Matias Partanen

KETTERÄN MENETELMÄN RÄÄTÄLÖINTI

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2013

TIIVISTELMÄ

Partanen, Matias
Ketterän menetelmän räätälöinti
Jyväskylä: Jyväskylän yliopisto, 2013, 44 s.
Tietojärjestelmätiede, kandidaatin tutkielma
Ohjaaja: Leppänen, Mauri

Ohjelmistokehityksessä käytetään hyvin usein jotain kehittämismenetelmää. Ketterien menetelmien käyttäminen on lisääntynyt 2000–luvulla, ja niiden käyttämisestä on havaittu olevan hyötyä muun muassa muuttuviin vaatimuksiin vastattaessa. Organisaatiot ja projektit kuitenkin harvoin noudattavat ketteriä menetelmiä täysin kehittäjien tarkoittamalla tavalla, vaan räätälöivät ne vastaamaan kehittämiskontekstia ja sen erityispiirteitä. Tässä tutkielmissa tarkastellaan ketterien menetelmien räätälöintiä.

Tutkielmassa esitellään lyhyesti mitä kehittämismenetelmällä tarkoitetaan, millä tavoilla menetelmän kehittämistä ja räätälöintiä voidaan suorittaa sekä millaisia ohjelmistokehitykseen ja räätälöintiin vaikuttavia tilannetekijöitä on olemassa. Tämän jälkeen tutkielmassa esitetään määritelmiä ketteryydelle ja esitetään ketterän ohjelmistokehityksen arvot ja sekä kuvataan ketteristä menetelmistä tarkemmin Scrumia ja XP:tä prossessien, roolien ja käytänteiden näkökulmasta.

Tämän jälkeen tutkitaan millä tavoilla ketteriä menetelmiä on esitetty räätälöitäväksi ja mitä käytännön kokemuksia räätälöinnistä on saatu. Tutkimusta varten on valittu ja tutkittu tarkemmin viittä kirjallisuudessa esitettyä tapaustutkimusta, joissa on käsitelty joko Scrumin tai Scrumin ja XP:n yhdistelmän räätälöintiä. Tapaustutkimuksien käsittely tapahtuu ketterän menetelmän viitekehyksen avulla, joka koostuu kehittämisympäristön kontekstista ja sen erityispiirteistä, prosessista ja strategiasta sekä räätälöinnin tuloksesta. Tutkielman tulosten avulla voidaan nähdä, millä tavoin eri prosesseja, rooleja ja käytänteitä Scrumista ja XP:tä on erilaisissa tapauksissa räätälöity. Tuloksia voidaan hyödyntää ketterää menetelmää räätälöidessä organisaation tai projektin käyttöön.

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

ABSTRACT

Partanen, Matias
Tailoring of an Agile Method
Jyväskylä: University of Jyväskylä, 2013, 44 p.
Information systems science, bachelor's thesis
Supervisor: Leppänen, Mauri

It is common that information systems development and software engineering is based on some method. During the last decade the use of agile methods has increased a lot, due to their ability, for instance, to allow rapid changes in information requirements. However, organizations and projects rarely adopt and use the methods as the way their developers have intended, but rather tailor them to meet the contexts and their features. This thesis deals with tailoring of agile methods.

This thesis will first briefly go through what a method means, in which ways it can be engineered and tailored and what situational factors there are in software development. After that the concepts of agile and agile method will be discussed. Two agile methods, Scrum and XP, are described in terms of their processes, roles and practices.

Third, the thesis will discuss how agile methods can be tailored and which kind of experience has been got from the tailoring in practice. Five case studies will be analyzed based on the agile method tailoring framework composed of development context and its special features, tailoring process and strategy and outcome of the tailoring. The results of this thesis show how different processes, roles and practices from Scrum and XP have been tailored in different cases. 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.

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ohjelmistokehityksen yhteydessä ohjelmiston tai tietojärjestelmän kehittämismenetelmä on määritetty eri tavoin. Iivari, Hirscheim ja Klein (1998) määrittelevät sen 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 eri tietojärjestelmää rakentavien osapuolien työtä sekä yhteistyötä, ja näitä toimintatapoja tukee joukko suositeltuja tekniikoita, työkaluja sekä aktiviteetteja. Menetelmiä on perinteisesti käytetty ohjelmistojen kehittämisen yhteydessä jo pitkään, ja niitä on luokiteltu eri tavoilla. Niiden käyttämisestä käytännön työssä on saatu erilaisia hyötyjä. Tolvanen (1998) sanoo menetelmän käytön parantavan dokumentaatiota, systematisoivan ohjelmistokehityksen prosessia, parantavan mahdollisuuksia saada kehityksen lopputuloksen vastaamaan vaatimuksiaan sekä lisäävän käyttäjien osallistumista kehitystyöhön.

Menetelmiä voidaan muokata ja kehittää eri tavalla. Brinkkemper (1996) määrittelee menetelmän kehittämisen tarkoittavan työtä, jonka tarkoituksena on suunnitella, rakentaa ja soveltaa menetelmiä, tekniikoita ja työkaluja tietojärjestelmien kehittämistä varten. Menetelmäkehittämistä voidaan tehdä eri strategioin, erilaisiin konteksteihin ja erilaisin prosessein (Leppänen, 2005). Organisaation menetelmän kehittämistä, eli konfigurointia, sekä projektin menetelmän kehittämistä, eli kustomisointia, voidaan kutsua yhteisellä nimellä menetelmän räätälöinniksi. Backlund, Hallenborg ja Hallgrimsson (2003) toteavat, että menetelmä tulisi sopeuttaa ja räätälöidä organisaatioon ensiksi, mikäli siitä haluaa tehdä osan organisaation omaa tietämyspohjaa. Ohjelmistokehitykseen liittyy myös paljon tilanne– tai kontingenssitekijöitä, jotka vaikuttavat menetelmän räätälöintiin. Van Slooten, Brinkkemper ja Hoving (1996) näkevät kontingenssitekijät projektin olosuhteina, jotka järjestelmäkehityksessä jollain tapaa vaikuttavat valittavaan tai rakennettavaan lähestymistapaan. Räätälöintiä suoritetaankin usein näiden tekijöiden suhteen.

Ketterät kehittämismenetelmät ovat 2000–luvulla tulleet enemmän käyttöön ohjelmistojen kehitysmenetelmänä. Abrahamssonin, Salon, Ronkaisen

ja Warstan (2002) menetelmä on ketterä, mikäli se toimii inkrementaalisesti, korostaa yhteistoiminnallisuutta ja asiakaslähtöisyyttä, on suoraviivainen ja helposti opittava sekä valmis mukautumaan muutoksiin. Conboy (2009) taas toteaa, että tuskin mikään ketterä menetelmä määrittelee ketteryyden samalla tavalla. Scrum (Schwaber & Sutherland, 2010) ja XP (eXtreme Programming) (Beck, 1999) ovat kaksi suosittua ketterää kehittämismenetelmää, jotka pitävät sisällään erilaisia prosesseja, rooleja, vastuita ja käytänteitä.

Tässä tutkielmassa on tarkoituksena tutkia ketterän menetelmän räätälöintiä. Tutkielman tutkimusongelmat voidaan muotoilla seuraavasti:

- Mitä tarkoitetaan menetelmällä ja sen räätälöinnillä?
- Mitä tarkoitetaan ketterällä kehittämisellä?
- Millä perusteilla ja tavoilla ketteriä menetelmiä on esitetty räätälöitäväksi?
- Millaisia käytännön kokemuksia ketterien menetelmien räätälöinnistä on saatu?

Vastauksen saamiseksi ensimmäiseen tutkimusongelmaan perehdytään ensin tietojärjestelmien ja ohjelmistojen kehittämismenetelmiä ja niiden kehittämistä yleisellä tasolla käsittelevään kirjallisuuteen. Toisen tutkimusongelman edellyttämänä luodaan yleiskuvaus ketterästä kehittämisestä ja kahdesta ketterästä menetelmästä. Kolmanneksi työssä tehdään lyhyt kirjallisuuskatsaus ketterän menetelmän räätälöintiä koskevasta teoreettisesta tutkimuksesta, muodostetaan sen pohjalta viitekehys ja käytetään viitekehystä ketterän menetelmän räätälöintiä koskevien tapaustutkimusten analysointiin. Tutkielmaa varten on valittu viisi tapaustutkimusta, joissa kontekstina ja erityispiirteinä ovat ulkoistetut elektronisen liiketoiminnan projektit (Hong, Yoo & Chaa, 2010), ohjelmistotuoteperheiden suunnittelu (Díaz, Perez, Yagüe & Garbajosa, 2011), julkinen sektori (Scott, Johnson & McCullough, 2008), tietoturvapalveluihin keskittynyt monikansallinen, keskikokoinen yritys (Wang, Conboy & Pikkarainen, 2012) sekä suorittimien ohjelmistokehitykseen keskittynyt organisaatio (Fitzgerald, Hartnett & Conboy, 2006).

Tutkielma jakaantuu viiteen lukuun. Luvussa 2 käsitellään, mitä menetelmällä, menetelmän kehittämisellä ja menetelmän räätälöinnillä tarkoitetaan. Luvussa 3 käsitellään ketterää lähestymistapaa sekä ketteristä menetelmistä Scrumia ja XP:tä. Neljännessä luvussa käsitellään ketterän menetelmän räätälöintiä koskevia tutkimuksia, esitellään ketterän menetelmän räätälöinnin viitekehys ja käydään tapaustutkimuksia läpi viitekehyksen avulla. Tutkielma päättyy yhteenvetoon.

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, mitä hyötyjä voidaan sen käyttämisellä saavuttaa sekä millaisia menetelmiä on olemassa. Toiseksi jäsennetään menetelmän kehittämisen käsitettä viitekehyksen avulla. Lopuksi tarkastellaan menetelmän räätälöinnin käsitettä esittelemällä eri lähteissä olevia määritelmiä sekä tapoja esittää räätälöinti ja siihen vaikuttavia tekijöitä.

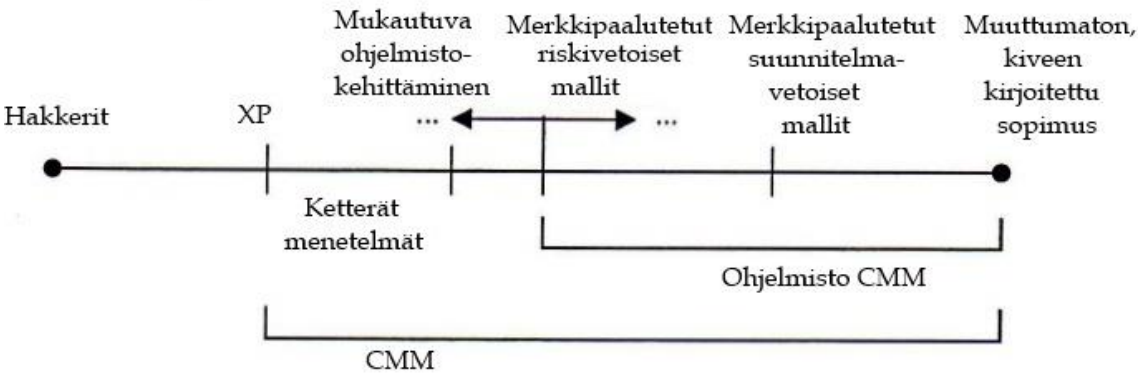
2.1 Menetelmästä yleisesti

Menetelmällä *(*engl. *method, methodology)* tarkoitetaan yleisesti ottaen "tapaa tai joukkoa tapoja tehdä jotakin" (Webster, 1989). Ohjelmistokehityksen yhteydessä menetelmälle on annettu erilaisia määritelmiä. 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. Tolvanen (1998) määrittelee kehittämismenetelmän olevan ennalta määritetty ja järjestetty kokoelma tekniikoita ja sääntöjä, jotka määrittelevät, millä tavalla, missä järjestyksessä ja kuka tekniikoita käyttää, jotta saavutetaan tavoitteet. 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.

Menetelmän käytön 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 ennalta–arvattuvuutta sekä tuovan yhteisen pohjan keskustelulle ja ymmärrykselle. Menetelmä auttaa lisäksi uusia työntekijöitä tunnistamaan, mistä heidän tulee tietää enemmän, ja helpottaa keskusteluja kokeneempien työntekijöiden kanssa, suojaavan ja tukevan tietämystä ottaen esteet viestinnän edeltä pois. Se myös auttaa tietämyksen kehityksen olemisessa ei–satunnaista tai ei–systemaattista ja helpottaa tietämyksen jakamista tarjoamalla muun muassa yhteisen terminologian ja työnkulut keskustelujen pohjaksi. (Schönström & Carlsson, 2003.)

Vuosikymmenien aikana on kehitetty lukuisa määrä erilaisia menetelmiä. Näitä menetelmiä voidaan luokitella eri tavoin. Eräs luokittelu perustuu siihen, miten tarkkaan määriteltyjä ohjeistoja menetelmät sisältävät ja miten tarkkaan niitä oletetaan seurattavan. Menetelmiä voidaan luokitella myös sen mukaan, miten paljon etukäteissuunnittelua edellytetään ennen ohjelmien koodaamista. Tällaisen luokituksen on esittänyt Boehm (2002) (kuvio 1).



KUVIO 1 Suunnitteluspektri (Boehm, 2002, 65)

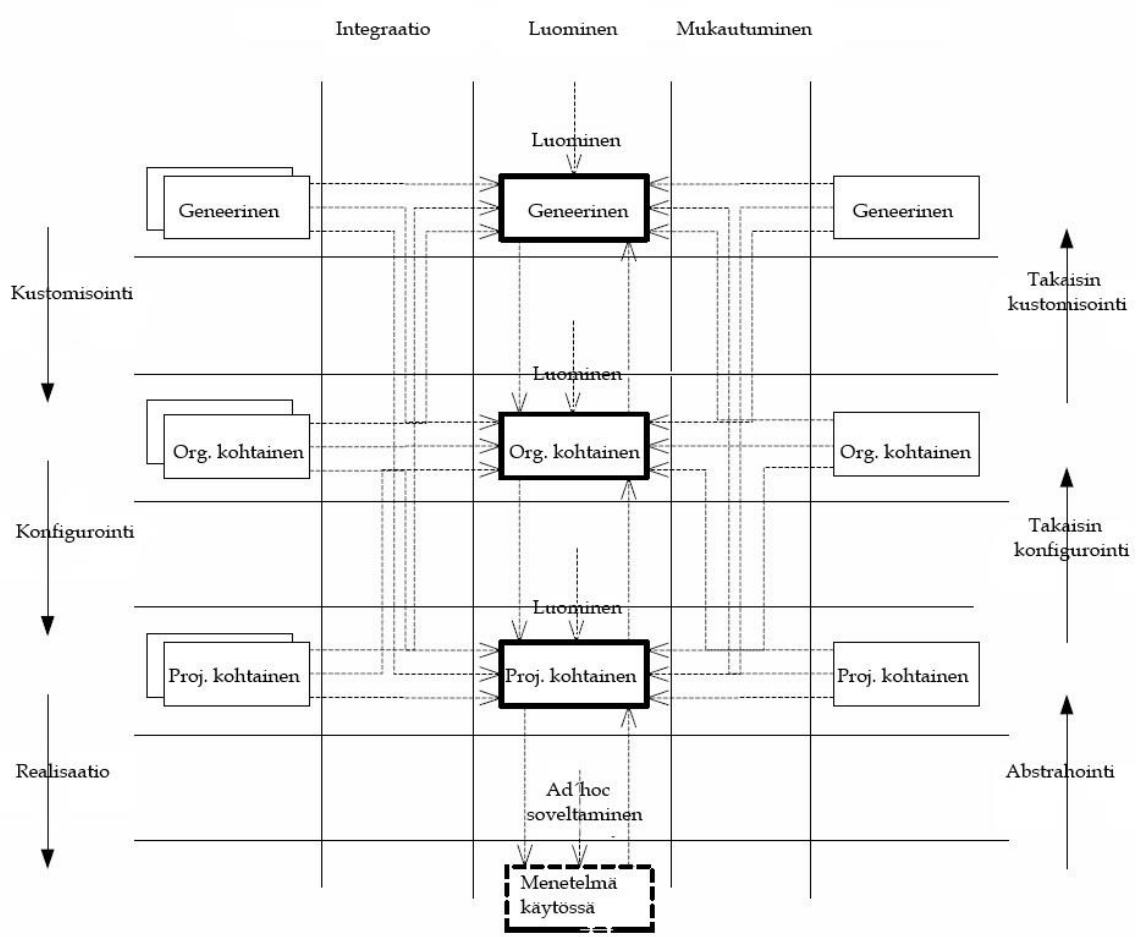
Suunnitteluspektrissä vasemmassa reunassa ovat niin kutsutut hakkerit. jotka eivät tee mitään suunnitelmia. Oikeassa laidassa taas ovat ne menetelmät, joissa edellytetään ”kiveenhakattuja” sopimuksia vaatimuksena ennen kuin niitä lähdetään edes toteuttamaan. Perinteisesti tietojärjestelmien kehittämismenetelmät ovat olleet etukäteissuunnittelultaan kuvion oikeassa laidassa, mutta 2000–luvulla ketterien menetelmien käytön myötä myös kuvion vasemmassa laidassa olevaa suunnitelmallisuusastetta on käytetty entistä enemmän.

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. 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 2). 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 jo olemassa olevia komponentteja tai palasia jo olemassa olevista menetelmistä. *Mukauttaminen* taas tarkoittaa olemassa olevan menetelmän komponenttien muokkaamista tai poisjättämistä, tai sen laajentamista uusilla osilla. (Leppänen, 2005.)



KUVIO 2 Menetelmän kehittämisen viitekehys (Leppänen, 2005, 439)

Menetelmän kehittämisen prosesseilla taas 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 (method engineering) 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.)

Seuraavassa yllä esitettyä konfigurointia ja kustomisointia kutsutaan yhdessä *menetelmän räätälöinniksi*.

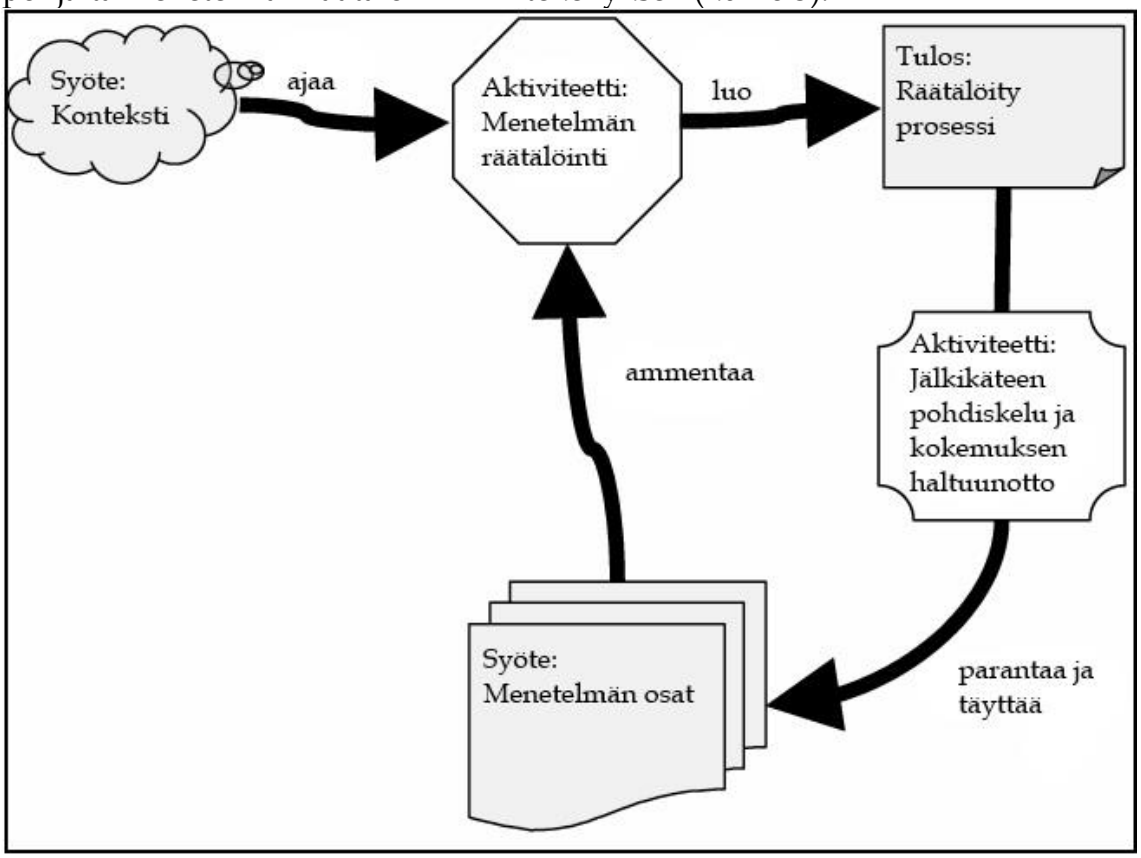
2.3 Menetelmän räätälöinti

Menetelmän räätälöinnistä käytetään kirjallisuudessa monenlaisia nimityksiä. Englanninkielisessä kirjallisuudessa siitä käytetään muiden muassa nimityksiä ”method customisation”, ”method adaptation” ja ”method configuration”. Sitä voi tapahtua niin organisaatio– 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 suorittaa sille 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 tulisi sopeuttaa ja räätälöidä organisaatioon ensiksi, mikäli siitä haluaa tehdä osan organisaation omaa tietämyspohjaa. 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 kahdella eri tasolla: organisaatiotasolla tai 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 (2003) taas määrittelevät räätälöinnin tietyn menetelmän sopeuttamisena erilaisten tilannetekijöiden mukaan, jolloin 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 tilanteeseen.

Menetelmän räätälöinnistä on kirjallisuudessa tehty erilaisia viitekehyksiä ja kattavampaan kuvaan tähtääviä tutkimuksia. Patel ym.(2004) ovat tutkineet useita menetelmiin liittyviä tutkimuksia ja verranneet niitä toisiinsa, muun muassa mistä lähtökohdista tutkimukset ovat lähteneet liikkeelle sekä millaista terminologiaa ne käyttävät räätälöinnissä. Heidän käyttämiään tutkimuksia ovat muun muassa Harmsen, Brinkkemper ja Oei (1994), Baskerville ja Stage (2001), Henninger, Ivaturi, Nuli ja Thirunavukkaras (2002) sekä Fitzgerald, Russo ja Stolterman (2003). Tutkimuksista kävi ilmi, että niissä on monia päällekkäisyyksiä niin havaintojen kuin termien osalta, vaikka esimerkiksi niiden painotusarvo saattoi vaihdella. Patel ym. rakensivat näiden tutkimusten pohjalta menetelmän räätälöinnin viitekehyksen (kuvio 3).



KUVIO 3 Menetelmän räätälöinnin viitekehys (Patel ym., 2004, 6)

Viitekehys koostuu kontekstista, menetelmän räätälöintiprosessista, menetelmäosista, räätälöidystä prosessista sekä kokemuksen haltuunotosta. *Kontekstilla* tarkoitetaan ympäristöä, josta projekti 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 menetelmäosaset. Menetelmän tulee jossain

määrin tukea tämän kaltaista modulaarisuutta, jotta räätälöintiä voidaan suorittaa.

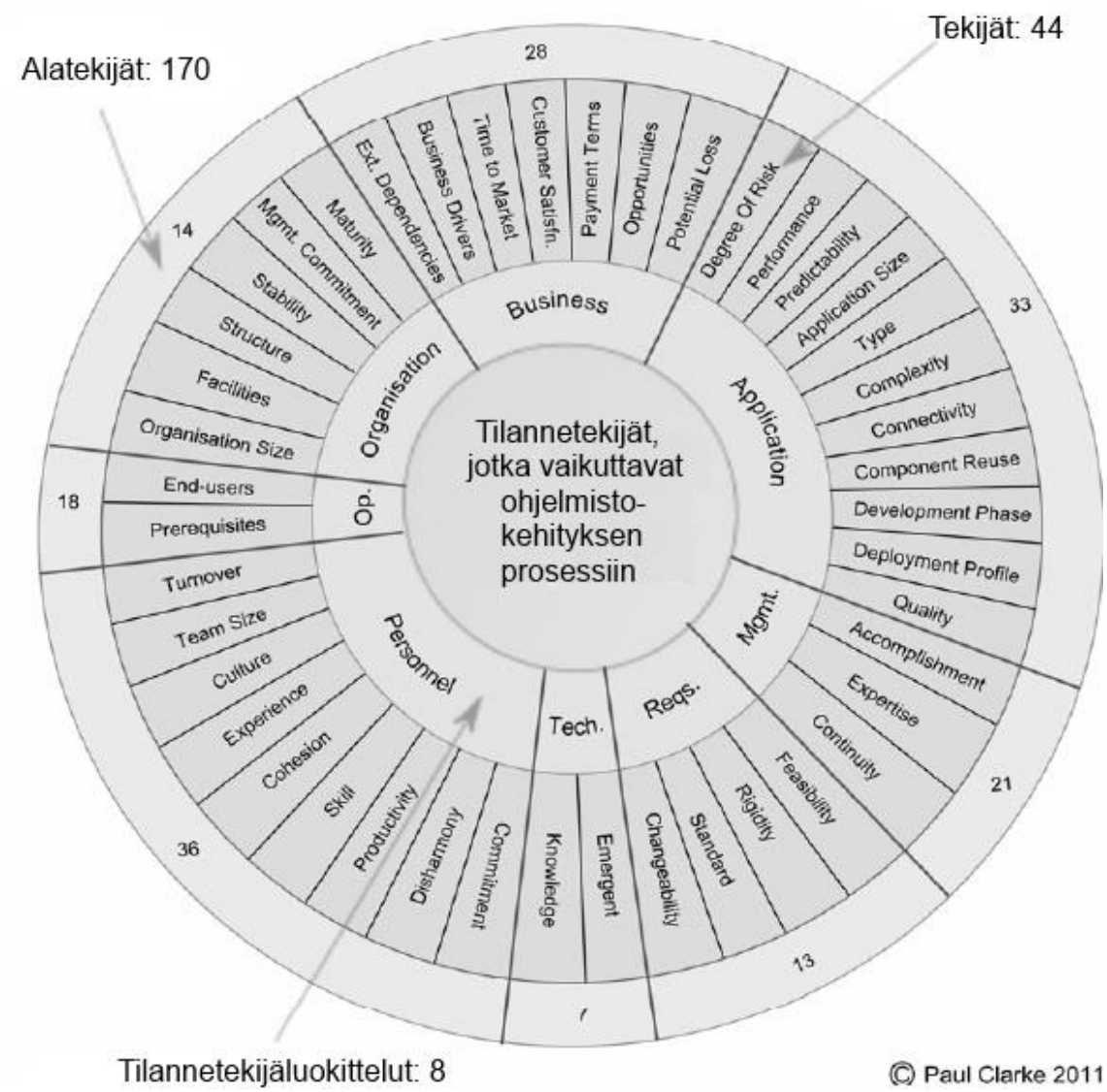
Menetelmän räätälöintiprosessi viittaa menetelmäosien valitsemis– ja kokoamisprosessiin, jonka avulla kehitetty menetelmä saadaan sopimaan kehittämiskontekstiin prosessiosien lisäämisellä, poistamisella sekä muuttamisella. Räätälöity prosessi/menetelmä on tämän prosessin lopputulema, jota käytetään itse projektissa tai organisaatiossa. Se pitää sisällään kaikki 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, ja 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.)

Kontekstia pyritään kuvailemaan tiettyjen ominaispiirteiden mukaisesti. Näitä piirteitä kutsutaan *kontingenssitekijöiksi* (engl. contingency factors) tai *tilannetekijöiksi* (engl. situational factors). Van Slooten ym. (1996) näkevät kontingenssitekijät projektin olosuhteina, jotka järjestelmäkehityksessä jollain tapaa vaikuttavat valittavaan tai rakennettavaan lähestymistapaan. Kontingenssitekijät ohjaavat menetelmän ja menetelmäosien valintaa menetelmäportfoliosta tilanteeseen sopivan menetelmän aikaansaamiseksi.

Kontingentti– ja tilannetekijöitä voidaan luokitella eri tavoin. Van Slooten ja Schoonhoven (1994) listaavat tällaisiksi muun muassa projektin ajan, suunnan, laajuuden, syvyyden, alkuperän, henkilöiden keskinäiset suhteet, budjetin, teknologian, asiakkaan standardit sekä kehittäjien ammattitaidon. Backlund (2002) määrittelee tilannetekijät viiteen luokitteluun yhdistettynä menetelmään, joka on käytössä. Nämä ovat liiketoiminta/kehityskonteksti, kehitettävä järjestelmä, kehittäjät, menetelmä sekä menetelmän rooli. Nämä jaetaan edelleen osiin.

Clarke ja O´Connor (2012) esittävät hyvin laaja–alaisen viitekehyksen tilannetekijöiden jäsentämiseksi (kuvio 4).



KUVIO 4 Tilannetekijät, jotka vaikuttavat ohjelmistokehityksen prosessiin (Clarke & O´Connor, 2012, 16)

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 tunnistettu vielä lisää (yhteensä 170 kappaletta) yksittäisiä tekijöitä. Tässä tutkielmassa ei ole mahdollisuutta tarkastella näitä tekijöitä tarkemmin. (Clarke & O´Connor, 2012.)

2.4 Yhteenveto

Menetelmällä tarkoitetaan kokoelmaa käytänteitä, tekniikoita, työkaluja ja dokumentaation apuvälineitä, jotka auttavat kehittäjiä ohjelmistokehityksessä. Menetelmä koostuu vaiheista, jotka jakautuvat edelleen osavaiheisiin.

Menetelmää voidaan kehittää joko ”tyhjästä”, integroimalla olemassa olevien menetelmien osia toisiinsa tai muokkaamalla menetelmän osia ja prosessia. Mikään menetelmä ei sellaisenaan sovellu kaikkiin tilanteisiin. Tästä syystä sitä joudutaan räätälöimään. Räätälöinti tehdään 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, organisaatioon, toimintaa, johtamiseen ja liiketoimintaan. Edellä mainittuja menetelmän ja sen räätälöinnin käsitteitä ja jäsennyksiä käytetään hyväksi luvussa 4, jossa rajaudutaan tarkastelemaan ketterien menetelmien räätälöintiä.

3 KETTERÄ KEHITTÄMINEN

Tässä luvussa tarkastellaan ensin lyhyesti ketterää lähestymistapaa ja sen jälkeen kahta ketterää menetelmää. Lähestymistavasta pohditaan ketteryyden käsitettä ja esitellään Agile–manifestia. Lukuisista ketteristä menetelmistä tarkasteluun on valittu Scrum ja XP. Niistä käydään lävitse 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, 2006). Se ei myöskään rajoitu pelkästään ohjelmistokehityksen tieteenhaaraan. Esimerkiksi Wong ja Whitman (1999) määrittelivät yrityksen olevan ketterä, mikäli se pystyy nopeasti vastaamaan huomisen ennalta arvaamattomiin ja odottamattomiin muutoksiin. Ohjelmistokehityksessä ketteryyden määritelmä ja arvot pohjautuvat usein pääasiallisesti eri kehittäjistä koostuvan Agile–allianssin (2001) laatimaan kettärän ohjelmistokehityksen manifestiin, sen arvoihin ja periaatteisiin:

> 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.
>
> Toisin sanoen, vaikka oikealla puolella olevissa asioissa on arvoa, arvostamme enemmän vasemmalla puolella olevia asioita. (Agile–alliance, 2001)

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, eikä menetelmien lähtökohdat tai pääpaino ole sama eri menetelmillä. Hän näkee joustavuuden (flexibility) ja keveyden (leaness) osana ketteryyttä. Ketteryyden hän määrittelee näin:

> 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).

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 (Anderson, 2010). Tässä tutkielmassa rajoitutaan tutkimaan Scrumia ja XP:ä, koska ne ovat kaksi käytännön työssä suosituimpia sekä 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. Se soveltuu etenkin pienten, mielellään alle kymmenestä ja yli kolmesta henkilöstä koostuviin tiimeihin. (Schwaber & Sutherland, 2010). Menetelmä tarjoaa etenkin projektin johtamisen käytäntöjä enemmän kuin muut ketterät menetelmät (Conboy, 2009). Menetelmässä prosessi on jaettu kolmeen kokonaisuuteen, esipelivaiheeseen (engl. pregame phase),

peli/kehitysvaiheeseen *(*engl. development phase*)* sekä loppupelivaiheeseen *(*engl. postgame phase*)*. Kehitys etenee iteratiivisesti sekä inkrementaalisesti. (Abrahamsson ym., 2002.)

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, 2011.)

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.)

Scrumtiimissä on kolmenlaisia henkilörooleja. *Scrum–mestari (*engl. *Scrum Master)* on hallinnollinen henkilö, jonka vastuulla on projektin ja sprittien eteneminen Scrumin käytäntöjen, arvojen ja sääntöjen mukaisesti. Samalla hän myös toimii tiimiä palvelevana johtajana ja pyrkii maksimoimaan tiimin työn arvon. *Kehittäjätiimi (*engl. *development team)* koostuu ohjelmistoalan ammattilaisista. Tiimi tuottaa kehitettävää ohjelmistoa inkrementeissä, jolloin ominaisuus siirtyy työlistasta valmiiksi. Scrumissa 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 ja tiimin työn arvon varmistamisesta. Tuoteomistaja voi esimerkiksi olla työntekijä kohdeorganisaatiosta, jolle ohjelmistoa ollaan kehittämässä. (Schwaber & Sutherland, 2011.)

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, tarkentaessaan tai neuvotellakseen tarvittaessa tuoteomistajan kanssa sprintin työlistasta. (Schwaber & Sutherland, 2011.)

*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, 2011.)

Sprintin lopussa järjestetään *sprinttikatselmus (*engl. *Sprint Review)*, jossa tuoteomistaja katsoo mikä osa työstä on ”valmista” ja mikä ei ole ”valmista”. Tällöin tiimi ja eri sidosryhmät 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, tarvittaessa tehden suunnitelman tiimin tekemisen ja työskentelemisen parantamiselle. (Schwaber & Sutherland, 2011.)

3.3 XP

Extreme Programming eli XP on toinen suosittu ketterä kehittämismenetelmä, jonka pääpaino on ohjailevilla toimintaohjeilla itse kehittäjille (Conboy, 2009). Se on suunniteltu vastaamaan ohjelmistokehityksen eritystarpeita, jossa pienet tai keskisuuret tiimit harjoittavat kehitystä 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.)

*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 toiminnalisten 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 isoa kuvaa kehitettävästä ohjelmistosta, ja arvioida missä tahdissa ohjelmisto esimerkiksi täyttää tietyt vaatimukset. Hän myös dokumentoi muun muassa toiminnallisten testien tulokset. (Beck, 1999.)

*Valmentaja (*engl. *coach)* on vastuussa kehitysprosessista kokonaisuutena. Hänen tulee ymmärtää ohjelmiston normaalia syvällisemmin ja ohjeistaa kehitystiimiä. 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. (Beck, 1999.)

XP:n prosessi jaetaan kuuteen osaan. *Tutkimisvaiheessa (*engl. *exploration)* asiakas tekee tarinakortteja tulevan ohjelmiston vaatimuksista. Samaan aikaan ohjelmoijat kehittävät arkkitehtuuri– ja teknologiaratkaisuja, tarvittaessa konsultoiden aiheesta hyvin tietäviä. Myös laitteistoa voidaan testata ja miettiä reaalimaailman elementeillä, kuten realistisella määrällä verkkoliikennettä. Mikäli kehitystiimi on ennestään toisille tuttu 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 toiminnalisuus on tärkeämpänä osana iteraatoiden sisältöä. Asiakas valitsee mitkä tarinat valitaan iteraatioon toteutettavaksi. Jokaisen iteraation lopussa testit on mahdollista ajaa ja järjestelmä on toimintavalmis. (Beck, 1999.)

*Tuotteistaminen (*engl. *productionizing)* on julkaisun jälkeistä palautteen antamista ja suorituskyvyn testausta. Tuottaistemisen 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.

*Lopetusvaiheessa (*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. (Beck, 1999.)

Edellä mainittujen lisäksi XP:ssä on useita toisiinsa liittyviä tai toisistaan erillään olevia käytänteitä. *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. (Beck, 1999.)

*Pariohjelmointi (*engl. *pair programming)* on käytäntö, jossa ohjelmakoodin kirjoittamista suoritetaan kehittäjäpareissa. Tällöin yhden tietokoneen edessä 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 koko 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 pari ajavat tarvittavia testejä niin kauan, kunnes ne kaikki menevät lävitse. *40 tunnin työviikko (*engl. *40–hour week)* tarkoittaa työmäärän pitämistä kurissa. Ihminen ei jatkuvasti pysty työskentelemään ylikuormitettuna samaan aikaan kun pitää olla luova työntekijä. Kahdella perittäisellä viikolla ylitöiden teettäminen on kiellettyä. (Beck, 1999.)

*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.)

4 KETTERÄN MENETELMÄN RÄÄTÄLÖINTIÄ KOSKEVIA TUTKIMUKSIA

Tässä luvussa tarkastellaan ketterän menetelmien räätälöintiä. Ensiksi luodaan yleiskatsaus aihealuetta tarkastelevaan kirjallisuuteen. Toiseksi esitetään viitekehys, jolla voidaan jäsentää ja verrata ketterän menetelmän räätälöintiä koskevia tapaustutkimuksia. Kolmanneksi tarkastellaan viitekehyksen avulla viittä tapaustutkimusta. Lopuksi esitetään yhteenveto luvusta.

4.1 Aihetta käsittelevä kirjallisuus

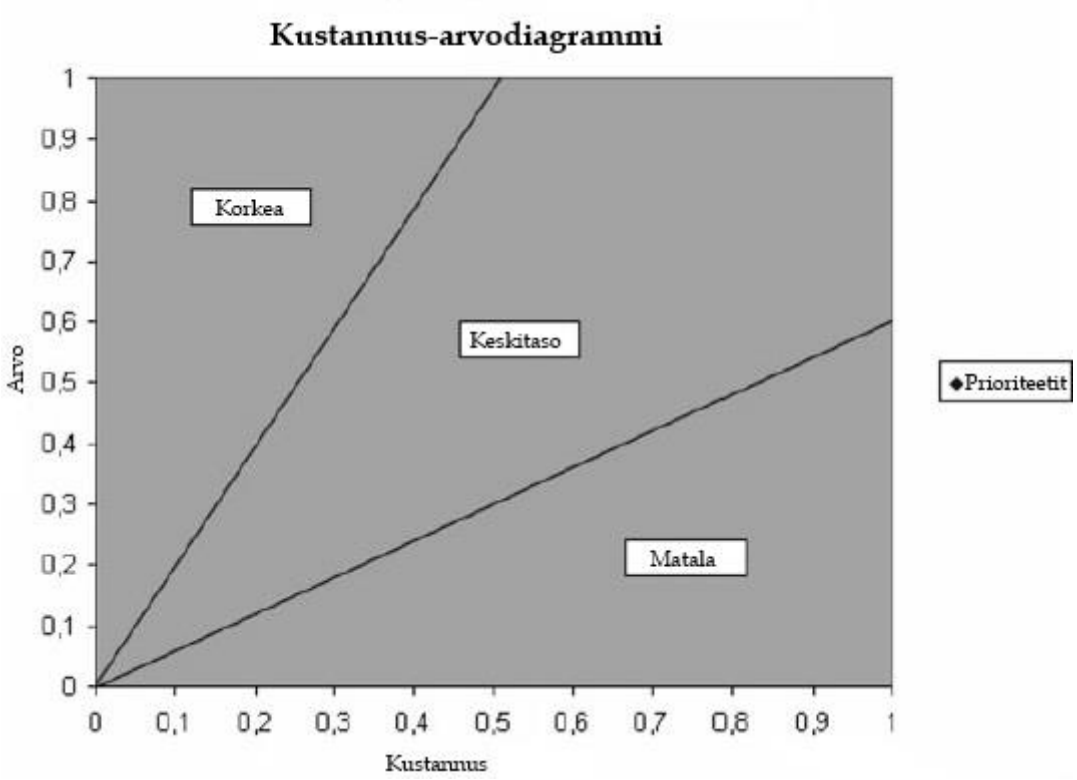
Ketterän menetelmän räätälöinnistä on tehty monenlaisia tutkimuksia. Osa näistä on empiirisiä, lähinnä tapaustutkimuksia, joissa on raportoitu ja analysoitu ketterän menetelmän räätälöintiä joko tietylle organisaatiolle (konfigurointi) tai projektille (kustomisointi). Tutkimuksissa kerrotaan kontekstista ja sen erityispiirteistä, kuten kehittämisen mahdollisesta hajautuksesta, organisaatiorakenteesta tai kehitettävän ohjelmiston ominaisuuksista. Esimerkkeinä tälläisistä kontekstista ovat globaalisti hajautettu ohjelmistokehitys (Hossain, Bannerman & Jeffery, 2011), sulautetut järjestelmät (Salo & Abrahamsson, 2008), turvallisuuskriittiset järjestelmät (Bowers, May, Melander, Baarman & Ayoob, 2002), ohjelmistotuoteperheen suunnittelu (Díaz ym., 2011), ulkoistetut elektronisen liiketoiminnan projektit (Hong ym., 2010), hajautettu kaupallisen väliohjelmiston kehitys (Wang ym., 2012), kaupallisten tietoturvapalveluiden ja ratkaisuiden kehittäminen (Wang ym., 2012), räätälöidyt tietoturva– ja johdon ohjelmistot (Wang ym., 2012), räätälöidyt web–pohjaiset järjestelmät (Wang ym., 2012), yritysjärjestelmät (Bass, 2012), julkinen sektori (Scott ym., 2008), Sri Lankan ympäristö (Jayawardena & Ekanayake, 2010) ja suorittimien ohjelmistokehitykseen keskittyvä organisaatio (Fitzgerald ym., 2006).

Toisen osan kirjallisuutta muodostavat ne julkaisut, joissa esitetään viitekehyksiä ja ohjeita ketterän menetelmän räätälöinnistä organisaation ja

projektien käyttöön. Keenan (2004) pitää räätälöintiä edellytyksenä ketterän menetelmän käyttämiselle. Hänen mukaansa tulisi tutkia, kuinka voidaan määritellä, ylläpitää ja tehdä saavutettavaksi tietämysvarasto, joka pitää sisällään yksityiskohtaiset prosessit, tekniikat ja kokemukset räätälöinnistä. Ideat ja tekniikat saataisiin esille kokemuksen avulla ja parhaat käytänteet jaettua toimialakohtaiseksi. El–Said, Hana ja Eldin (2009) esittävät ketterän menetelmän räätälöintityökalun (agile tailoring tool), joka tarjoaa matemaattisen mallin avulla tukea ketterien menetelmien valitsemiseen ohjelmistokehityksen eri vaiheisiin.

Mikulenas ja Kapocius (2011) ovat esittäneet erilaisia keinoja, miten priorisointia voidaan käyttää hyödyksi ketteriä käytäntöjä valittaessa. Näitä keinoja ovat Top10 vaatimukset –tekniikka, jossa viiteryhmät priorisoivat kymmenen mielestään tärkeää käytäntöä, käytäntöjen numeroiminen prioriteeteittain, käytäntöjen arvioiminen sanallisesti prioriteetin mukaan esimerkiksi kriittisiin, normaaleihin sekä valinnaisiin käytäntöihin, sekä sadan dollarin testi. Viimeksi mainitussa testissä sidosryhmät saavat sata kuvitteellista rahayksikköä (raha, tunnit yms.) jaettavaksi eri käytäntöjen kesken. (Mikulenas & Kapocius, 2011.)

Näissä tekniikoissa on kuitenkin joitain ongelmia, kuten ainoastaan yhteen muuttujaan keskittyminen. Ratkaisuksi Mikulenas ja Kapocius (2011) esittävät prosessimallin, jonka avulla ketteriä käytänteitä ja prosesseja pystytään valitsemaan tiettyihin olosuhteisiin. Prosessin ensimmäisessä vaiheessa tunnistetaan tärkeimmät ja ongelmallisimmat projektialueet ja ongelmat. Tämän jälkeen kerätään yhdessä lista sopivista ketteristä käytänteistä, jotka soveltuvat edellä mainittujen ongelmien ratkaisemiseksi. Käytänteiden keräämisen jälkeen ne arvioidaan numeerisesti ja priorisoidaan verraten niiden tuottamaa arvoa toisiinsa. Arvon lisäksi käytänteitä verrataan niitten tuottamien kustannusten perusteella. Lopuksi asetetaan käytänteelle saadut kustannus– ja arvokertoimet kustannus–arvodiagrammille (kuvio 5). (Mikulenas & Kapocius, 2011.)



KUVIO 5 Kustannus–arvodiagrammi (Mikulenas & Kapocius, 2011, 491)

Kyseisellä diagrammilla voidaan verrata eri käytänteitä toisiinsa. Kun vertailut on suoritettu, voidaan päätökset käytettävistä käytänteistä tehdä niiden tuottaman hyödyn ja kustannusten perusteella. (Mikulenas & Kapocius, 2011.)

4.2 Ketterän menetelmän räätälöinnin viitekehys

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

Räätälöinnin *kohteena* voi olla jokin ketterä menetelmä (esim. Scrum tai XP) tai niiden osa (esim. jatkuva integraatio, testilähtöinen kehittäminen). *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 pilotoinnoin ja kokeilun kautta räätälöinti organisaatiolle sopivan menetelmän kehittämiseksi, tai perusteellinen

menetelmän suunnittelu, joka edeltää projektien käynnistämistä. *Räätälöintiprosessi* koostuu askeleista, jotka useimmiten etenevät kontekstin tilannetekijöiden tunnistamisesta ja räätälöitävän menetelmän valinnasta menetelmän osien (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 ilmaista millä tavalla se eroaa olennaisilta osiltaan pohjalla olleesta menetelmästä, mitä prosesseja, rooleja ja käytänteitä räätälöinnin kohteesta on säilytetty, mitä niistä on muutettu ja mitä niistä on jätetty pois.

Menetelmien räätälöintiä käsittelevästä laajasta kirjallisuudesta on valittu seuraavassa alaluvussa tarkasteltaviksi tapaustutkimuksia seuraavin kriteerein. Ensinnäkin edellytetään, että tutkimuksessa on raportoitu riittävän seikkaperäisesti edellä mainituista seikoista. Toiseksi tarkoituksena on saada joukkoon erilaisia tapauksia. Kolmanneksi edellytetään, että räätälöinnin kohteena on ollut joko Scrum, XP tai näiden yhdistelmä siksi, että näitä menetelmiä on tässä tutkielmassa aiemmin esitelty. Näiden kriteerien mukaisesti on valittu viisi tapaustutkimusta: Hong ym. (2010), Díaz ym. (2011), Scott ym. (2008), Wang ym. (2012) ja Fitzgerald ym. (2006).

4.3 Tapaustutkimusten tutkiminen

Tässä alaluvussa kuvataan ensin valitut tapaustutkimukset ja sen jälkeen esitetään niistä yhteenveto, joka on muodostettu käyttäen aiemmin määriteltyä viitekehystä.

4.3.1 Ulkoistetut elektronisen liiketoiminnan projektit

Hong ym. (2010) tutkivat ketterän menetelmän räätälöintiä ulkoistettujen elektronisen liiketoiminnan projektien tapauksessa. Tapaustutkimuksen kohteena ollut yritys on eteläkorealainen elektronisen liiketoiminnan yritys, joka on käyttänyt kolmansien osapuolten palveluita kehitystyössä saadakseen uudet palvelut ajoissa valmiiksi sekä vähentääkseen kustannuksia. Menetelmänä yrityksellä on ollut käytössä 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 käyttämiselle 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. (Hong ym., 2010.)

Scrumia muokattiin kolmelta eri osa–alueelta. Scrum–menetelmästä roolien ja vastuiden jakautuminen koettiin liian epäselväksi. Sen 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 jokaisen kehitysvaiheen aikana tehdä. Tämän havainnollistamiseksi luotiin taulukko, josta ilmenee jäsenten roolit ja vastuut jokaisen sprintin aikana. (Hong ym. 2010.)

Toiseksi projektien keston ollessa pitkä 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 luomisen ongelmaan ratkaisuna luotiin projektin alussa oleva *pääsprintti (*engl. *master sprint)*, jonka aikana määritellään koko projektin merkkipaalut 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. (Hong ym., 2010.)

Kolmantena räätälöinnin kohteena ollut osio on 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. (Hong ym., 2010.)

Kokonaisuutena yritys sai mitattavia hyötyjä Scrumin räätälöinnistä ulkoistettuihin projekteihin. Vesiputousmallia käyttämällä sisäisten projektin onnistumisprosentti oli vuonna 2009 89 prosenttia, 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 kasvoi. (Hong ym. 2010.)

4.3.2 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 käytännön työssä sovellettu testauksen kehitysympäristössä. (Díaz ym., 2011.)

Ohjelmistotuoteperheen ja ketterän kehittämisen välillä voidaan nähdä joitain ristiriitoja. Niiden sovittaminen on kuitenkin mahdollista tavalla, jossa molempien olennaiset periaatteet säilyvät (Diáz ym., 2011). Ongelmana on etenkin ohjelmistotuoteperheen vaatima arkkitehtuuri– ja pitkäaikaissuunnittelu kettärän 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 ja ne tukevat niin ulkoista evoluutiota, eli muutoksia arkkitehtuurin konfiguraatiossa, ja sisäistä evoluutiota, eli muutoksia itse komponentin sisällä. Arkkitehtuurin konfiguraation muuttaminen tapahtuu lisäämällä tai poistamalla muita komponentteja ja yhteyksiä. PPC:llä on erilaisia *vaihtelevuuskohtia (*engl. *variability points*), jotka määrittelevät sen vaihtelevuuden, sekä *variantteja (*engl. *variant)*, jotka liittävät ohjelmakoodin osia siihen. Vaihtelevuuskohdat 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. (Diáz ym., 2011.)

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 tekijöihin ja sidosryhmien vaatimuksiin. Arkkitehtuuria kehitetään inkrementaalisesti ja iteratiivisesti, ja se on valmis ottamaan vastaan odottamattomia muutoksia koko ohjelmistonkehityksen elinkaaren ajan. (Diáz ym., 2011.)

Kolmas konsepti on *reflektiivinen uudelleenkäyttö (*engl. *reflective reuse)*. Käytäntö soveltaa systemaattista uudelleenkäyttöä tuoteperheiden

suunnittelussa sekä opportunistista uudelleenkäyttöä, jota käytetään ketterien menetelmien yhteydessä. Reflektiivisessä uudelleenkäytössä koodiin uudelleenkäyttö suunnittellaan aina lyhyeksi aikaa kerrallaan. Scrumin tapauksessa luonteva aika tälle on yksi sprintti. Suunnittelun jälkeen koodia kirjottaessa 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. (Díaz ym, 2011.)

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 ne sprintteihin. Yleisten ominaisuuksien priorisoinnissa tulisi käyttää hyödyksi reflektiivisen uudelleenkäytön mahdollisuutta. Seuraavaksi sprintin suunnittelupalaverissa määritetään yksittäisen sprintin tavoite ja työlista. (Díaz ym., 2011.)

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 vaihtelevuuskohdat 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. (Díaz ym, 2011.)

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 vaihtelevuuskohtia ja niiden variantteja. Ketterällä ohjelmistotuoteperheiden suunnittelulla saatiin joustavuutta kehitykseen. (Díaz ym, 2011.)

4.3.3 Julkinen sektori

Calgary on Kanadan neljänneksi suurin kaupunki, jossa on yhteensä noin miljoona asukasta. Sillä on yhteensä 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, 2008.)

Räätälöintistrategiaksi oli valittu pilottiprojektin tekeminen Calgaryn kaupungille kustomisoiden projektin menetelmä valikoimalla prosesseja ja käytänteitä pääasiallisesti Scrumista ja XP:stä. Samalla tarkoitus oli kehittää organisaatiokulttuuria kohti ketterän kehittämisen periaatteita ja arvoja. 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 kokemusta 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. (Scott ym., 2008.)

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. (Scott ym., 2008.)

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. (Scott ym., 2008.)

Pilottiprojektissa otettiin käytänteitä ja prosesseja niin Scrumista kuin XP:stä. Huolimatta onnistuneesta projektista, koko organisaation kulttuurin ja kaiken kehitystyön muuttamiseksi ketteräksi on monenlaisia haasteita. Projekti kuitenkin osoitti, että ketterä kehittäminen voidaan räätälöidä myös hierarkkiaallisiin organisaatioihin. Projekteja toteuttaessa tulee ottaa huomioon riittävä organisaation tuki uuden menetelmän käyttöönotolle. (Scott ym., 2008.)

4.3.4 Tietoturvallisuuspalveluihin keskittynyt kaupallinen tuote

Wang ym. (2012) ovat tehneet useita tapaustutkimuksia ketterän kehittämiseen liittyen etenkin *prosessi–innovaatioden* (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, on kuusihenkinen, ja sille järjestettiin koulutusta niin Scrumista kuin XP:stäkin. Pilottiprojektin tekemisestä on ollut kirjoitushetkellä 1,5 vuotta ja tiimi on toiminut ketterästi sen jälkeen, joten käytännöt ovat muotoutuneet vielä räätälöinnin jälkeenkin. Strategiana räätälöinnillä olikin konfigurointi kehittäjätiimin käyttämäksi menetelmäksi. (Wang ym., 2012.)

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. (Wang ym., 2012.)

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 ottaesssaan käytäntöä toimeen. Saatuaan ongelmat ratkaistua, tiimi alkoi tekemään automaattista *käännöstä* (engl. *build)* tunneittain. Jatkuvan integraation avulla testausta saatiin tehtyä merkittävästi helpotettua. (Wang ym., 2012.)

Koodin yhteisomistajuus oli myös käytössä vielä pidemmälle vietynä kuin XP:ssä on alunperin 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 kustannuksista. Tiimi käytti myös avointa työtilaa. (Wang ym., 2012.)

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. (Wang ym., 2012.)

4.3.5 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 ohjelmisto– sekä piisirujen suunnittelu tehdään Irlannissa. Tutkimuksen avulla selvitettiin miten Scrumia ja XP:tä räätälöityivät organisaatiossa käyttöönotossa. Yritys on käyttänyt sisäisesti *prosessien kypsyysmallin (*engl. *capability maturity model, CMM)* tason 2 mukaista prosessimallia, mutta se oli siirtynyt markkinoiden vaatiessa siirtynyt käyttämään myös ketteriä menetelmiä. Tutkimuksen teon aikaan Intel Shannon on käyttänyt XP:tä viisi vuotta ja Scrumia kolme vuotta, Scrumia projektin hallintaan ja XP:ä varsinaiseen ohjelmistokehitykseen. (Fitzgerald ym., 2006.)

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 onkin käytänteenä yleisesti käytössä, mutta joihinkin soveltumattomiin vaiheisiin kehitystä sitä ei käytetä. Testivetoinen kehitys oli sovellettu niin, että yksikkötestit kirjoitetaan samaan aikaan kun toteutus tehdään. Tämän oli huomattu antavan suunnan kehitykselle ja ymmärrystä kehitettävästä kohteesta. Refaktorointi on käytössä, ja sen oli havaittu vähentävän virheitä. Yksinkertainen suunnittelu on refaktorointiin yhteydessä, ja sitä sovellettiin tekemällä suunnittelu valkotaululle jokaisen ohjelmakoodilohkon tapauksessa, jolloin toteutusta voidaan 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. (Fitzgerald ym., 2006.)

Intel Shannonilla jätettiin käyttämättä monia XP:n käytäntöjä. Suunnittelupeliä ei käytetty, vaan sen sijasta käytettiin Scrumin käytäntöjä. Pienten julkaisujen käyttäminen ei soveltunut käyttöön, sillä suorittimien tapauksessa ohjelmistojulkaisut on sidottu niihin. Jatkuvaa integrointia sovelletaan jokaiselle komponentille erikseen, mutta kokonaisvaltaista integrointia ei tehdä 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 ole saatavilla, joten kyseisen käytännön soveltaminen ei myöskään ollut mahdollista. (Fitzgerald ym., 2006.)

Scrum oli ollut Intel Shannonilla käytössä kolme vuotta, ja osana melkein jokaiseta 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 ennaltamäärättyä piisirujen tapauksessa. Jokaisen sprintin alussa tiimi päättää, 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. (Fitzgerald ym., 2006.)

Ketterien menetelmien hyötyinä todennettiin Intel Shannonilla parantunut ohjelmakoodin laatu ja virheiden väheneminen tuotteissa. Ketteriä menetelmiä ei sovellettu sellaisenaan, vaan pragmaattisesti valittiin sopivimmat käytännöt käyttöön.

4.4 Yhteenveto tapaustutkimuksista

Tämän luvun tarkoituksena oli tarkastella ketterän menetelmän räätälöintiä koskevaa kirjallisuutta. Alussa esitetyn yleiskatsauksen jälkeen luotiin viitekehys tapaustutkimusten jäsentämiseen. Sen jälkeen kuvattiin viittä tapaustutkimusta. Tässä alaluvussa esitetään yhteenveto tapaustutkimuksista käyttäen pohjana viitekehystä. Tiivistelmä yhteenvedosta on esitetty taulukossa 1.

TAULUKKO 1 Yhteenveto tapaustutkimuksista

Tutkimus	Pohjana käytetty menetelmä(t)	Konteksti ja erityispiirteet	Räätälöintiprosessi- ja strategia	Räätälöity menetelmä ja sen eroavaisuudet lähtökohtana olleseen menetelmään
Hong, Yoo & Chaa (2010)	Scrum	Ulkoistetut elektronisen liiketoiminnan projektit	Räätälöity Scrum muutamalta olennaiselta osalta vastaamaan organisaation tarpeita	Henkilöiden roolit ja vastuut taulukoidaan, projektin alkuun pääsprintti, mittarina käytetään valmiita nettisivuja
Díaz, Perez, Yagüe & Garbajosa (2011)	Scrum	Ohjelmistotuoteperheiden suunnittelu	Ketterän kehittämisen ja ohjelmistotuoteperheiden konseptien räätälöinti yhteen organisaation tasolla	Mukautuvien osakomponenttien käyttäminen, toimivan arkkitehtuurin luominen, reflektiivisen uudellenkäytön käyttöönotto, Scrumin prosessimallin mukautus sopimaan ohjelmistotuoteperheiden kanssa
Scott, Johnson & McCullough (2008)	Scrum & XP	Julkinen sektori	Pilottiprojekti, jossa Scrum kustomisoidaan julkiselle sektorille käyttäen ulkopuolista konsultointia hyödyksi	Scrumin ja XP:n käytäntöiden valitseminen sopivuuden mukaan, organisaation mukautuminen hitaampaa
Wang, Conboy & Pikkarainen (2012)	Scrum & XP	Tietoturvallisuuspalveluihin keskittynyt kaupallinen tuote, keskikokoinen monikansallinen yritys	Pilottiprojekti räätälöimälle organisaation kehittäjätiimille, ulkopuolisen konsultoinnin suuri hyödyntäminen	Suurin osa käytännöistä otettu käyttöön ja viety alkuperäistä ohjetta pidemmälle, muutama käytäntö jätetty XP:stä käyttämästä johtuen erinäisistä syistä
Fitzgerald, Hartnett & Conboy(2006)	Scrum & XP	Suorittimien ohjelmistokehitykseen keskittynyt organisaatio	Räätälöity organisaatiolle ottamalla sopivat osat käyttöön ja ottamatta soveltumattomat osat	Noin puolet XP:n käytänteistä käytössä ja puolet ei, Scrumin prosessimallia muokattu hieman tukemaan organisaation erityispiirteitä

Kahdessa tapaustutkimuksessa oli räätälöinnin kohteena Scrum ja kolmessa tapaustutkimuksessa Scrumin ja XP:n yhdistelmä. Hongin ym. (2010) tapaustutkimus käsitteli ulkoistettuja elektronisen liiketoiminnan projekteja, Díaz ym. (2011) käsittelivät ketterää ohjelmistotuoteperheiden suunnittelua, Scott ym. (2008) julkista sektoria, Wang ym. (2012) monikansallisen yrityksen tietoturvapalveluihin keskittynyttä projektitiimiä sekä Fitzgerald ym. (2006) käsittelivät suorittimien ohjelmistokehitykseen keskittynyttä organisaatiota.

Ulkoistettujen elektronisen liiketoiminnan projektien tapauksessa Scrum räätälöitiin, jotta projektien onnistumista voitaisiin parantaa ja laatua parantaa. Räätälöity menetelmä erosi Scrumista sillä, että projektin henkilöiden roolit ja vastuut oli taulukoitu, projektin alkuun lisättiin pääsprintti ja mittarina käytettiin valmiita nettisivuja. Ohjelmistotuoteperheen tapauksessa tehtiin muokkauksia niin ohjelmistiotuoteperheiden käsitteisiin kuin myös ketterään menetelmään. Toteutuksessa käytettiin muokautuvia osakomponentteja, arkkitehtuuri luotiin toimivaksi vastaamaan muuttuvia tekijöitä ja sidosryhmien vaatimuksia sekä käytettiin reflektiivistä uudelleenkäyttöä hyödyksi. Samalla Scrumin prosessimalli muokattiin tukemaan ohjelmistuoteperheiden suunnittelua. Julkisen sektorin tapauksessa tehtiin pilottiprojekti Calgaryn kaupungille. Tällöin kustomisoitiin Scrumin ja XP:n käytäntöjä projektin menetelmäksi käyttäen konsultointia hyödyksi, ja käytänteet valittiin sopivuuden mukaan. Tietoturvapalveluihin keskittyneen yrityksen tapauksessa Scrum ja XP oli otettu käyttöön 1,5 vuotta aiemmin, ja niiden käytännöt valittu tai jätetty valitsematta sopivuuden mukaan. Osa käytännöistä oli viety ohjekirjan mallia pidemmälle. Suorittimien ohjelmistokehityksen tapauksessa 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.

5 YHTEENVETO

Tämän tutkielman tarkoituksena oli tarkastella ketterän menetelmän räätälöintiä. Luvussa 2 tarkasteltiin mitä menetelmä tarkoittaa ohjelmistokehityksen yhteydessä ja millaisia määritelmiä sille on, mitä menetelmäkehittäminen tarkoittaa ja millaisia ulottuvuuksia sillä on, mitä menetelmän räätälöimisellä tarkoitetaan sekä millaisia ohjelmistokehitykseen vaikuttavia tilannetekijöitä on olemassa. Luvussa 3 esiteltiin ketterän ohjelmistokehityksen manifesti ja mitä määritelmiä ketteryydelle on olemassa, millaisia ketteriä menetelmiä on sekä tarkemmin millaisia prosesseja, rooleja ja käytänteitä on Scrumissa ja XP:ssä. Luvussa 4 tarkasteltiin, millaisia ketterän menetelmän räätälöintiä koskevia tutkimuksia on olemassa. Osa näistä käsittelee empiirisesti pääasiallisesti tapaustutkimuksena räätälöintiä johonkin tiettyyn kehittämiskontekstiin ja sen erityispiirteisiin, ja toinen osa keskittyy ohjeiden sekä viitekehysten antamiseen ketterän menetelmän räätälöimiseksi joko organisaation tai projektin menetelmäksi. Tutkielma tehtiin kirjallisuuskatsauksena.

Ketterät menetelmät ovat olleet 2000–luvulla suosiossa, sillä niiden käyttämisellä on voitu saada monenlaisia hyötyjä. Scrum ja XP ovat tutkituimpia ja suosituimpia ketteriä menetelmiä, mutta eivät ainoita. Menetelmän käyttöönotolle tulee aloite usein johdon tasolta, sillä tuottavuuden on huomattu nousevan käyttämisellä. Hyötyjen saavuttamiseksi on kuitenkin usein tarpeellista räätälöidä menetelmä vastaamaan paremmin kehittämiskontekstia ja sen erityispiirteitä. Erilaisia tilannetekijöitä ja erityispiirteitä ohjelmistokehityksessä on olemassa paljon, ja niiden kaikkien läpikäymistä ei tämän tutkielman puitteissa ole mahdollista suorittaa.

Menetelmän käyttöönotossa niin organisaatio– kuin projektitasolla tulee huomioida, että menetelmä kannattaa pitää ainoastaan lähtökohtana, jolle suorittaa räätälöintiä. Etenkin isoissa organisaatioissa pilottiprojektin läpiviemistä yksittäisellä tai muutamalla kehittäjätiimillä tai kehitysprojektilla voidaan pienentää käyttöönoton riskejä ja tarkkailtua eri käytänteiden, prosessien ja roolien sopivuutta pienemmillä riskeillä ja resursseilla.

Erityispiirteistä ja sovellusalueesta riippuen räätälöintiä voidaan kohdentaa tiettyihin käytäntöihin, rooleihin tai prosesseihin. Ulkoistettujen elektronisen liiketoiminnan projektien tapauksessa käyttöönoton yhteydessä räätälöinti keskittyy projektin hallinnan käytänteisiin ja rooleihin. Ketterää menetelmää räätälöidessä räätälöinnin ei tarvitse myöskään rajoittua ainoastaan menetelmän räätälöintiin. Tämä tuli esille ohjelmistotuoteperheitä tutkittaessa, jossa muokattiin niin ohjelmistotuoteperheiden konsepteja kuin myös Scrumia ketterän ohjelmistotuoteperhekehityksen saavuttamiseksi. Organisaatiot voivat tarvittaessa ostaa ulkopuolisilta toimijoilta käyttöönottoon konsultointia ja kehittäjien koulutusta.

Yksi yleinen lähestymistapa räätälöintiin on valitseva käytäntö, jossa jonkin menetelmän käytänteet käydään läpi yksi kerrallaan, ja otetaan se tai jätetään ottamatta käyttöön, riippuen sen sopivuudesta. Esimerkiksi XP:n käytäntöä, jossa asiakas on paikalla kehittämisessä, on mahdoton soveltaa mikäli asiakasta ei vielä kehittämisvaiheessa ole olemassa. Monissa organisaatioissa on käytössä Scrum projektin hallintaan ja XP:ä teknisiin ratkaisuihin liittyvissä käytänteissä. Scrumin iteratiivinen prosessimalli soveltuu paremmin muokattavaksi kokonaisuutena eikä yksittäisten käytäntöiden valitsemisella. XP:n käytännöt taas vaihtelevat, eivätkä ole niin sidoksissa toisiinsa, jolloin soveltumattomien käytäntöjen pois jättäminen vaikuttaa vähemmän kokonaisuuden toimimiseen.

Kaikki tilanteet yleistäviä johtopäätöksiä räätälöinnistä on tilannetekijöiden kattavuuden takia mahdoton tehdä. Organisaation toteuttamien projektien sovellusalueiden välillä voi olla vaihtelua, ja organisaation käyttämää menetelmää voidaan joutua räätälöimään projektikohtaisesti. Käytetyille käytännöille voidaan suorittaa räätälöintiä, ja näin viedä niitä pidemmälle kuin menetelmän kehittäjien alkuperäinen tarkoitus on ollut, jolloin niistä voidaan saada suurempia hyötyjä.

Tulevaisuudessa ketterien menetelmien käyttö ei oletettavasti tule vähenemään, sillä informaatioteknologian ala on yleisesti kasvussa, ja erilaisia ohjelmistoja sekä järjestelmiä kehitetään jatkuvasti. Liiketoimintaympäristö ja markkinat asettavat myös paineita tuotteiden markkinoille tuloaikojen pienentämiseen ja kehitystyön tuottavuuden parantamiseen. Yksi keino tämän saavuttamiseksi on ketterien menetelmien käyttäminen organisaatioissa ja sisäistämällä sen osaksi organisaation ja projektien jokapäiväistä toimintaa ja organisaatiokulttuuria. Ketterien menetelmien kritiikitön arviointi ja käyttäminen ei kuitenkaan ole tarkoituksenmukaista, vaan on tärkeää säilyttää terve pohdinta sen käyttöä ja räätälöintiä sekä vaihtoehtoisia menetelmiä kohtaan. Tutkielman tuloksia voidaan käytännön työssä hyödyntää räätälöidessä organisaatioon tai projektiin käyttöön ketterää menetelmää. Tällöin voidaan nähdä, millä tavoilla Scrumin ja XP:n prosesseja, rooleja ja käytänteitä on eri tapauksissa räätälöity, ja millaisia tuloksia räätälöinnistä on saatu. Myös räätälöityjen menetelmien eroavaisuudet lähtökohtana olleeseen menetelmään voidaan huomata.

Mahdollisia jatkotutkimusaiheita aihepiirissä on lukuisia. Ketterien menetelmien räätälöintiä tutkivia tapaustutkimuksia on suotavaa tehdä tulevaisuudessa lisää. Tällöin voidaan johdonmukaisesti esittää tutkittavan tapauksen konteksti ja sen erityispiirteet sekä miten ketterä menetelmä räätälöitiin nämä huomioonottaen. Tässä tutkielmassa tutkittiin tarkemmin viittä tapaustutkimusta, mutta jatkotutkimuksella tapaustutkimuksia tutkimalla voisi syvällisemmin saada selville millä tavoin ketteriä menetelmiä on räätälöity organisaatioiden ja projektien käyttöön. Kirjallisuudessa tapaustutkimuksia on olemassa lisää, ja niiden tarkasteleminen on mahdollista myös tässä tutkielmassa esitetyn viitekehyksen avulla. Lisäksi räätälöintiä koskevat tapaustutkimukset tämän tutkielman puitteissa perustuivat ainoastaan Scrumin tai XP:n räätälöintiin, mutta ketteriä menetelmiä on olemassa paljon muitakin. 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.

6 LÄHTEET

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

Agile–allianssi. (2001). Manifesto for Agile Software Development. Haettu 25.2.2013 osoitteesta http://www.agilemanifesto.org/

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

Anderson, D. (2010). *Kanban, Succeessful Evolutionary Change for Your Technology Business*. (1. painos). United States of America: Blue Hole Press.

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

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).

Backlund P. (2002) Identifying situational factors for IS development processes: applying the method–in–action framework. Teoksessa R. D. Banker, H. Chang, Y. C. Kao (toim.), *In Proc. of 8th Americas Conference on Information Systems, Texas, USA, August 9-11 (*s. 1369–1380).

Baskerville, R & Stage, J. (2001). Accommodating Emergent Work Pratices: 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).

Bass, 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).

Beck, K. (1999). *Extreme Programming Explained*. (1. painos). Addison–Wesley Professional.

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

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.

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.

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.

Clarke P., O'Connors R. (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.

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).

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).

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.

Harmsen, F., Brinkkemper, S. & Oei, H. (1994). Situational Method Engineering for Information–System Project Approaches. Teoksessa A. A. Verrijin–Stuart & T. William Olle (toim.), *Methods and Associated Tools for the Information Systems Life Cycle* (s. 169–194).

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.

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

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.), *ICSSP '11 Proceedings of the 2011 International Conference on Software and Systems Process, Honolulu, USA, May 21–22, 2011* (s. 110-119).

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.

Jayawardena, D. S. & Ekanayake L. L. (2010). Adaptation Analysis of Agile Project Management for managing IT projects in Sri Lanka. *Teoksessa Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka, September 29 – October 1, 2010* (s. 1-4).

Karlsson, F. & Ågerfalk, P. (2003). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology*, 46, 619–633.

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).

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

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).

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).

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.

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

Qumer, A. & Henderson–Sellers, B. (2006). 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).

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.

Schwaber, K. & Beedle, M. (2002). *Agile Software Development with Scrum*. (1. painos). Prentice Hall.

Schwaber, K. & Sutherland, J. (2011). *The Official Scrum Rulebook*. Haettu 26.2.2013 osoitteesta
www.scrum.org/Scrum–Guides

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).

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).

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

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

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.

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).

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