. & Trott, J. R. (2010). *Lean-Agile Software Development: Achieving Enterprise Agility.* Stoughton, Massachusetts, USA: Addison-Wesley Professional.

Stapleton, J. (1997). *DSDM Dynamic Systems Development Method: The Method in Practice*. (1. painos). Great Britain: Addison–Wesley.

Subramanian, G. H., Klein, G., Jiang, J. J. & Chan, C-L. (2009). Balancing Four Factors in System Development Projects. *Communications of the ACM*, 52(10), 118-121.

Terlecka, K. (2012). Combining Kanban and Scrum - Lessons from a Team of Sysadmins. Teoksessa R. Bilof (toim.), *Proceedings of AGILE 2012, Dallas, Texas, USA, August 13-17* (s. 99-103). Los Alamitos, CA: IEEE Computer Society.

Tolvanen J–P. (1998). *Incremental Method Engineering with Modeling Tools – Theoretical Principles and Empirical Evidence*. Tietojärjestelmätieteen väitöskirja. Jyväskylän yliopisto.

Turk, D., France, R. & Rumpe, B. (2002). Limitations of Agile Software Processes. Teoksessa *In Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering, Alghero, Italy, May 26-29* (s. 43-46).

Van Slooten, K. & Schoonhoven, B. (1996). Contingent Information Systems development. *Journal of Systems and Software*, 33(11), 1–9.

Van Slooten, C., Brinkkemper, S. & Hoving, P. (1994). Contingency Based Situational Systems Development in Large Organizations. Teoksessa M. Khosrowpour (toim.), *Managing Social and Economic Change With Information Technology, Proceedings of the 5th Information Resources Management Association International Conference, San Antonio, Texas, May 1994* (s. 267–275). Harrisburg: Idea Group Publishing.

Vanderose, B., Kamseu, F. & Habra, N. (2010). Towards a Model-Centric Quality Assessment. Teoksessa A. Abran, G. Buren, R. R. Dumke & J. J. Cuadrado-Gallego (toim.), *Applied Software Measurement: Proceedings of the Joined International Conferences on Software Measurement*

IWSM/Metrikon/Mensura 2010, Stuttgart, Germany, 10-12 November (s. 21-34). Germany: Shaker Verlag.

VersionOne. (2014). 8th Annual State of Agile Development Survey. Haettu 18.8.2014 osoitteesta http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

Wastell, D. G. (1996). The Fetish of Technique: Methodology as a Social Defense. *Information Systems Journal*, 6(1), 25-40.

Wang, X., Conboy K. & Pikkarainen M. (2012). Assimilation of agile practices in use. *Information Systems Journal*, 22(6), 435–455.

Wong, S–P. & Whitman, L. (1999). Attaining Agility At The Enterprise Level. Teoksessa *Proceedings of the 4th Annual International Conference on Industrial Engineering Theory, Applications and Practice. San Antonio, Texas, USA, 17–20 November* (s. 1–5). IADIS.

Webster. (1989). *Webster's Encyclopedic Unabridged Dictionary of the English Language*. New York: Gramercy Books.

Yeh, R. (1991). System development as a wicked problem. *International Journal of Software Engineering and Knowledge Engineering*, 1(2), 117-130.

Ågerfalk, P. & Fitzgerald, B. (2006). Flexible and Distributed Software Processes: Old Petunias in New Bowls? *Communications of the ACM*, 49(10), 27-34.

Özcan, T., Kocak, S. & Brune, P. (2013). Agile Software Development with Open Source Software in a Hospital Environment – Case Study of an eCRF-System for Orthopaedical Studies. Teoksessa F. Daniel, P. Dolog & Q. Li (toim.), *Proceedings of the 13th International Conference on Web Engineering (ICWE 2013), Aalborg, Denmark, July 8-12* (s. 439-451). Berlin: Springer-Verlag.