Hanna Kuirinlahti

SCRUM- JA XP-KÄYTÄNTEIDEN KÄYTTÖ: HAASTATTELUTUTKIMUS

Tietojärjestelmätieteen
pro gradu –tutkielma
1.11.2011

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Kuirinlahti, Hanna Maria

Scrumin ja XP:n käytänteiden käyttö: haastattelututkimus

Jyväskylä: Jyväskylän yliopisto, 2011

166 s.

Tietojärjestelmätiede, pro gradu –tutkielma

Ohjaaja: Leppänen, Mauri

Tässä tutkielmassa tutkitaan kahden ketterän menetelmän, Scrumin ja XP:n, käytänteiden käyttöä suomalaisissa ohjelmistoalan yrityksissä. Tarkoituksena on selvittää, mitä käytänteitä käytetään ja millä tavalla, millaisia kokemuksia niistä on saatu ja miten kokemukset vertautuvat aiempien tutkimusten tuloksiin.

Tutkimus koostuu kahdesta osasta. Ensimmäisessä osassa on rakennettu työlle ketterää lähestymistapaa, ketteriä menetelmiä ja käytänteitä koskeva käsitteellinen perusta ja esitellään alan aiempia tuloksia. Toinen osa on toteutettu puolistrukturoituna haastattelututkimuksena, johon osallistui kuusi haastateltavaa neljästä suomalaisesta yrityksestä.

Tutkimuksessa todettiin, että aikaisempaa empiiristä tutkimusta yksittäisistä käytänteistä on tehty hyvin vähän. Haastattelututkimuksessa havaittiin, että suurinta osaa tutkituista käytänteistä muokattiin yrityksissä monella tavalla palvelemaan paremmin projektien tavoitteita. Kaikki käsitellyt käytänteet koettiin hyödyllisiksi, vaikka joitakin ongelmiakin joidenkin käytänteiden soveltamisessa havaittiin.

AVAINSANAT: Ketterä lähestymistapa, XP, Scrum, Lean, ketterät käytänteet, käyttökokemukset

ABSTRACT

Kuirinlahti, Hanna Maria

Application of Scrum and XP Practices: Interview Study

Jyväskylä: University of Jyväskylä, 2011

166 p.

Information Systems, Master's Thesis

Supervisor: Leppänen, Mauri

This study considers the application of agile practices of two methods, Scrum and XP, in Finnish software companies. The purpose is to find out which agile practices are deployed, how and with which kind of experience, as well as how these findings compare to those in earlier studies.

The study is composed of two parts. First, a conceptual basis of agile approach, methods and practices is constructed and earlier research is reviewed. Second, a qualitative study based on interviews of six representatives of four Finnish companies is conducted.

It is discovered that previous empirical research on individual practices, in particular, is scarce. The results from the interviews indicate that most of the practices were altered somehow in all the companies to better fit the situational factors of projects. All the practices were seen useful although some challenges and problems in the application were found.

KEYWORDS: Agile approach, XP, Scrum, Lean, Agile practices, usage experience

KUVIOT

KUVIO 1 Tutkijan näkemys Agile-manifestin neljän arvon ja 12 periaatteen keskinäisistä suhteista. 14
KUVIO 2 Leanin, Scrumin ja XP:n painopisteet (Bourne 2010). 15
KUVIO 3 Scrumin prosessimalli (perustuu lähteeseen Abrahamsson, Salo ja Ronkainen (2002), 28). 17
KUVIO 4 Kehämalli XP:n käytänteistä (Lindström & Jeffries 2004, 8). 22
KUVIO 5 XP:n vaiheistus (perustuu lähteeseen Abrahamsson, Salo ja Ronkainen (2002), 19). 23
KUVIO 6 Eri menetelmien periaatteiden ja käytänteiden määrät (Kniberg & Skarin 2010). 31
KUVIO 7 Menetelmien käytön yleisyys (West & Grant 2010, 2). 32
KUVIO 8 Kooste VersionOne:n tekemien kyselytutkimusten tuloksista vuosilta 2006-2010. 33
KUVIO 9 Käytetyimmät ketterät menetelmät vuonna 2010 (VersionOne 2010, 6). 34
KUVIO 10 XP:n käytänteiden keskinäiset suhteet (Lui & Chan 2005, 477) 37
KUVIO 11 XP:n käytänteet käyttöönottojärjestyksessä (Lui & Chan 2005, 480). 38
KUVIO 12 Havainnollistava kuvio menetelmän käyttöönoton ja käytön tasosta yrityksessä (Fichman & Kemerer 1997, 1350-1351) 40
KUVIO 13 Ketterien käytänteiden käytön aste VersionOnen vuoden 2010 kyselytutkimuksessa (VersionOne 2010, 4). 125

TAULUKOT

TAULUKKO 1 Menetelmän käyttöönoton ja käytön taso yrityksessä (Fichman & Kemerer 1997, 1350-1351) 39
TAULUKKO 2 Haastateltavien yrityksissä käytössä olevat käytänteet ja niiden käyttöaste 52
TAULUKKO 3 Haastatteluissa käytetty käytänteiden hyödyllisyysasteikko 55
TAULUKKO 4 Yleiset organisaatioon ja haastateltavan taustaan liittyvät kysymykset ja saadut vastaukset 59
TAULUKKO 5 Käytänteiden omaksumisen taso haastatelluissa yrityksissä 121
TAULUKKO 6 Scrumin ja XP:n käytänteiden hyödyllisyyden arviointi asteikolla -3 (selvästi haittaa) ja 3 (erittäin hyödyllinen). 127

SISÄLTÖ

1 JOHDANTO ... 7
2 KETTERÄ LÄHESTYMISTAPA JA KETTERIÄ MENETELMIÄ ... 11
2.1 Ketterä lähestymistapa ... 11
2.2 Scrum-menetelmä ... 16
2.2.1 Scrumin vaiheistus ... 16
2.2.2 Scrumin roolit ... 18
2.3 Extreme Programming -menetelmä ... 21
2.3.1 XP:n vaiheistus ... 23
2.3.2 XP:n roolit ... 25
2.4 Lean ja Kanban ... 28
2.5 Ketterien menetelmien käytön yleisyys ... 30
2.6 Ketterän lähestymistavan käyttöönotto ... 34
2.7 Yhteenveto ... 40
3 YLEISKATSAUS KETTERÄÄ LÄHESTYMISTAPAA KOSKEVIIN EMPIIRISIIN TUTKIMUKSIIN ... 42
4 HAASTATTELUTUTKIMUKSEN TOTEUTTAMINEN ... 48
4.1 Tutkimusmenetelmän valinta ... 48
4.2 Tutkimuskohteiden valinta ... 49
4.3 Tutkimuksen aihealueen rajaus ja alustava kysely ... 50
4.4 Tiedon keruu ja käsittely ... 53
5 TUTKIMUKSEN TULOKSET ... 58
5.1 Haastateltavien taustatiedot ... 58
5.2 Scrumin käytänteet ... 61
5.2.1 Sprintti ... 61
5.2.2 Tuotteen tehtävälista ... 65
5.2.3 Sprintin suunnittelukokous ... 70
5.2.4 Sprintin tehtävälista ... 75
5.2.5 Päivittäinen Scrum-tapaaminen ... 80
5.2.6 Sprintin katselmointitapaaminen ... 85
5.2.7 Sprintin arviointitapaaminen ... 89
5.3 XP:n käytänteet ... 95
5.3.1 Pariohjelmointi ... 96
5.3.2 Koodin yhteinen omistajuus ... 104
5.3.3 Testivetoinen kehitys ... 108
5.3.4 Jatkuva integraatio ... 113
5.3.5 Jatkuva asiakkaan läsnäolo ... 116
5.4 Ketterien menetelmien käyttöönoton taso haastateltavissa yrityksissä ... 120

5.5 Yhteenveto ... 123

6 DISKUSSIO ... 124

6.1 Ketterien käytänteiden tunnettuus ja käytön aste ... 124
6.2 Scrumin ja XP:n käytänteiden hyödyllisyys ... 126
6.3 Käytänteiden muokkaus ... 129
6.4 Scrumin ja XP:n käytänteiden väliset suhteet ... 130
6.5 Tutkimuksen reliabiliteetti ja validiteetti ... 132

7 YHTEENVETO ... 135
LIITE 1 ALUSTAVA KYSELY ... 154
LIITE 2 HAASTATTELURUNKO ... 155
LIITE 3 SCRUMIN TARKISTUSLISTA ... 161
LIITE 4 TIIVISTELMÄ TULOKSISTA ... 162

1 JOHDANTO

Tänä vuonna tulee kuluneeksi 10 vuotta siitä, kun ketterän lähestymistavan ”kivijalka”, Agile-manifesti (Agile Alliance 2001), julkaistiin. Ketterän lähestymistavan mukaiset menetelmät ja käytänteet ovat manifestin julkaisemisen jälkeen ehtineet olla käytössä laajalti, ja ne ovat vuosi vuodelta saavuttaneet enemmän jalansijaa ohjelmistoalan projekteissa (VersionOne 2010).

Ohjelmiston kehittämismenetelmä on ketterä, jos se muovautuu ja muuttuu nopeasti asiakkaiden tai muun ympäristön vaatimusten mukaisesti, tukee muutosten tekemistä, auttaa oppimaan muutoksen tuomista haasteista ja hyödyntämään niitä tulevissa haasteissa (Conboy 2009). Sujuva muutosten tekeminen on mahdollista tiimin ja asiakkaan tiiviin yhteistyön ja lyhyin väliajoin tapahtuvan toimivan ohjelmiston toimittamisen ansiosta (Abrahamsson, Salo & Ronkainen 2002). Ketteryyden uskotaan mm. tuovan läpinäkyvyyttä ja avoimuutta asiakkaan ja ohjelmistotoimittajan välille, selkeyttävän tiimin sisäistä tehtävänjakoa, auttavan paremman ohjelmiston suunnittelussa, ja nopeuttavan ohjelmiston valmistumista (Dybå & Dingsøyr 2008).

Ketteriä menetelmiä on lukuisia, esimerkiksi Scrum (Schwaber & Beedle 2002), eXtreme Programming, (XP) (Beck 1999b) ja DSDM (Stapleton 1997) sekä Lean ohjelmistokehitys (Poppendieck & Poppendieck 2003) ja Kanban (Kniberg 2011). Menetelmät eroavat toisistaan monella tavalla (Abrahamsson, Salo & Ronkainen 2002). Esimerkiksi Scrum antaa työkaluja projektin ketterään hallintaan eivätkä sen opit juurikaan ota kantaa koodaustyön tekemiseen. XP antaa konkreettisempia ohjeita itse kehitystyöhön. (Abrahamsson ym. 2002, Bourne 2010)

Ketteriä menetelmiä sovelletaan käytännössä organisaatiosta ja projektista riippuen varsin eri tavoin. Tietystä menetelmästä on voitu ottaa käyttöön vain joitakin käytänteitä, tai käytetty menetelmä on voitu ”rakentaa” useammasta menetelmästä poimituista käytänteistä. Myös tavat, joilla käytänteitä sovelletaan, vaihtelevat yritys- ja projektikohtaisesti.

Ketteristä menetelmistä ja käytänteistä saatavia hyötyjä ja niistä mahdollisesti aiheutuvia haittoja on selvitetty aiemmissa tutkimuksissa (Dybå & Dingsøyr 2008). Tieteellistä tutkimusta ketteristä menetelmistä on tehty paljon, mutta sen laatu ja laajuus eivät vastaa kuitenkaan sitä, millainen merkitys ketterillä menetelmillä on nykypäivän ohjelmistoalalla (Dybå & Dingsøyr 2008, Sjøberg, Dybå & Jørgensen 2007). Eniten tutkittuja ketteriä käytänteitä ovat pariohjelmointi (esim. Cockburn & Williams 2001, Arisholm, Gallis, Dybå & Sjøberg 2007 ja Auvinen, Back, Heidenberg ym. 2005) ja testivetoinen kehitys (esim. Begel & Nagappan 2007, Tessem 2003 ja Siniaalto & Abrahamsson 2007), mutta niiden hyödyntämisen aste ei ole yhtä suuri kuin esimerkiksi Scrumiin liittyvillä käytänteillä (vrt. VersionOne 2010).

Suomessa ketteriä käytänteitä koskevia empiirisiä tutkimuksia on jonkin verran tehty (esim. Abrahamsson ym. 2005, Ikonen 2010, Kettunen ym. 2005, Komi-Sirviö ym. 2005, Korhonen 2009, Korkala ym. 2006, Kähkönen 2004, Paasivaara ym. 2008, Pikkarainen ym. 2006). Useimmissa näistä tutkimuksen kohteena ovat olleet kuitenkin yksittäiset menetelmät tai käytänteet tai tietynlainen konteksti (esim. hajautettu kehittäminen). Näissä tutkimuksissa ei ole katettu laajasti ketteriä käytänteitä.

Tämän tutkimuksen tarkoituksena on kartoittaa ketterien käytänteiden käyttöä suomalaisissa ohjelmistoalan yrityksissä. Koska jo pelkästään ketteriä menetelmiä on olemassa lukuisia, puhumattakaan kaikista erilaisista ketteriin menetel-

miin liittyvistä käytänteistä, on käsiteltävien käytänteiden joukkoa rajattu. Menetelmistä rajaudutaan tarkastelemaan pääasiassa kahta suosituinta menetelmää, Scrumia (Schwaber & Beedle 2002) ja XP:tä (Beck 1999b).

Tutkimuksen tutkimusongelmana on:

> Mitä ketteriä käytänteitä ja millä tavoin käytetään suomalaisissa ohjelmistoalan yrityksissä, millaisia kokemuksia niistä on saatu ja miten kokemukset vertautuvat aiempien tutkimusten tuloksiin?

Tutkimus on toteutettu pääasiassa kvalitatiivisena haastattelututkimuksena. Ensimmäisessä osassa rakennetaan ketterää lähestymistapaa sekä ketteriä menetelmiä ja käytänteitä koskevaa käsitteellistä perustaa. Lisäksi esitellään aiempien empiiristen tutkimusten tuloksia. Toisessa osassa toteutetaan haastatteluja neljässä suomalaisessa ohjelmistoalan yrityksessä. Ennen haastatteluja osallistujille lähetetään kysely siitä, mitä käytänteitä kyseisissä yrityksissä käytetään. Tämän mukaan rajaudutaan tarkastelemaan niitä käytäntöjä, joista haastatteluun osallistuvilla on eniten henkilökohtaista käytännön kokemusta.

Tutkimuksen tuloksista on monenlaista hyötyä. Ne tarjoavat ensinnäkin ajankohtaisen tilannekuvan ketterien käytänteiden soveltamisen yleisyydestä ja tavasta. Toiseksi ohjelmistoalan yritykset voivat oppia muiden yritysten tavoista soveltaa käytänteitä ja käytänteiden käyttökokemuksista, olivatpa ne sitten positiivisia tai negatiivisia. Kolmanneksi tehty tutkimus tarjoaa pohjan jatkotutkimukselle syvällisemmän käsityksen saamiseksi tiettyjen käytänteiden käytöstä ja tämän tutkimuksen ulkopuolelle jääneiden käytänteiden tutkimiselle.

Tutkielma on jäsennetty seitsemään lukuun. Seuraavassa luvussa esitellään ketterä lähestymistapa ja ketteristä menetelmistä lähemmin Scrum- ja XP-menetelmät. Luvussa 3 esitetään katsaus ketterän lähestymistavan soveltamista koskeviin empiirisiin tutkimuksiin ja ketterän lähestymistavan tutkimuksen tilaan yleisesti. Luvussa 4 kuvataan tutkielman empiirisen osan tutkimusmenetelmä, jonka jälkeen luvussa 5 esitellään käytänne kerrallaan kirjallisuuskatsauksen ja oman empiirisen tutkimuksen tulokset. Luvussa 6 esitetään yhteenveto ja johtopäätökset. Viimeinen luku kokoaa tutkielman tärkeimmät asiat.

2 KETTERÄ LÄHESTYMISTAPA JA KETTERIÄ MENETELMIÄ

Tämän luvun tarkoituksena on kuvata lyhyesti ketterä lähestymistapa ja esitellä yleisimpiä ketteriä menetelmiä. Luvun aluksi kuvataan ketterien menetelmien taustaa ja perusperiaatteita Agile-manifestin mukaisesti. Sen jälleen esitellään yleisimmät ketterät menetelmät, Scrum ja XP (eXtreme Programming) sekä lyhyesti myös Lean ja Kanban. Tarkoituksena on antaa tiivistetty yleiskuva ohjelmistotuotannossa suosituimmista ja laajimmin käytetyistä ketteristä menetelmistä. Menetelmien kuvausten jälkeen kerrotaan tehtyihin tutkimuksiin nojaten, miten yleistä menetelmien käyttö on. Luvun lopuksi esitellään keinoja, joilla voidaan arvioida ketterän menetelmän käytön laajuutta yrityksessä.

2.1 Ketterä lähestymistapa

Perinteisempien prosessimallien, kuten vesiputousmallin (engl. waterfall) (Royce 1970), iteratiivisen kehityksen (engl. iterative development) (Larman & Basili 2003), RUP:n (engl. Rational Unified Process) (Kroll & Kruchten 2003) tai Spriaalimallin (engl. Spiral) (Boehm 1988) mukaiset ohjelmistoprojektit toteutetaan etukäteen määritellyn ja suunnitellun vaiheistuksen mukaan siitä mitenkään poikkeamatta (West & Grant 2010). Ohjelmistoprojektit ovat kuitenkin harvoin ennustettavissa, joten niiden aikana käytettävien menetelmien tulisi muuttua projektin mukaan eikä projektien menetelmän mukaan. Ketteriksi määritellyissä projekteissa suunnitelmia ei lyödä lukkoon koko projektin loppuun asti, vaan kehittäjät keskittyvät enemmän kuuntelemaan asiakkaan muuttuvia toiveita ja tarpeita. Ketterissä menetelmissä asiakkaalle pyritään myös toimittamaan toimivaa ohjelmistoa mahdollisimman varhaisessa vaiheessa ja usein (West & Grant 2010). Menetelmät ovat usein hyväksi havaittujen

työtapojen kokoelmia (nk. best practices), joiden pilkuntarkka noudattaminen tai kaikkien käytänteiden hyödyntäminen ei ole oleellista.

Molemmilla lähestymistavoilla on hyvät ja huonot puolensa (Boehm & Turner 2003). Joissakin yrityksissä jo pelkästään projektien sopimusmalli määrittää käytettävän menetelmän. Jos rahoitusta saadaan projektin aikana vain tiettyjen ennalta määrättyjen vaiheiden loputtua, voi tämä sopimusmalli olla aivan liian byrokraattinen vapaammin tilanteen mukaan muokkautuvalle projektille (Grossman, Bergin, Leip ym. 2004). Tässä tutkielmassa keskitytään tähän vapaampaan lähestymistapaan, jota yleisesti kutsutaan ketteräksi lähestymistavaksi (engl. Agile approach).

Kevyemmät ohjelmistokehityksen menetelmät alkoivat kiinnostaa ohjelmistotuotannon ammattilaisia 1990-luvun loppupuolella (Larman & Basili 2003). Myöhemmin näitä kevyitä menetelmiä alettiin kutsua ketteriksi menetelmiksi (engl. agile methods). Vuonna 2001 joukko eri ketteriksi luokiteltavien menetelmien (XP, Scrum, DSDM, Crystal, ym.) edustajia kokoontui yhteen tarkoituksenaan saada jotain vastapainoa raskaammille ohjelmistojen kehitysmalleille ja tuoda ketteryyden ideologia esille selkeämmin (Agile Alliance 2001). Tapaaminen tuottikin tulosta ja sen seurauksena julkaistiin Agile-manifesti (engl. Agile Manifesto), jossa tiivistetään ketterän lähestymistavan oleellisin ydin ketterän lähestymistavan neljään arvoon (Agile Alliance 2001):

- Yksilöitä ja heidän välistä vuorovaikutusta arvostetaan prosesseja ja työkaluja enemmän.
- Toimiva ohjelmisto on tärkeämpi kuin kattava dokumentaatio.
- Asiakasyhteistyö on tärkeämpää kuin sopimusneuvottelut.
- Muutokseen reagoimista arvostetaan enemmän kuin suunnitelman noudattamista.

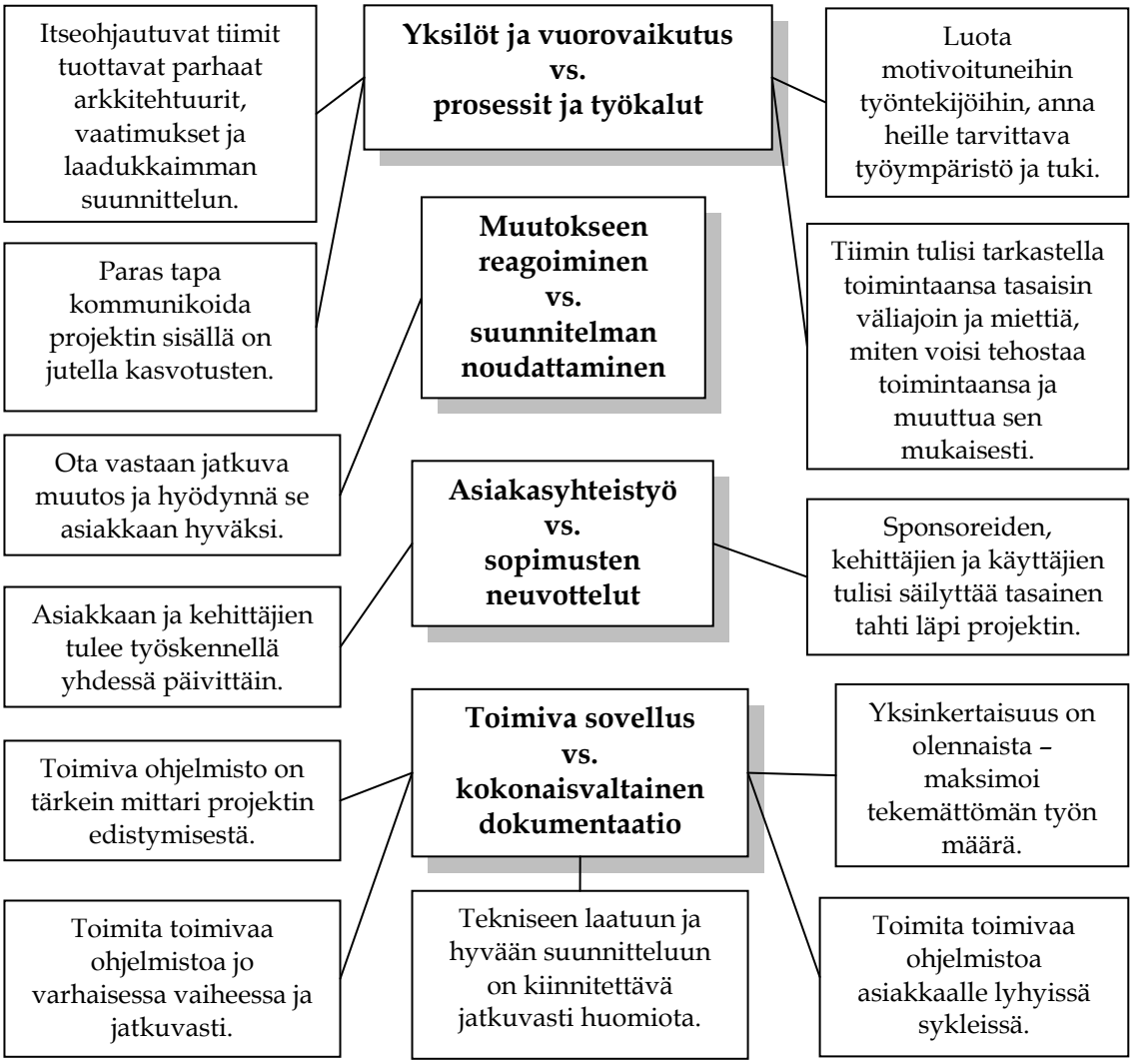
Manifestin ensimmäinen periaate korostaa ohjelmistokehityksen inhimillistä puolta, jossa kehittäjien yhteistyöllä ja tiimityöllä on todella merkitystä työnteon sujuvuuteen. Toimivan ohjelmiston periaate auttaa kehittäjiä keskittymään, koska heidän täytyy saada toimitettua asiakkaalle toimivaa ja testattua ohjelmistoa säännöllisin, lyhyin väliajoin. Tällöin kehittäjät keskittyvät todella tuottamaan tarpeeksi yksinkertaista, mutta toimivaa koodia, jolloin tarvittavan dokumentaation määräkin vähenee. Yhteistyö asiakkaan kanssa takaa sen, että ketterässä projektissa asiakas saa todella sitä, mistä hänelle on sillä hetkellä eniten hyötyä. Tämä ei silti korvaa sopimusten ja vaatimusmäärittelyjen tekoa asiakkaan kanssa, periaatteen hienous on asiakkaan ja kehittäjien tiiviissä yhteistyössä. Viimeinen periaate tarkoittaa sekä asiakkaan että kehittäjien asennoitumista siihen, että projektin edetessä projektiin voidaan ja tullaankin tekemään monenlaisia muutoksia. (Abrahamsson, Salo & Ronkainen 2002, 11-12)

Nämä neljä arvoa tarkentuvat 12 periaatteeseen (Agile Alliance 2001). Kuviossa 1 on esitetty kirjoittajan oma näkemys neljän perusarvon ja 12 periaatteen keskinäisistä suhteista.

Agile-manifesti ei suoraan määrittele ketterää lähestymistapaa. Kirjallisuudessa on sen sijaan esitetty hyvin monenlaisia määritelmiä. Yhteisenä tekijänä näille määritelmille on, että ne hakevat lähtökohtansa manifestin arvoista ja periaatteista. Perusteellisimman tarkastelun käsitteen merkityksestä on tehnyt Conboy (2009). Hän on tehnyt laajan, useampaa tieteenalaa koskevan kirjallisuusanalyysin ja todennut, että ketteryys perustuu kahteen peruskäsitteeseen, joustavuuteen (engl. flexibility) and ”hoikkuuteen” (engl. leanness). Hän on johtanut ketterälle menetelmälle määritelmän, joka ottaa huomioon monipuolisesti siihen sisältyviä moninaisia merkityksiä. Määritelmä kuuluu seuraavasti (Conboy 2009, 340):

> ”[Ketteryys tarkoittaa] kehittämismenetelmän jatkuvaa valmiutta nopeasti tai luontaisesti luoda muutosta, ennakoivasti tai reagoivasti tukea muutosta sekä oppia muutoksesta samalla kuin se lisää asiakkaan saamaa arvoa (tuottavuus, laatu ja yksinkertaisuus) yhteisten [menetelmä]-komponenttien ja ympäristösuhteiden avulla. ”

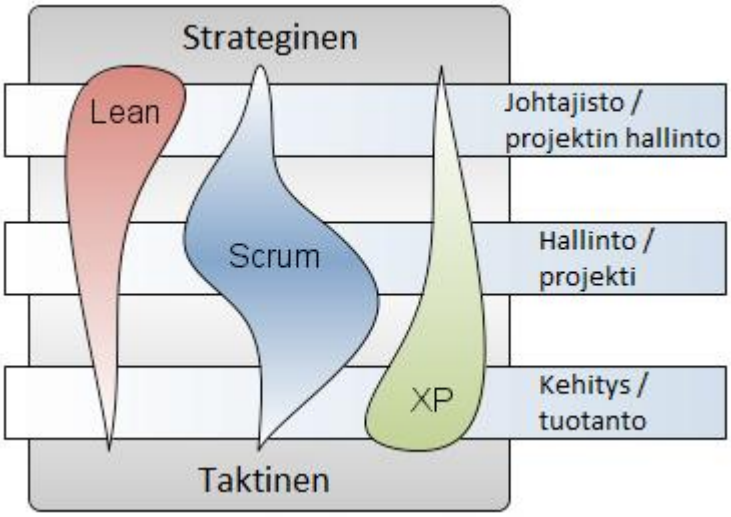
Määritelmässä korostuvat muutokset, joihin ketterän kehittämisen tulee ennakoivasti ja reagoivasti pystyä vastaamaan, sekä lisäarvo, jonka kehittämisen tulee tuottaa asiakkaalle.



KUVIO 1 Tutkijan näkemys Agile-manifestin neljän arvon ja 12 periaatteen keskinäisistä suhteista.

Ketteriksi luokiteltavia menetelmiä on lukuisia, esimerkiksi Abrahamsson, Salo ja Ronkainen (2002) listaa ketteriksi menetelmiksi seuraavat: eXtreme Programming (XP) (Beck 1999b), Scrum (Schwaber & Beedle 2002), Crystal family of methodologies (Cockburn 2002), Feature Driven Development (FDD) (Palmer & Felsing 2002), Rational Unified Process (RUP) (Kruchten 1996), Dynamic Systems Development Method (DSDM) (Stapleton 1997), Adaptive Software Development (Highsmith 2000) ja Open Source Software Development (O'Reilly 1999).

Näistä suosituimpia ovat Scrum, XP ja Scrumin ja XP:n yhdistelmä (VersionOne 2010). Viime vuosina myös Leanin (Poppendieck & Poppendieck 2003) ja Kanbanin (Kniberg 2011) suosio on kasvanut. Scrum tarjoaa vahvan tuen tiimitasoiselle projektinhallinnalle, kun taas XP sisältää käytännön kehittämistyötä tukevia käytänteitä (Abrahamsson, Salo & Ronkainen 2002). Tästä syystä menetelmien yhdistäminen onkin yleistä projekteissa. Lean-periaatteet ovat yleisluonteisempia, ja ne tarjoavat tukea myös yrityksen johdolle. Bourne (2010) havainnollistaa eroja menetelmien painopisteissä kuvion 2 avulla.



KUVIO 2 Leanin, Scrumin ja XP:n painopisteet (Bourne 2010).

Seuraavissa alaluvuissa kuvataan tarkemmin menetelmistä Scrum ja XP, sekä lyhyesti myös Lean ja Kanban. Scrumin ja XP:n osalta käydään läpi tässä vaiheessa vain menetelmien perusrakenne ja roolit. Menetelmiin kuuluvat yksittäiset käytänteet esitellään yksityiskohtaisemmin luvussa 5.

2.2 Scrum-menetelmä

Scrum terminä esiintyi kirjallisuudessa ensimmäisen kerran jo vuonna 1986, jolloin Hirotaka Takeuchi ja Ikujiro Nonaka käsittelivät pienten tiimikokojen hyötyä tuotekehityksessä (Nonaka & Takeuchi 1986). Tästä ideasta hioutui myös ohjelmistotuotannon tarpeet huomioon ottava Scrum (Schwaber 1995). Scrumin alkuperäinen perusperiaate on muodostaa sellaisia ohjelmistotuotannon tiimejä, joissa kaikki tiimin jäsenet ovat jatkuvasti tietoisia siitä, missä yleisesti projektissa mennään (Schwaber 1995). Scrumiin kuuluvat käytänteet ovat seuraavat (Sutherland & Schwaber 2007):

- sprintti (engl. sprint)
- tuotteen tehtävälista (engl. product backlog)
- sprintin suunnittelukokous (engl. sprint planning meeting)
- sprintin tehtävälista (engl. sprint backlog)
- päivittäinen Scrum-tapaaminen (engl. daily stand-up meeting)
- sprintin katselmointitapaaminen (engl. sprint review)
- sprintin arviointitapaaminen (engl. sprint retrospective)
- työmäärän arviointi (engl. effort estimation)

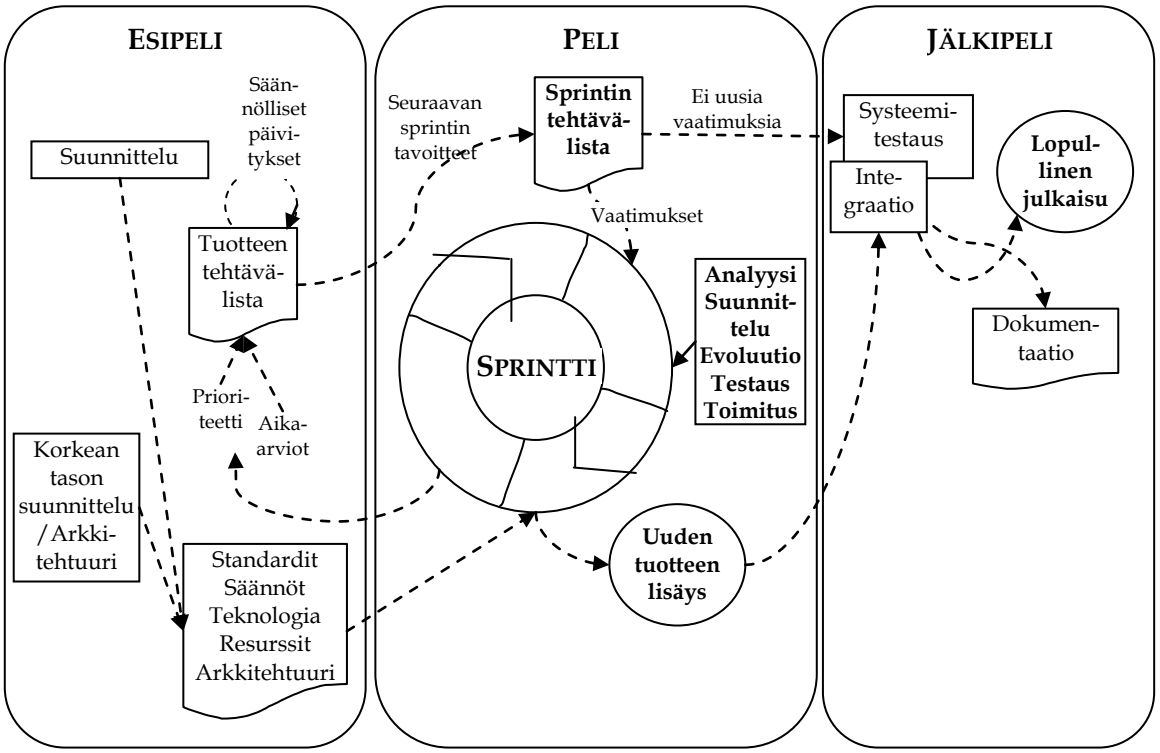
Scrumia sovelletaan tänä päivänä monissa yrityksissä hyvinkin eri tavoin, mutta seuraavissa alakohdissa kuvataan Scrumin vaiheistus ja tarkastellaan Scrumin rooleja perustuen seuraaviin lähteisiin: Abrahamsson, Salo ja Ronkainen (2002), Schwaber (1995), Schwaber ja Beedle (2002), Schwaber ja Sutherland (2010) sekä Sutherland ja Schwaber (2007).

2.2.1 Scrumin vaiheistus

Menetelmän erikoinen nimi tulee alun perin rugby-nimisen pelin aloitusmuodostelmasta (engl. Scrum). Schwaber (1995) jaotteleekin artikkelis-

saan Scrumin kolme vaihetta pelin henkeen sopivalla tyylillä: esipeli, peli ja jälkipeli. Prosessin kuvaus on esitetty tarkemmin kuviossa 3.

Esipeli (engl. pregame) pitää sisällään esimerkiksi aikataulutuksen ja hinnoittelun alustavan määrittelyn sekä järjestelmän arkkitehtuurin suunnittelun. Uutta järjestelmää suunniteltaessa selvitetään sitä koskevat standardit, säännöt ja käytettävä teknologia sekä analysoidaan resurssien käyttöä ja arkkitehtuuria. Esipelin aikana mietitään myös projektitiimin kokoonpano ja tarpeet (Abrahamsson, Salo & Ronkainen 2002). Tuotteen tehtävälistaan kootaan asiakkaan haluamien toiminnallisuuksien kuvauksia prioriteetti- ja aika-arvioineen. (Schwaber 1995)



KUVIO 3 Scrumin prosessimalli (perustuu lähteeseen Abrahamsson, Salo ja Ronkainen (2002), 28).

Peli (engl. game) kuvaa Scrumin iteratiivista vaihetta, niin kutsuttua sprinttivaihetta, jossa on itsessään neljä osaa: kehitysvaihe, kasausvaihe, tarkasteluvaihe

sekä sopeutumisvaihe (Schwaber 1995). Jokaisen uuden sprintin alkaessa asiakas ja Scrum-tiimi miettivät yhdessä tulevan sprintin tavoitteen, jonka perusteella tuotteen tehtävälistasta valitaan tavoitteen täyttymisen varmistavat tehtävät (Schwaber & Beedle 2002). Kyseiseen sprinttiin mukaan valitut tehtävät jaetaan vielä pienempiin osiin ja jokaisen tehtävänpalasen aika-arviota pyritään tarkentamaan entisestään (Schwaber & Beedle 2002). Nämä hienojakoiset tehtävät muodostavat sprintin tehtävälistan (Schwaber & Beedle 2002). Sprintin tehtävälistan tehtävien muuntaminen toimivaksi ohjelmistoksi on tiimin omalla vastuulla, jolloin työskentelytapana yritys ja erehdyskään ei ole mitenkään kiellettyä (Marcal, De Freitas, Soares ym. 2008). Sprinttivaihe kestää tyypillisesti viikosta neljään viikkoon (Schwaber 1995).

Jälkipelivaiheessa (engl. postgame) tehdään viimeistään järjestelmälle integraatio- ja systeemitestaukset, jotta voidaan olla varmoja ohjelmiston toimivuudesta kokonaisuutena ennen sen lopullista julkaisua. Sen aikana tuotetaan myös käyttöönotossa mahdollisesti tarvittavat koulutusmateriaalit, käyttöoppaat, markkinointimateriaalit sekä muut tärkeät järjestelmän toimintaan liittyvät dokumentit (Schwaber 1995).

2.2.2 Scrumin roolit

Scrumissa projektiin osallistujien roolit ovat hyvin olennainen osa menetelmää. Scrumia hyödyntävän projektin parissa työskentelee projektihallintoon liittyviä ihmisiä sekä kehitykseen keskittyvä Scrum-tiimi, joiden tärkeimmät yksittäiset roolit on Scrumissa määritelty hyvin tarkasti (Schwaber 1995). Tässä esitellään lyhyesti kolme kehitykseen liittyvää roolia, Scrum-mestari, tuotteen omistaja ja Scrum-tiimi, sekä projektiin liittyvät roolit asiakas ja johto, joiden toiminta ja tehtävä on hyvä olla selvillä ennen kuin siirrytään tarkastelemaan yksittäisiä käytänteitä.

Scrum-mestari

Ehkä tärkeimpänä projektin hallinnoimiseen liittyvänä jäsenenä voidaan nähdä *Scrum-mestari* (engl. Scrum Master), jonka tehtävänä on käytännössä pitää huoli menetelmän oikeaoppisesta toteuttamisesta (Schwaber & Beedle 2002). Mutta se, että Scrum-tiimi tuntee työskentelytapansa ja osaa toimia sen mukaisesti, ei riitä. Scrum-mestarin haastavaan työnkuvaan kuuluu myös projektin asiakkaiden (lähinnä kuitenkin tuotteen omistajan) ohjaaminen ja neuvominen Scrum-tiimin kanssa työskentelyyn, jotta näiden kahden tahon työskentely saman tavoitteen eteen olisi mahdollisimman suoraa ja saumatonta (Schwaber 2004). Scrum-mestari pyrkii varmistamaan tiimille mahdollisimman optimaaliset työskentelyolosuhteet puuttumalla työtä hidastaviin asioihin sekä tekemällä tiimin puolesta nopeaa päätöksentekoa vaativat päätökset (Schwaber & Beedle 2002). Vaikka Scrum-mestarin roolissa saattaa usein toimia projektipäällikkö, on huomattava, että Scrum-mestarin rooliin ei kuulu tehtävien jakaminen tiimille, vaan nimenomaan sen varmistaminen, että tiimi olisi mahdollisimman hyvin itseorganisoituva (Sutherland & Schwaber 2007).

Tuotteen omistaja

Tuotteen omistaja (engl. Product Owner) on vastuussa projektin rahoituksen hankkimisesta ja siitä huolehtimisesta, että näille rahoille saadaan todellista vastinetta projektin edetessä. Hänen vastuullaan on tuoda parhaansa mukaan esille eri osapuolten toiveet kehitettävän ohjelmiston suhteen. Tuotteen omistaja on ensisijainen asiakkaan edustaja projektissa, joka määrää viime kädessä mm. tuotteen tehtävälistan sisällön tärkeysjärjestyksen. (Schwaber & Beedle 2002, Schwaber 2004)

Scrum-tiimi

Scrum-tiimin perimmäinen tehtävä on luonnollisesti toteuttaa asiakkaan vaatimusten mukainen ohjelmisto. Scrumissa tiimi on koko projektin kriittisin

tekijä. Tiimille annetaan muihin menetelmiin verrattuna enemmän valtaa päättää siitä, miten kehitteillä oleva ohjelmisto toteutetaan. Itse asiassa Scrum-tiimi saa olla koko kehitysjakson, eli sprintin ajan täysin rauhassa, ja sillä on täysi valta toteuttaa annetut tehtävät juuri niin kuin se itse parhaaksi näkee (Schwaber & Beedle 2002). Tällä pyritään valjastamaan tiimin sisäinen tietotaito projektin hyväksi mahdollisimman tehokkaasti, koska vapaiden käsien lisäksi tiimi saa kontolleen myös vastuun toteutettavasta työstä (Schwaber & Beedle 2002). Tiimin rooliin kuuluu konkreettisen koodaustyön lisäksi myös kehitysideoiden tuominen ja esittäminen esimerkiksi tuotteen omistajalle, jotta tuotteesta saataisiin mahdollisimman hyvin asiakkaan tarpeita vastaava (Sutherland & Schwaber 2007). Tiimikooksi Schwaber ja Beedle (2002) suosittelevat alle kahdeksaa henkilöä. Jos projektin laajuus vaatii enemmän kehittäjiä, voidaan yhteen projektiin perustaa useampia Scrum-tiimejä (Schwaber & Beedle 2002).

Asiakas

Asiakkaan rooli on Scrumissa tärkeä, koska asiakkaalta vaaditaan Scrumia hyödyntävässä projektissa huomattavasti enemmän sitoutumista ja yhteistyötä kuin esimerkiksi perinteisemmässä vesiputousmallin mukaisessa projektissa. Asiakas osallistuu Scrumissa aktiivisesti tuotteen kehitykseen myös sen kehityksen aikana, esimerkiksi tuotteen tehtävälistan koostamiseen, eikä ole mukana esimerkiksi vain vaatimusmäärittelyn tekemisen aikana. (Abrahamsson, Salo & Ronkainen 2002)

Johto

Johdon edustajat ovat mukana Scrumin prosessissa luonnollisesti päätöksentekijän roolissa, mutta myös tarvittaessa Scrum-tiimin tukena ja apuna. Erona totuttuun johdon rooliin Scrum tuo sen haasteen, että johdon edustajien tulisi Scrumissa ennemminkin johdatella tiimiä kohti ratkaisua kuin päättää asioista

itsenäisesti ja ojentaa sen valmiina tiimille tehtäväksi. (Sutherland & Schwaber 2007)

2.3 Extreme Programming -menetelmä

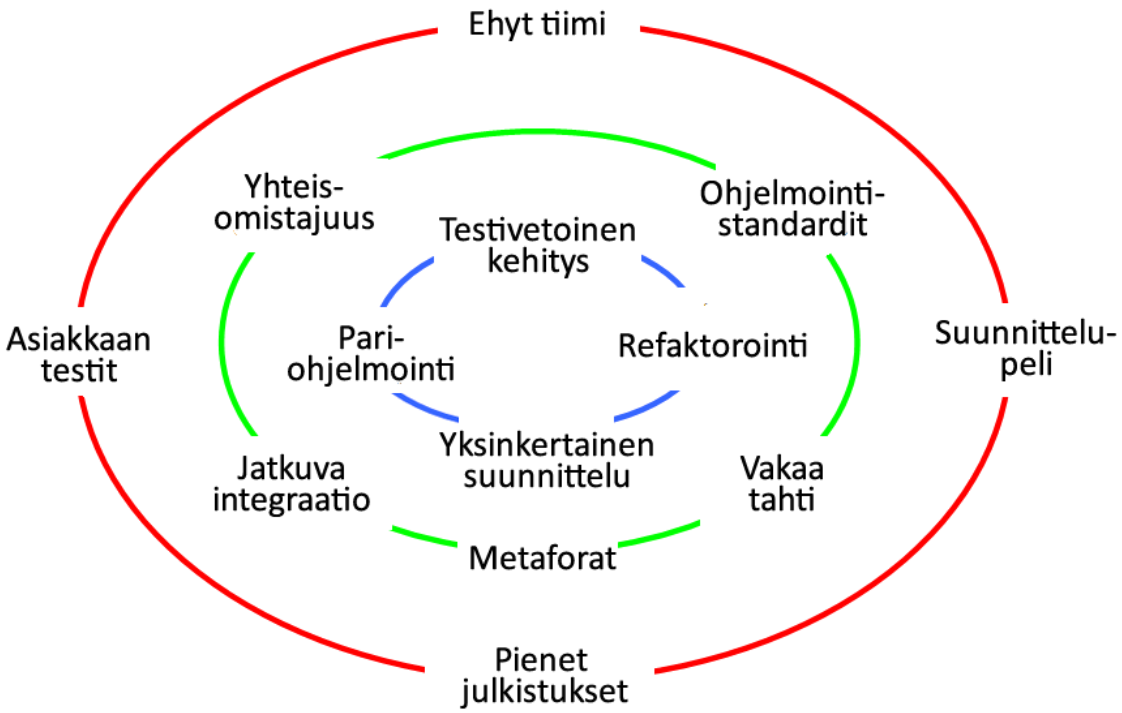
Extreme Programming tunnetaan myös lyhyemmin nimellä XP (Beck 1999b). Kent Beck esitteli menetelmän perusperiaatteet kirjassaan ”Extreme Programming Explained: Embracing Change” vuonna 1999 (Beck 1999a). Poiketen hieman muista aiemmin esitellyistä menetelmistä, XP:hen kuuluu kokoelma hyvinkin konkreettisia ohjeita eli käytänteitä (engl. practices), jotka koskettavat menetelmistä eniten ohjelmoijan päivittäistä työtä. Nämä 13 käytännettä ovat (Beck 1999a):

- pariohjelmointi (engl. pair programming)
- koodin yhteinen omistajuus (engl. collective ownership)
- testivetoinen kehitys (engl. test-driven development, TDD)
- jatkuva integraatio (engl. continuous integration)
- refaktorointi (engl. refactoring)
- suunnittelupeli (engl. planning game)
- pienet julkistukset (engl. small releases)
- metaforat (engl. metaphor)
- yksinkertainen suunnittelu (engl. simple design)
- jatkuva asiakkaan läsnäolo (engl. on-site customer)
- vain säännöt (engl. just rules)
- 40 tunnin työviikko (engl. 40-hour weeks)
- avoin työtila (engl. open workspace)

Monissa lähteissä listan viimeistä kohtaa, avointa työtilaa, ei ole laskettu mukaan XP:n käytänteisiin. Yksittäisten käytänteiden sisältö esitellään tarkemmin alaluvussa 5.3.

Kirjallisuudessa on esitetty lukuisia tapoja XP:n käytäntöjen kategorioimiseksi ja niiden välisten suhteiden osoittamiseksi (esim. Beck 1999b, Lindström & Jeffries 2004, Mirakhorli, Khanipour Rad, Shams ym. 2008). Esimerkkinä näistä on kuviossa 4 esitetty XP:n 12 käytännettä (osittain hieman eri termein kuin yllä

esitetyssä listassa) kehämallin avulla (Lindström & Jeffries 2004). Sisimmällä kehällä on esitetty XP:n kehittäjiä koskevat käytänteet, keskimmäisellä kehällä ovat koodin laatua ylläpitävät käytänteet sekä lopulta ulkokehällä on esitetty kehittäjien ja asiakkaan välisen kommunikaation välineet.



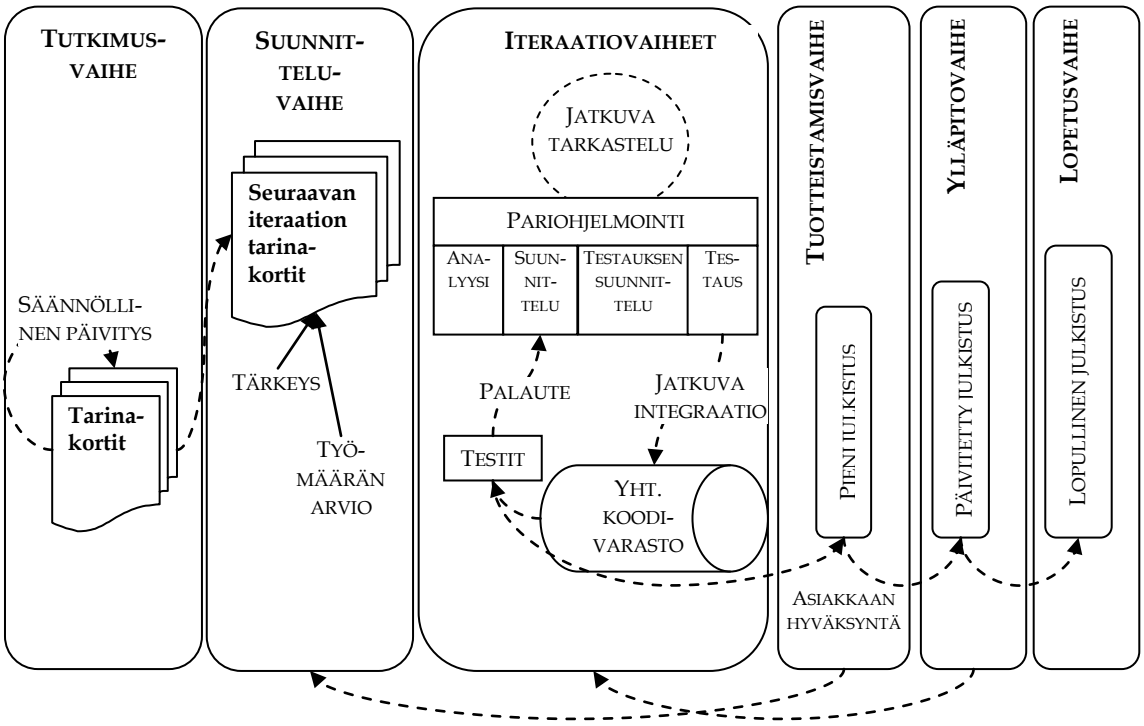
KUVIO 4 Kehämalli XP:n käytänteistä (Lindström & Jeffries 2004, 8)

Mirakhorlin ym. (2008) mielestä osa XP:n käytänteistä on selitetty hieman ympäripyöreästi, mistä syystä XP:tä voi toteuttaa ilman, että hyödyntää jokaista siihen kuuluvaa käytännettä. Siksi he luokittelevat XP:n säännöt kahteen kastiin: sitoutumisen säännöt (engl. rules of engagement) ja toiminnan säännöt (engl. rules of play). Sitoutumisen säännöt kertovat, millainen ympäristö antaa puitteet XP:n tehokkaaseen hyödyntämiseen, ja toiminnan säännöt keskittyvät tarkemmin kuvailemaan työn etenemisen vaiheita.

Seuraavissa alaluvuissa kuvataan XP:n vaiheistus ja XP:ssä määriteltyjä rooleja. Tarkastelu perustuu seuraaviin lähteisiin: Abrahamsson, Salo ja Ronkainen (2002), Beck (1999a) sekä Beck (1999b).

2.3.1 XP:n vaiheistus

Beckin (1999b) mukaan XP:ssä ohjelmistoprosessi jakautuu kuuteen vaiheeseen, jotka ovat tutkimus, suunnittelu, iteraatio, tuotteistaminen, ylläpito ja lopetus. Vaiheiden yhteydet toisiinsa näkyvät kuviosta 5.



KUVIO 5 XP:n vaiheistus (perustuu lähteeseen Abrahamsson, Salo ja Ronkainen (2002), 19).

Tutkimusvaiheessa (engl. exploration phase) tutustutaan yleisesti kohdealueen toimintatapoihin sekä mietitään asiakkaan kanssa, mitä asioita halutaan ohjelmistoon toteuttaa (Abrahamsson, Salo & Ronkainen 2002). Asiakas pääsee tässä totuttelemaan sopivien tarinakorttien (engl. story cards) kirjoittamiseen samalla kun XP-tiimi antaa kirjoitetuista tarinakorteista palautetta asiakkaalle ja arvioi samalla kunkin tarinakortin toteutuksen kestoa. Tiimin tärkein tehtävä on kuitenkin tutustua käytettävään teknologiaan ja miettiä kehitettävää ohjelmistoa arkkitehtuurin tasolla. Kokeneen tiimin kanssa tutkimusvaiheen ei tarvitse Beckin (1999b) mukaan kestää välttämättä muutamaa viikkoa pidem-

pään, joten kun tarinakortteja on saatu kirjoitettua ja niiden kestoa arvioitua tarpeeksi tarkalla tasolla, voidaan luontevasti siirtyä jo oikean tekemisen pariin. Tarinakorttien muokkaamista ja päivitystä jatketaan kuitenkin säännöllisesti seuraavienkin vaiheiden aikana. (Beck 1999b)

Suunnitteluvaiheessa (engl. planning phase) pohditaan asiakkaan kanssa, mitä toiminnallisuuksia halutaan toteuttaa ensimmäisessä julkaisussa (engl. release) (Abrahamsson, Salo & Ronkainen 2002). Halutun tuotteen kannalta keskeisimmät ja siksi ensimmäisenä toteutettavaksi valitut tarinakortit otetaan erilleen ja niiden tärkeysjärjestystä ja työmäärää arvioidaan tarkemmalla tasolla (Beck 1999b).

Iteraatiovaiheita (engl. iterations to release) on XP:ssä useita. Ensimmäisissä iteraatioissa keskitytään toteuttamaan sellaisia tehtäviä, jotka tukevat järjestelmän rungon muodostumista. Asiakas päättää kunkin iteraation alussa, mitkä määritellyistä tehtävistä toteutetaan sen iteraation aikana. Iteraation lopuksi suoritetaan asiakkaan määrittelemät toiminnalliset testit, joiden onnistuttua kehitetty ohjelmiston osa julkaistaan. Jos toteutettavia, asiakkaan mielestä tärkeitä tehtäviä on vielä julkaisun jälkeen tekemättä, jatketaan seuraavaan iteraatioon. Kukin iteraatio kestää keskimäärin yhdestä neljään viikkoon. (Beck 1999b)

Tuotteistamisvaiheessa (engl. productionizing phase) ohjelmistoa testataan entistä tarkemmin, jotta nähdään, soveltuuko kehitetty ohjelmisto tuotantoon. Tässä vaiheessa ei keskitytä uusien toiminnallisuuksien tekemiseen, vaan jokaisen lisäysehdotuksen kohdalla pohditaan tarkkaan, kannattaako sitä ottaa mukaan juuri tähän pieneen julkistukseen. Tästä syystä iteraatioiden kestoa saatetaan tarvittaessa lyhentää viikkoon. (Beck 1999b)

Ylläpito- ja lopetusvaiheet (engl. maintenance and death phases): Ylläpitovaiheessa ohjelmisto on jo julkaistu asiakkaan käyttöön. Olemassa olevan ohjelmiston ylläpidon lisäksi XP-projektin tehtäviin kuuluu usein vielä uusien toiminnallisuuksien toteuttaminen. Kun asiakas on päättänyt, ettei lisää toiminnallisuuksia tai päivityksiä enää toteuteta, ollaan lopetusvaiheessa. Koska muutoksia ei enää tällöin tehdä, on loppudokumentaation vuoro. (Beck 1999b)

2.3.2 XP:n roolit

XP:ssä projekteissa käytettävien tiimien kooksi on suositeltu alle 10 ihmistä, jotta tiimin sisäinen kommunikaatio pysyisi sujuvana (Beck 1999a). Tiimiin kuuluvilla ihmisillä voi olla useita erilaisia rooleja, esimerkiksi sama ihminen voi toimia vaikkapa ohjelmoijan ja testaajan roolissa. Enemmän projektin johtamiseen liittyviä rooleja ovat valmentajan (engl. coach) ja jäljittäjän (engl. tracker) roolit (Beck 1999b). Seuraavaksi kuvataan kaikki XP:ssä eritellyt roolit ja rooleihin kuuluvat vastuualueet.

Ohjelmoija

Ohjelmoija on luonnollisesti tärkeimmässä roolissa myös XP:ssä, koska hän toteuttaa teknisesti asiakkaan toiveet. Beck mainitsee artikkelissaan kuitenkin muutamia erityispiirteitä, joita XP:tä hyödyntävässä projektissa työskentelevällä ohjelmoijalla tulee olla teknisen osaamisen lisäksi. XP edellyttää kaikilta osapuolilta hyvää kommunikointia, mutta erityisesti yhden käytänteen, pariohjelmoinnin, toimivuus ja tuloksellisuus riippuu täysin ohjelmoijien kyvystä kommunikoida keskenään luonnollisesti ja mahdollisimman avoimesti. Yhteistyön tärkeys tulee esille myös muissa käytänteissä. XP:ssä kaikki kirjoitettava ohjelmakoodi on yhteistä, jolloin ohjelmointitiimin jäsenten on kyettävä ohjelmoimaan yhteisten pelisääntöjen mukaan ja kestettävä sitä, että heidänkin kirjoittamaansa koodia voidaan muuttaa. (Beck 1999b)

Testaaja

Testaajan roolin vastuualueisiin kuuluu asiakkaan auttaminen testitapausten kirjoittamisessa sekä testien ajaminen ja niistä raportointi tarvittaville tahoille. Testaajan toimenkuvaan kuuluu myös huolehtia siitä, että ohjelmisto läpäisee aina kaikki aikaisemmin kirjoitetut testit. Tällä varmistetaan, ettei ohjelmiston kehityksen edetessä oteta takapakkia ja vahingossa rikota jotakin vanhempaa toiminnallisuutta. Testaaja ei tee vain ja ainoastaan testaajan työtä, koska esimerkiksi yksikkötestauksesta huolehtivat ohjelmoijat. (Beck 1999b)

Asiakas

Asiakkaan rooli XP:ssä on yhtä vaativa kuin ohjelmoijankin. Asiakkaan on osattava esittää ideansa toteutettavasta ohjelmistosta tarinakorttien avulla, mikä voi olla alussa asiakkaalle hyvin vaikeakin tehtävä. Myös muutoksenhallinta on asiakkaan harteilla. Asiakkaan on tehtävä nopeita päätöksiä projektin edetessä iteraation sisällä ja iteraatioiden välillä. Esimerkiksi jos aikataulullisesti on mahdotonta toteuttaa kaikki meneillä olevaan iteraatioon valitut toiminnallisuudet, on asiakkaan pystyttävä kertomaan nopeasti kehitystiimille, mitä toiminnallisuutta voidaan karsia ja mitä voidaan mahdollisesti jättää iteraatiosta kokonaan pois. XP:ssä asiakkaan tehtäviin kuuluu myös toiminnallisten testien kirjoittaminen, mikä voi kehitettävästä järjestelmästä riippuen olla hyvinkin vaikeaa. Jokaisessa XP-projektissa on kuitenkin määritelty projektiryhmään yksi testaaja, jonka vastuulla on asiakkaan auttaminen ja ohjaaminen testien kirjoittamisessa. Pyrkimyksenä on kirjoittaa sellaisia testejä, joiden avulla asiakas voi varmistaa itselleen kehitettävän ohjelmiston riittävän laadun. (Beck 1999b)

Valmentaja

Valmentajan (engl. coach) rooli kuuluu projektissa useimmiten jollekin hieman kokeneemmalle henkilölle. Beck (1999b) määrittelee valmentajan vastuualuei-

siin koko prosessin seuraamisen. Valmentajan tärkein tehtävä on seurata ohjelmiston valmistumisprosessia ja puuttua siihen, jos joitakin epäkohtia ilmenee. Tarkoituksena olisi, että valmentajan tarvitsisi puuttua asioihin hyvin harvoin, ja tarpeenkin vaatiessa mahdollisimman epäsuorasti. Epäsuoralla asioihin puuttumisella tarkoitetaan sitä, että valmentaja tuo epäkohdat tiimin jäsenten tietoon sellaisella tavalla, että tiimi näkee itsekin, mikä on menossa pieleen ja osaa itse korjata ongelmansa. Valmentajan roolin merkitys pienenee sitä myötä, mitä kokeneempi XP-tiimi on ohjattavana. (Beck 1999b)

Jäljittäjä

Beck (1999b) kuvailee *jäljittäjää* (engl. tracker) koko tiimin omaksitunnoksi, jonka vastuualueeseen kuuluu koko projektin ja sen iteraatioiden etenemisen seuraaminen ja niistä kertominen tiimin jäsenille. Jäljittäjän tehtäviin kuuluu historiatietojen kerääminen esimerkiksi tiimin jäsenten tehtävien suunnitelluista ja toteutuneista tunneista. Tällöin jäljittäjän huomioiden perusteella tiimi pystyy tarkentamaan seuraavissa iteraatioissa suoritettavien yksittäisten tehtävien aika-arviota aiempaa paremmin. Projektin tarkkailu laajemmasta näkökulmasta taas auttaa tiimin jäseniä hahmottamaan suorituskykynsä rajat, jolloin iteraatioihin valitaan sopivan kokoisia tehtäviä ja asiakkaalle seuraavaan julkaisuun luvatut toiminnallisuudet pystytään myös toteuttamaan. Jäljittäjän kirjanpito pitää sisällään myös testaukseen liittyviä tunnuslukuja, kuten esimerkiksi ohjelmasta ilmoitetut virheet ja kenen vastuulla ne ovat. Mitattavia ja raportoitavia asioita on paljon, joten jäljittäjän on oltava tarkkana, ettei datan kerääminen tai tiimin ohjaaminen oikealle polulle häiritse itse tiimin työskentelyä, vaan tukee ja parantaa sitä. (Beck 1999b)

Konsultti

Konsultin tehtävä on tuoda lisäpanostusta silloin, kun XP-tiimissä ei aina ole tarpeeksi ohjelmistoarkkitehtejä, testaajia tai muihin ohjelmistokehityksen osa-

alueisiin erikoistuneita ihmisiä. Projektiin voidaan tällöin ottaa mukaan konsultti, jonka tehtävänä on ratkaista kyseessä oleva ongelma. Konsulttia vaativat ongelmat ovat usein kriittisiä ohjelmiston valmistumisen kannalta ja sellaisia, jotka tiimin on hyvä hallita jatkossakin. Tästä syystä usein muutama tiimin jäsenistä istuu konsultin vierellä koko sen ajan, kun hän ratkoo ongelmaa, jolloin tiimi oppii ongelman ratkaisusta toivottavasti niin paljon, että se osaa seuraavan kerran ratkaista vastaavan ongelman itsenäisesti. (Beck 1999b)

Iso Pomo

Iso Pomo (engl. Big Boss) mainitaan Beckin (1999b) kirjassa lähinnä siksi, että sellainenkin johtotason henkilö, joka ei ole aikaisemmin työskennellyt XP:tä hyödyntävän projektin parissa, tietäisi varautua XP:n muista menetelmistä poikkeavaan työskentelytapaan. (Beck 1999b) Ison Pomon vastuualueita ei ole sen kummemmin eritelty, joten tätä roolia ei käsitellä tämän enempää.

2.4 Lean ja Kanban

Leanin perusajatukset on alun perin kehitetty autoteollisuutta (Honda, Toyota) varten 1980-luvun Japanissa. 1990-luvulla ne levisivät yleisemmin tuotevalmistukseen ja sieltä muillekin aloille. Periaatteet pohjautuvat ymmärrykseen siitä, mikä luo arvoa asiakkaalle, miksi nopea valmistusvirta on tärkeää ja kuinka vapauttaa ihmisten luova voimavara. Nämä ajatukset ovat universaaleja, ja niitä voidaan soveltaa hyvin monenlaisissa tilanteissa. (Poppendieck & Poppendieck 2003)

Poppendieck ja Poppendieck (2003) muunsivat Leanin perusajatukset ohjelmistotuotantoon sopiviksi periaatteiksi. Pääperiaatteita on seitsemän:

- vältä jätettä (engl. waste)
- tue oppimista
- päätä mahdollisimman myöhään
- toimita mahdollisimman nopeasti

- voimaannuta tiimiä
- varmista laatu
- näe kokonaisuus

Näistä yhtenä tärkeimmistä pidetään ensimmäistä periaatetta. Jätteeksi nähdään kaikki sellainen työ, joka ei lisää tuotteen tai palvelun arvoa asiakkaalle. Ohjelmistotuotannossa jätettä voivat aiheuttaa toteutetut mutta tarpeettomat tuoteominaisuudet, asiakkaiden ja työntekijöiden odottaminen, turhat siirrot ja siirtymiset, turhat prosessit, keskeneräinen työ, huomaamatta jääneet virheet, sekä vastuun vaihto. (Leffingwell 2007b, Poppendieck & Poppendieck 2003)

Poppendieck (2002) on tiivistänyt Lean periaatteet neljään kohtaan seuraavasti:

- Tee vain sellaista, joka tuottaa lisäarvoa (karsi jätettä).
- Keskity ihmisiin, jotka tuottavat lisäarvoa.
- Lisäarvo on seurausta kysynnästä (lykkää sitoutumista).
- Optimoi yli organisaatioiden.

Joissakin tilanteissa Scrum-menetelmän vaatimus sprinteistä jaksoina, joiden aikana kenenkään tiimiin kuulumattoman ei sovi puuttua sprintin alussa päätettyyn sprintin tehtävälistaan, on koettu hankalaksi. Ympäristö voi olla niin dynaaminen, että muutoksia tehtävälistaan joudutaan tekemään hyvinkin pienellä varotusajalla. Tällaisia tilanteita varten sopii paremmin Kanban, jossa ohjelmistokehitys nähdään pikemminkin katkeamattomana kehittämistehtävien virtana kuin määräpituisina sprintteinä (Anderson 2010).

Kanbanin keskeisiä ohjeita on käytännössä vain kolme kappaletta, jotka ovat (Kniberg & Skarin 2010)

- havainnollista työn etenemistä (engl. visualize your workflow)
- rajoita meneillään olevien tehtävien määrää (engl. limit work in progress, WIP)
- mittaa työtehtävien läpimenoaikaa (engl. measure the lead time).

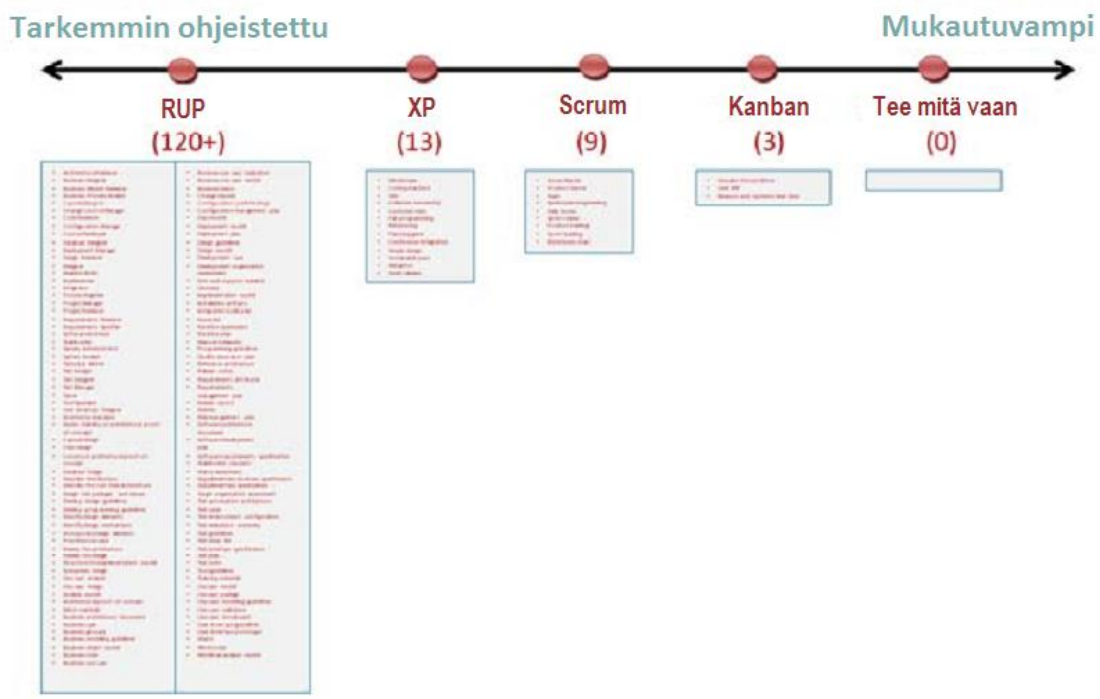
Kanbanissa meneillään olevan työn havainnollistaminen on tärkeää, ja havainnollistamisen tapa on samankaltainen Scrumin kanssa. Yksinkertaisimmillaan tehtävien etenemistä projektissa havainnollistetaan Kanbanissa tarralapuilla, jotka liikkuvat tehtävien tilanteen mukaan ”tekemättä – työn alla – tehty” -sarakkeissa työtilan seinällä olevassa taulussa. Scrumiin nähden yksi suurimmista eroista on se, että yhtäaikaisesti työn alla tietyssä vaiheessa ei voi olla kuin tietty määrä tehtäviä. Työskentely on tällöin liukuhihnamaisempaa kuin Scrumissa, jossa työtehtävien määrää on rajoitettu vain sprintin aikana. Kolmas sääntö perustuu siihen, että optimoidut ja mahdollisimman pieniksi palasiksi pilkotut tehtävät ovat helpommin ennustettavissa, jolloin se auttaa kokonaisprosessin suunnittelussa. Yksittäisten työtehtävien läpimenoaikojen optimointi, WIP-periaatteen rinnalla, pienentää tehokkaasti ”pullonkaulojen” syntymisen riskiä. (Kniberg & Skarin 2010)

Kanbanin mukaista ohjelmistotuotantoa ohjeistetaan hyvin pienellä määrällä periaatteita. Kuviossa 6 on verrattu Kanbanin ohjeistusta RUP:n (Kroll & Kruchten 2003), XP:n (Beck 1999b) ja Scrumin (Schwaber & Beedle 2002) ohjeistuksiin. Kuten kuviosta nähdään, Kanbania voidaan pitää liki ”Tee mitä vaan” –tyyppisenä toimintana. Käytännössä Kanbania käytetään kuitenkin useimmiten Scrumin kanssa luopuen joistakin sprinttejä koskevista käytänteistä. Tällöin menetelmästä käytetään nimitystä Scrumban (Ladas 2009).

2.5 Ketterien menetelmien käytön yleisyys

Ketterien menetelmien käytön yleisyydestä on vaikea saada luotettavaa tietoa. Ensinnäkin edustavan joukon saaminen vastaamaan on hyvin haasteellista, jopa mahdotonta, ja toiseksi eri maissa ja yrityksissä on aika lailla erilainen käsityksiä siitä, mitä milläkin menetelmällä tarkoitetaan. Usein yritykset ovat tehneet

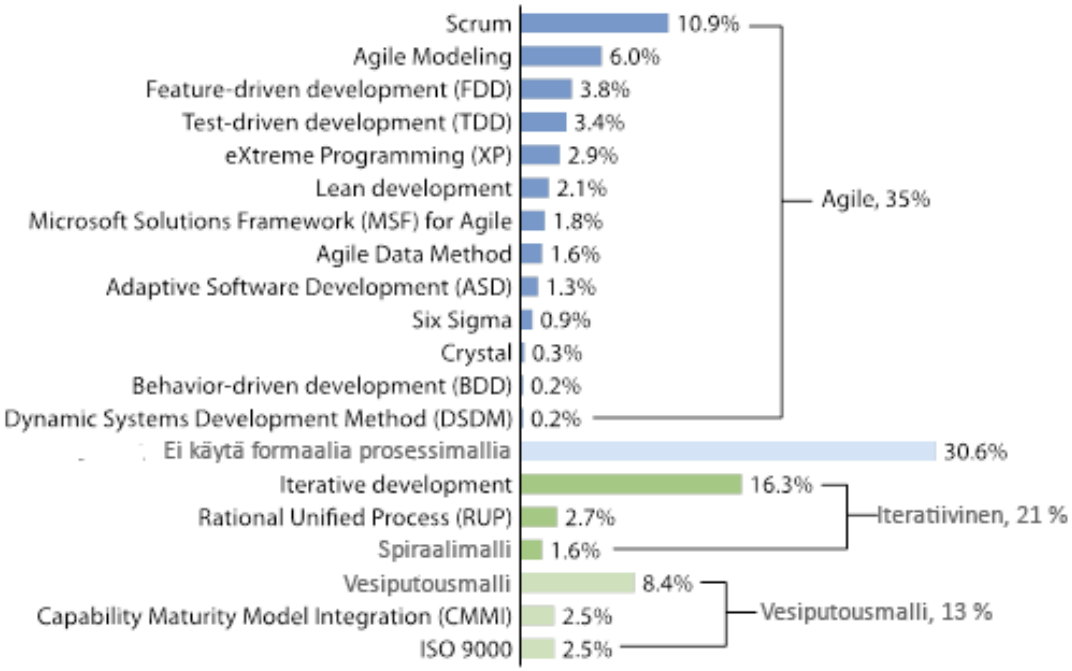
myös omia räätälöintejään menetelmiin ja yhdistäneet useampienkin menetelmien piirteitä omiinsa.



KUVIO 6 Eri menetelmien periaatteiden ja käytänteiden määrät (Kniberg & Skarin 2010).

Jotta tätä tutkimusta varten saataisiin kuitenkin jonkinlainen käsitys menetelmien yleisyydestä, kuvataan tässä kahta ei-akateemista tutkimusta. Ensimmäinen näistä on Westin ja Grantin (2010) vuonna 2009 netin kautta tehty kyselytutkimus, johon vastasi lähes 1300 ohjelmistokehityksen ammattilaista. Kysymys kuului: "Valitse yksi menetelmä, joka kuvaa parhaiten tällä hetkellä käyttämäänne kehitysprosessia". Vastausten perusteella yleisimpiä menetelmiä olivat Scrum (Schwaber & Beedle 2002), Agile Modeling (Ambler 2002), Feature-driven development (Palmer & Felsing 2002), testivetoinen kehitys (engl. Test-driven development), Extreme programming (Beck 1999b) ja Lean-ohjelmistokehitys (Poppendieck & Poppendieck 2003). Yksittäisenä menetelmänä kaikista laajimmin oli käytössä ei-ketteräksi luokiteltava iteratiivinen

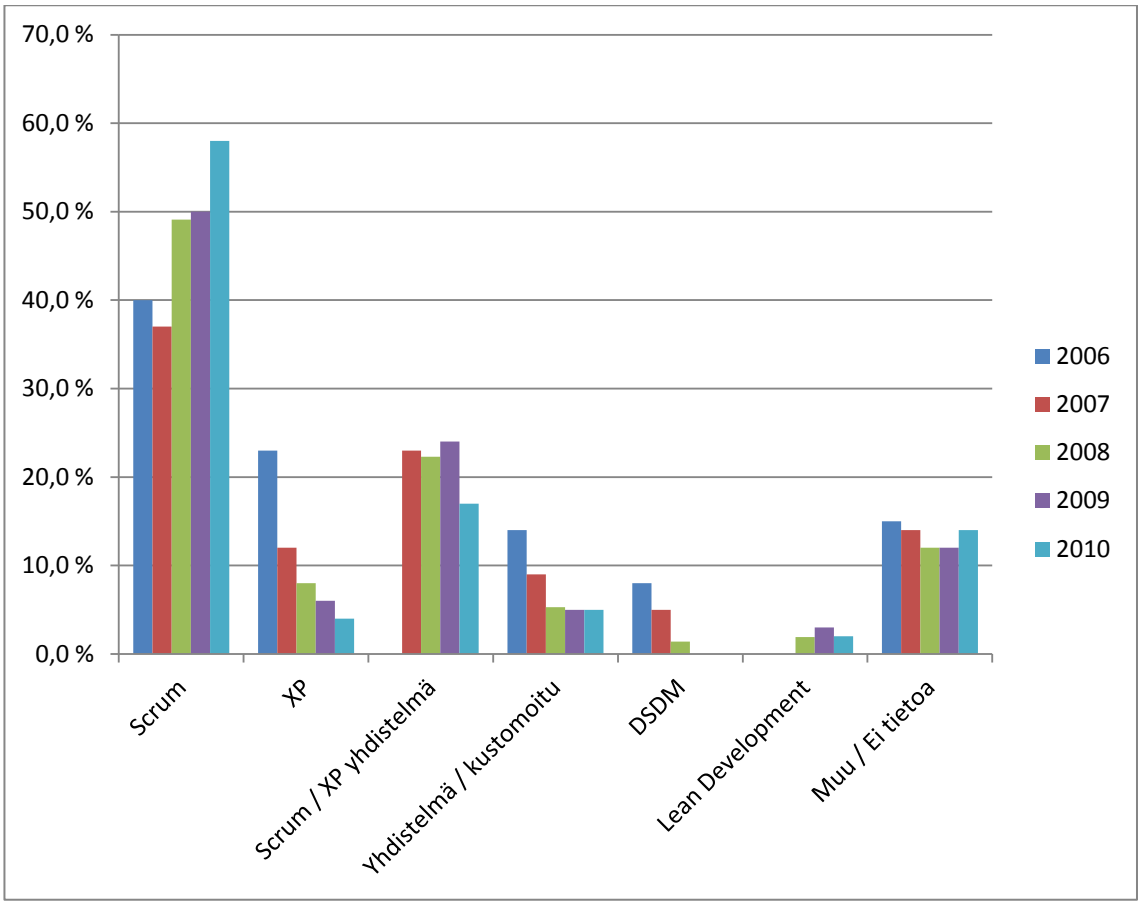
kehitys, jota hyödynsi yli 16 % vastaajista, mutta heti toisella sijalla oli Scrum, jota hyödynsi yli 10 % kyselyyn vastanneista (KUVIO 7).



KUVIO 7 Menetelmien käytön yleisyys (West & Grant 2010, 2)

Toinen tutkimus perustuu VersionOne-nimisen yrityksen nettikyselyyn. VersionOne on projektinhallintaohjelmia valmistava yritys, joka on vuosittain lähettänyt asiakkailleen ketteriä menetelmiä ja käytäntöjä koskevan kyselyn. Kyselytutkimuksiin on osallistunut 722 (v. 2006), 1700 (v. 2007), 2319 (v. 2008), 2570 (v. 2009) ja 4770 (v. 2010) ohjelmistotuotannon alalla toimivaa henkilöä (VersionOne 2006, 2007, 2008, 2009, 2010). Kuviossa 8 on esitetty kooste vuosien 2006 - 2010 aikana toteutettujen kyselytutkimusten tuloksista. Jos jokin menetelmä on mainittu kyselytutkimuksessa vain yhtenä tai kahtena vuotena, menetelmän prosenttiosuudet on laskettu mukaan ”Muu / Ei tietoa” –ryhmään. Jos tutkimustuloksissa ei mainita muuta kuin suosituimpien menetelmien prosenttiosuudet, myös loput prosentit on laskettu mukaan ”Muu / Ei tietoa” –ryhmään (vuodet 2006, 2007 ja 2008). Tuloksista nähdään, että Scrum on ollut

selvästi suosituin yksittäinen ketterä menetelmä viimeisen viiden vuoden aikana ja se on kasvattanut suosiotaan vuosi vuodelta. XP:n osalta käyttöaste on vuosittain laskenut, mutta Scrumin ja XP:n yhdistelmä on pitänyt vuoteen 2010 saakka pintansa suhteellisen tasaisesti.

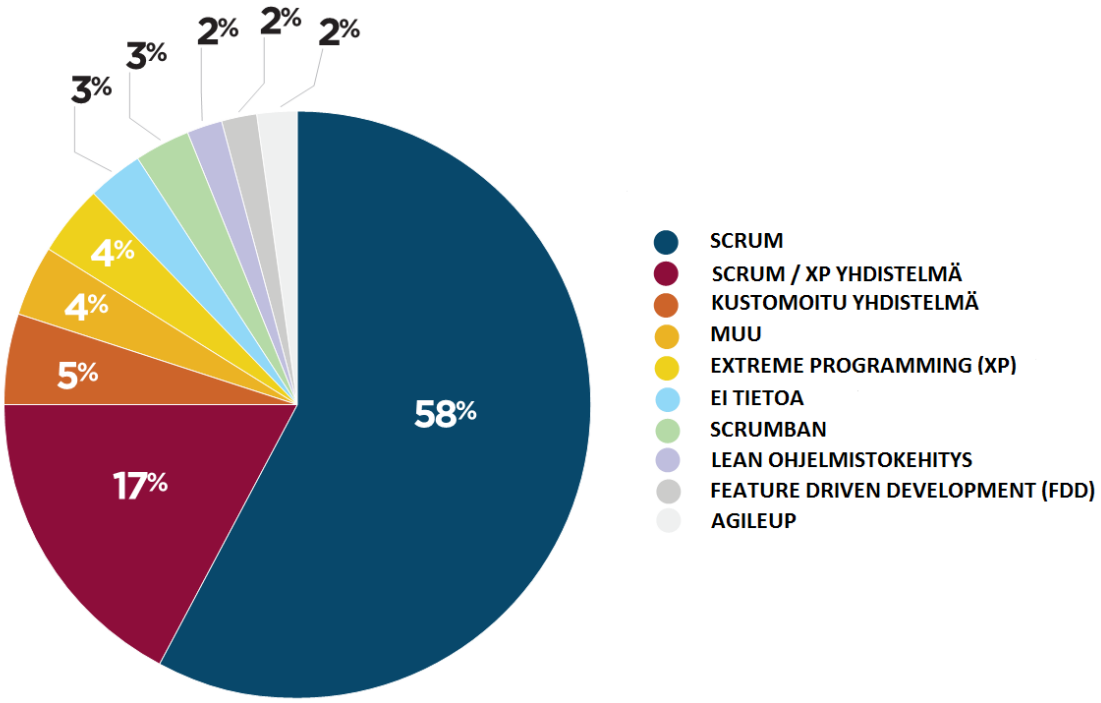


KUVIO 8 Kooste VersionOne:n tekemien kyselytutkimusten tuloksista vuosilta 2006-2010.

Scrumin ja XP:n ja niiden yhdistelmien lisäksi ketteristä menetelmistä on ollut pidemmällä aikavälillä käytössä vain kaksi muuta menetelmää, DSDM (lyhenne sanoista Dynamic Systems Development Method) (Stapleton 1997) ja Lean-ohjelmistokehitys (Poppendieck & Poppendieck 2003). DSDM:n kuuluu yhdeksän perusperiaatetta, jotka ovat hyvin lähellä Agile-manifestin periaat-

teita (Leffingwell 2007b). Menetelmä on kuitenkin menettänyt suosiotaan vuosi vuodelta ja vuoden 2009 tutkimuksessa sitä ei enää mainittu erikseen.

Kuviolla 9 havainnollistetaan vielä menetelmien käytön osuuksia vuonna 2010 (viimeisin tutkimus). Kuvio perustuu vastauksiin, joita on annettu kysymykseen: ”Mainitse ketterä menetelmä, jota lähinnä noudatatte” (VersionOne 2010). Kuviosta nähdään, että Scrum oli ylivoimaisesti suurin yksittäinen vuonna 2010 hyödynnetty ketterä menetelmä. Scrum sekä Scrumin ja jonkun muun ketterän menetelmän yhdistelmä kattaa lähes 80 % vastauksista (VersionOne 2010).



KUVIO 9 Käytetyimmät ketterät menetelmät vuonna 2010 (VersionOne 2010, 6).

2.6 Ketterän lähestymistavan käyttöönotto

Ketterän lähestymistavan mukaisia menetelmiä ja käytäntöjä voidaan ottaa yrityksissä käyttöön hyvin monella tavalla. Ääritapoina ovat inkrementaalinen periaate ja kaikki kerralla, eli ”one shot” –periaate (Rohunen ym. 2010). Edellisessä tapauksessa ketteriä käytänteitä otetaan käyttöön vähitellen. Jälkimmäi-

sessä tapauksessa käyttöönotto pyritään tekemään yhdellä kertaa. Käyttöönottoa tukemaan ja käyttötapojen arviointiin on kehitetty testejä (esim. Sutherland 2010), viitekehyksiä (esim. Qumer & Henderson-Sellers 2008, Sidky, Arthur & Bohner 2007) ja kypsyysmalleja (esim. Ambler 2010, Packlick 2007, Patel & Ramachandran 2009). Näiden avulla voidaan selvittää, miten pitkällä ketterän menetelmän käyttöönotossa ollaan tai kuinka hyvin käyttöönotto on onnistunut yrityksessä. Seuraavaksi esitellään muutamia testejä ja malleja. Ensin kuvataan kolmea testiä, joiden kohteena on Scrumin käyttö. Sen jälkeen esitetään tiekarttamainen malli (Lui & Chan 2005), jossa XP:n käytännöt on asetettu hierarkkisille tasoille sen mukaan, missä järjestyksessä käytäntöjä luontevasti voi ottaa käyttöön. Kolmanneksi esitetään yleisluontoisempi, menetelmästä riippumaton käyttöönoton asteikko, jota hyödynnetään myöhemmin esiteltävässä haastattelututkimuksessa.

Scrumin käyttöönoton arviointi

Scrumin käyttöönoton arviointiin erikoistuneita asteikkoja ja mittareita on olemassa useita. Näistä Nokia-testi (AgileCollab 2011) testaa Scrumin käyttöä muutaman yksinkertaisen kysymyksen avulla. Kysymykset on luokiteltu kahteen tasoon, joista ensimmäinen liittyy ketterään työskentelyyn yleensä ja toinen taso Scrumiin (AgileCollab 2011). Tasot ja niihin liittyvät kysymykset ovat:

Taso 1:

- Suunnitellaanko iteraatiot siten, että ne kestävät alle kuusi viikkoa?
- Onko ohjelmisto täysin testattu ja toimiva jokaisen iteraation lopussa?
- Voiko iteraatio alkaa ennen kuin vaatimusmäärittely on valmis?

Taso 2:

- Tietääkö tiimi, kuka on tuotteen omistaja?
- Onko tuotteen tehtävälista priorisoitu asiakkaan saaman hyödyn mukaan?
- Arvioiko tiimi tuotteen tehtävälistan tehtävät?

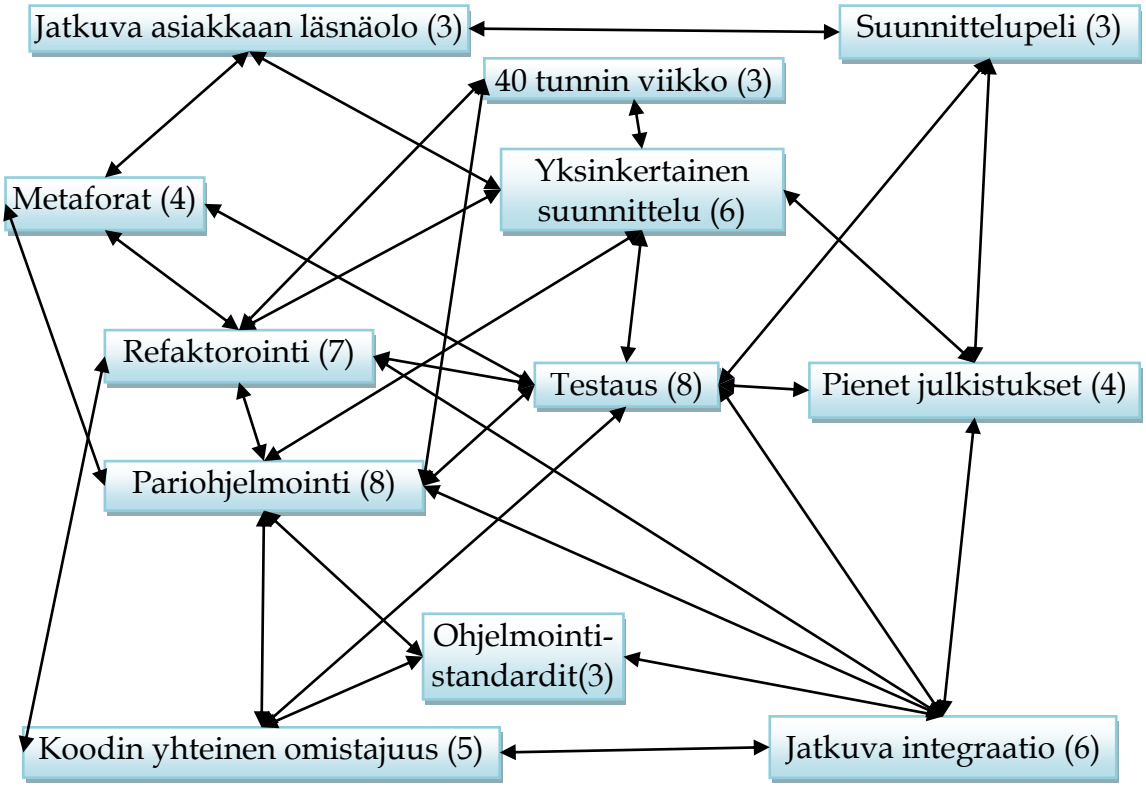
- Tekeekö tiimi sprintin edistymiskäyrän itse ja tietääkö tiimi oman nopeutensa?
- Häiritsevätkö tiimin ulkopuoliset ihmiset tiimin työskentelyä sprintin aikana?

Nokia-testi on hyvin suppea, mutta toisaalta tehokas keino sen selvittämiseksi, tietävätkö projektissa työskentelevät todella, mitä he tekevät. Toisessa, Glogerin (2011) esittämässä tarkistuslistassa käydään Scrumin käytänteet visuaalisesti läpi. Jokainen Scrumin käytänne käydään läpi listalla niin, että aluksi selitetään käytänteen tarkoitus ja perusasiat sekä käytänteeseen liittyvät roolit. Sitten katsotaan, mitä käytänteen tulisi tuottaa, mikä on sen menettelytapa ja lopuksi kerrotaan käytänteeseen kuluva aika ja paikka sprintissä (Gloger 2011). Lista auttaa erityisesti sellaisten käytänteiden osalta, joita ei yrityksessä vielä kovin hyvin tunneta, koska listan avulla voidaan tarkistaa nopeasti, mistä kussakin käytänteestä on oikeasti kyse.

Kolmannen Scrumin tarkistuslistan (”The Unofficial Scrum checklist”) on esittänyt Kniberg (2010) (liite 3). Lista sisältää kysymyksiä Scrumin käytänteistä luokiteltuna Scrumin ydinkäytänteisiin (engl. Core Practices), skaalautuvuuteen, positiivisiin indikaattoreihin ja ”suositeltuihin, mutta ei aina tarpeellisiin” käytänteisiin. Jos yrityksessä voidaan todeta, että vähintään neljän viikon välein asiakkaalle toimitetaan toimivaa ja testattua ohjelmistoa, toimitetaan sitä, mitä asiakas tarvitsee eniten ja prosessi kehittyy koko ajan, voi Knibergin (2010) mukaan listan muut kohdat jättää täyttämättä, prosessi on automaattisesti kunnossa. Lista ei anna esimerkiksi mitään konkreettista pisteytystapaa, mutta sen avulla on nopea todeta, mitkä yrityksen prosesseista ovat kunnossa ja mitkä kaipaavat vielä lisää työstämistä, jotta Scrumia hyödynnettäisiin vieläkin paremmin. Knibergin lista on näistä esitellyistä käyttöönoton mittareista tiivistetyin. (Kniberg 2010)

XP:n käyttöönoton arviointi

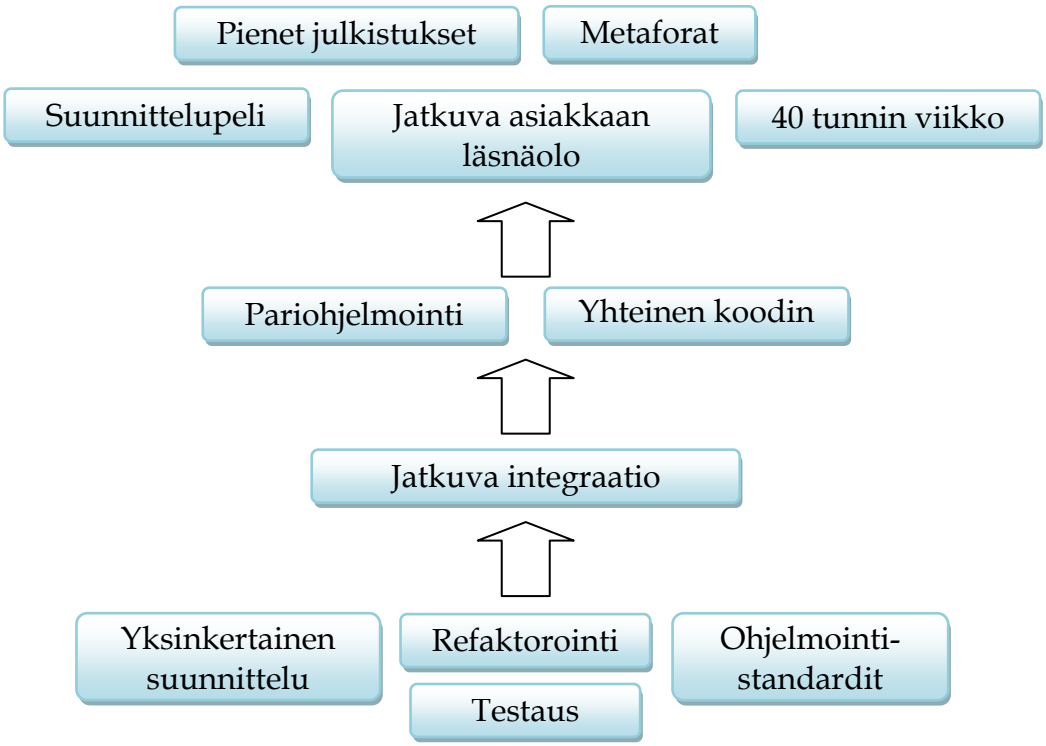
XP:n käyttöönoton luulisi olevan helpompaa kuin Scrumin, koska XP koostuu näennäisesti toisistaan erillään olevista käytänteistä. Lui ja Chan (2005) toteavat kuitenkin, että XP:n käytänteet liittyvät todellisuudessa hyvinkin tiiviisti toisiinsa. He havainnollistivat XP:n käytänteiden keskinäisiä suhteita piirtämällä jokaisen toisiinsa liittyvän käytänteen välille viivan (KUVIO 10). Laskemalla kuhunkin käytänteeseen liittyvien muiden käytänteiden määrän, saadaan esille, mitkä käytänteet ovat toistensa suhteen keskeisiä.



KUVIO 10 XP:n käytänteiden keskinäiset suhteet (Lui & Chan 2005, 477)

Suhteiden määriin perustuvalla klusterianalyysilla ovat tutkijat ryhmitelleen XP-käytänteet neljälle tasolle (KUVIO 11). Alimmalle tasolle sijoitetut käytänteet (testaus, yksinkertainen suunnittelu, refaktorointi ja ohjelmointi-standardit) tulisi ottaa käyttöön ensimmäisenä. Toisessa vaiheessa otettaisiin mukaan jatkuva integraatio, sitten jatkettaisiin pariohjelmoinnin ja yhteisen koodin

omistajuuden opettelulla. Neljännellä tasolla mukaan tulisivat loput käytänteistä. (Lui & Chan 2005)



KUVIO 11 XP:n käytänteet käyttöönottojärjestyksessä (Lui & Chan 2005, 480).

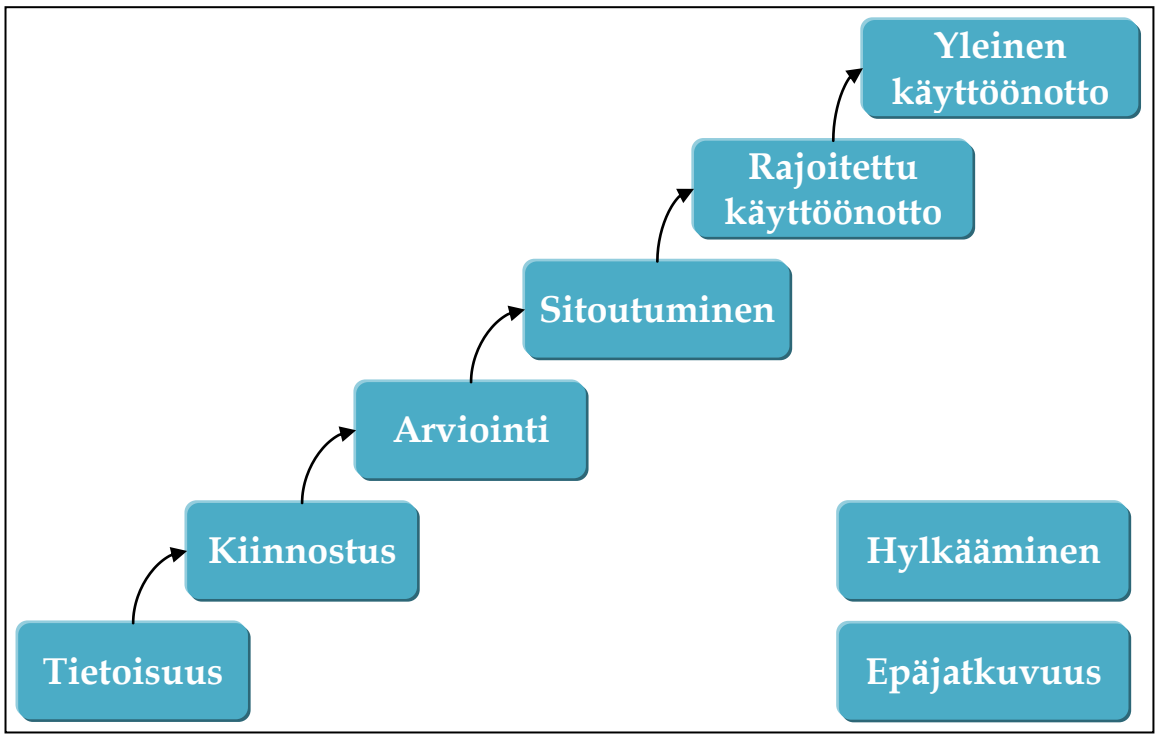
Ketterän menetelmän käyttöönotto yleisesti

Yksittäisten menetelmien käyttöönoton mittareiden lisäksi on olemassa ketterään kehittämiseen erikoistuneita kehyksiä ja kypsyysmalleja (esim. Benefield 2010, Packlick 2007, Patel & Ramachandran 2009, Qumer & Henderson-Sellers 2008, Sidky, Arthur & Bohner 2007) ja tietysti myös menetelmästä riippumattomia arviointiasteikkoja käyttöönoton tason arviointiin. Yksi esimerkki näistä menetelmästä riippumattomista arviointiasteikoista on Fichmanin ja Kemererin (1997) esittelemä ohjelmistokehityksen menetelmän käyttöönoton malli. He esittelevät nk. Guttmanin assimilaatiotasot, jotka tarjoavat luokittelun menetelmän käyttöönoton ja käytön tasolle (Fichman & Kemerer 1997). Taulukossa 1 on esitetty kaikki assimilaatiotasot ja tasoille siirtymisen kriteerit.

TAULUKKO 1 Menetelmän käyttöönoton ja käytön taso yrityksessä (Fichman & Kemerer 1997, 1350-1351)

Taso	**Tasolle siirtymisen kriteeri**
1. Tietoisuus (engl. awareness)	Avainpäättäjät ovat tietoisia menetelmästä X
2. Kiinnostus (engl. interest)	Organisaatio on sitoutunut aktiivisesti oppimaan lisää menetelmästä X
3. Arviointi / kokeilu (engl. evaluation / trial)	Organisaatio on hankkinut tiettyjä innovaatioon liittyviä tuotteita ja on aloittanut menetelmän arvioinnin tai kokeilun
4. Sitoutuminen (engl. commitment)	Organisaatio on sitoutunut käyttämään menetelmää X merkittävästi yhdessä tai useammassa projektissa
5. Rajoitettu käyttöönotto (engl. limited deployment)	Organisaatio on ottanut ohjelmaansa menetelmän X säännöllisen, mutta silti rajoitetun käytön
6. Yleinen käyttöönotto (engl. general deployment)	Organisaatio on saavuttanut tason, jossa menetelmää X käytetään uudessa kehityksessä merkittävässä osassa
Kategoria	**Kriteeri**
Hylkääminen (engl. rejection)	Organisaatio on arvioinut ja hylännyt menetelmän X nykyisessä tilanteessa
Epäjatkuvuus (engl. discontinuance)	Organisaatio on sitoutunut käyttämään menetelmää X jossakin vaiheessa menneisyyttä, mutta ei käytä sitä tällä hetkellä, eikä usko käyttävänsä sitä tulevaisuudessa

Kuviossa 12 on esitetty havainnollistava kuva Guttmanin assimilaatiotasoista täydennettynä Fichmanin ja Kemererin omilla kategorisoinneilla. Ensimmäinen taso käyttöönotossa Fichmanin ja Kemererin (1997) mukaan on tietoisuus menetelmästä yrityksen sellaisella tasolla, joka tekee päätöksiä. Tämä voi johtaa yrityksen sisällä kiinnostuksen tasolle, jossa halutaan ottaa selvää ja tutustua menetelmään paremmin. Arviointitasolla yritys on ottanut askeleen pidemmälle ja testaa menetelmää joko kokeilun tai muunlaisen arvioinnin avulla. Jos arvioinnista saadaan positiiviset tulokset, ollaan valmiita sitoutumaan menetelmän käyttöön laajemminkin yrityksessä useissa projekteissa. Rajoitetun käyttöönoton tasolla ollaan, jos yritys hyödyntää menetelmää projekteissaan säännöllisesti, mutta rajoitetulla alueella. Menetelmä on saavuttanut yleisen käyttöönoton tason, jos se on käytössä merkittävissä uusissa, myös kriittisissä ja laaja-alaisissa, projekteissa. (Fichman & Kemerer 1997)



KUVIO 12 Havainnollistava kuvio menetelmän käyttöönoton ja käytön tasosta yrityksessä (Fichman & Kemerer 1997, 1350-1351)

Jos menetelmä on arviointivaiheessa osoittautunut sellaiseksi, että sitä ei haluta hyödyntää senhetkisessä tuotannossa, on menetelmä mallin mukaan hylätty. Epäjatkuvuus tarkoittaa taas sitä, että menetelmä on edennyt mallin portailla arviointivaihetta ylemmäs ja se on ollut jossakin vaiheessa yrityksessä oikeassa tuotantokäytössä, mutta sitä ei hyödynnetä enää eikä uskota hyödynnettävän myöskään tulevaisuudessa. (Fichman & Kemerer 1997)

Tässä tutkimuksessa päädyttiin hyödyntämään tätä käyttöönotettavasta menetelmästä riippumatonta mallia sen yksinkertaisuuden vuoksi.

2.7 Yhteenveto

Tässä luvussa esiteltiin ensimmäiseksi ketterä lähestymistapa yleisesti Agile-manifestin pohjalta. Ketteryyden ideologiaan tutustumisen lisäksi luvussa esiteltiin tunnetuimpia ketteriä menetelmiä, Scrumia, XP:tä sekä lyhyesti Leania

ja Kanbania. Scrumiin ja XP:hen syvennyttiin tarkemmin käyden läpi menetelmien perusprosessi ja niihin tiiviisti liittyvät roolit. Menetelmien esittelyn jälkeen kerrottiin ketterien menetelmien käytön yleisyydestä sekä kuvattiin ketterän lähestymistavan käyttöönottoon liittyviä kypsyysmalleja ja arviointiasteikkoja, joita hyödynnetään myöhemmin haastattelututkimuksessa ja haastattelututkimuksen tulosten arvioinnissa. Seuraavassa luvussa tutustutaan aiemmin julkaistuihin empiirisiin tutkimuksiin ketterien menetelmien ja käytänteiden käytöstä.

3 YLEISKATSAUS KETTERÄÄ LÄHESTYMISTAPAA KOSKEVIIN EMPIIRISIIN TUTKIMUKSIIN

Ketterää lähestymistapaa koskevia empiirisiä tutkimuksia on tehty runsaasti, mutta niiden näkökulmat, tieteellinen taso, tutkimuksen toteuttamisen laajuus ja tutkimuksen kohteet vaihtelevat suuresti (Dybå & Dingsøyr 2008). Tämän luvun tarkoituksena on antaa kokonaiskuva ketterää lähestymistapaa koskevasta empiirisestä tutkimuksesta sekä tutkimuksen tilanteesta. Tämä tehdään käyttämällä hyväksi aiemmin tehtyjä kirjallisuusanalyyseja: Dingsøyr, Dybå ja Abrahamsson (2008), Dybå ja Dingsøyr (2008), Hulkko ja Abrahamsson (2005), Montesi ja Lago (2007), Siniaalto ja Abrahamsson (2007) sekä Sjøberg, Dybå ja Jørgensen (2007). Yksittäisiä periaatteita ja käytänteitä koskevia empiirisiä tutkimustuloksia käsitellään kuitenkin vasta luvussa 5 kunkin käytänteen kuvauksen yhteydessä.

Dybå ja Dingsøyr (2008) ovat käyneet läpi lähes 2000 kappaletta vuoteen 2006 mennessä kirjoitettua ketterää lähestymistapaa käsittelevää artikkelia, joista tieteelliseksi luokiteltavia empiirisiä tutkimuksia oli lopulta vain 36 kappaletta. Näistä 33 oli ensisijaisia tutkimuksia, joita käsitellään artikkelissa tarkemmin. Kirjoittajat kävivät systemaattisesti valitut artikkelit läpi niin, että niistä analysoitiin tutkitut aiheet sekä tärkeimmät tulokset. Empiiristen tutkimusten tuloksia on soveltuvilta osin myös vertailtu keskenään.

Suurin osa empiirisistä tutkimuksista koski XP:tä. Yksi syy tähän saattaa olla siinä, että XP:n käytänteet ovat, kuten jo aikaisemmin on mainittu, tavallaan kokoelma hyväksi havaittuja menettelytapoja sekä XP on menetelmänä hieman vanhempi kuin esimerkiksi Scrum tai Lean-ohjelmistokehitys. Tällöin empiiristä tutkimusta on voitu tehdä jo aiemmin. Tämän lisäksi XP:n käytänteet ovat ehkä muihin menetelmiin verrattuna helpompia hahmottaa yksittäisinä osina,

jolloin myös niiden yksittäinen tutkiminen on selkeämpää. Dybån ja Dingsøyrin (2008) tarkastelemat, XP:tä koskevat tutkimukset oli useimmiten toteutettu kuitenkin muissa ympäristöissä kuin oikeissa ohjelmistoalan projekteissa, eivätkä tutkimukseen osallistujat olleet kokeneita ammattilaisia, vaan opiskelijoita.

Muista kuin XP:stä tehtyjen empiiristen tutkimusten määrä on yllättävän vähäinen: kahdeksan kappaletta. Näistäkin tutkimuksista viisi koskee ketteriä menetelmiä yleensä, yksi Scrumia, yksi Lean ohjelmistokehitystä (engl. Lean Software Development) ja yksi tutkimuksen kohteena olleen yrityksen omaa ketterää menetelmää. (Dybå & Dingsøyr 2008)

Kirjoittajat kategorisoivat käsittelemänsä empiiriset tutkimukset neljään luokkaan (Dybå & Dingsøyr 2008):

- perehdyttäminen ja omaksuminen (engl. introduction and adoption)
- inhimilliset ja sosiaaliset tekijät (engl. human and social factors)
- asiakkaan ja kehittäjän näkemykset (engl. customer and developer perceptions)
- vertailevat tutkimukset (engl. comparative studies).

Perehdyttämiseen ja omaksumiseen keskittyneitä empiirisiä tutkimuksia oli seitsemän kappaletta (esim. Bahli & Zeid 2005, Merisalo-Rantanen, Tuunanen & Rossi 2005, Svensson & Höst 2005a), joiden osalta empiiriset havainnot olivat osittain ristiriidassa keskenään. Esimerkiksi XP:n käyttöönotto suuressa organisaatiossa oli osoittautunut vaikeaksi yhdessä pilottiprojektissa (Svensson & Höst 2005b), mutta toisessa, vesiputousmallista XP:hen siirtyneessä projektissa (Bahli & Zeid 2005), XP:n käyttöönotto oli sujunut kokemattomilta projektin jäseniltä vaivattomasti. Osassa tutkimuksista oli perehdyttämistä ja omaksumista tarkasteltu myös yksittäisten käytänteiden tasolla (esim. Svensson & Höst 2005b, Tessem 2003), eikä niissä noussut esille montaakaan negatiivista asiaa,

poikkeuksena Tessemin (2003) tutkimuksessa ilmenneet pariohjelmointia koskeneet seikat. (Dybå & Dingsøyr 2008)

Inhimilliset ja sosiaaliset tekijät olivat kiinnostuksen kohteena seitsemässä empiirisessä tutkimuksessa (esim. Chong 2005, MacKenzie & Monk 2004, Robinson & Sharp 2005). Pääosin niissä tutkittiin organisaation kulttuurin, yhteistyön tai tiimien luonteen vaikutusta ketterässä lähestymistavassa. Robinson ja Sharp (2005) totesivat kolmea yritystä tarkastelevassa tutkimuksessaan, että vaikka organisaation tyyppi, rakenne tai fyysiset puitteet olivat erilaiset, XP tuntui toimivan hyvin. XP:n todettiin myös vaikuttavan positiivisesti tiimien yhteistyöhön, koska sen käytänteet rohkaisevat avoimeen kommunikointiin ja pitävät kehitysprosessin läpinäkyvänä kaikille. Ketterien tiimien tärkeimmäksi ominaisuudeksi nostettiin luottamus omaan osaamiseen ja myös luottamus muihin tiimin jäseniin. (Dybå & Dingsøyr 2008)

Yhdeksän tutkimusta käsitteli havaintoja ketterästä lähestymistavasta asiakkaan ja kehittäjän näkökulmasta tarkasteltuna. Useassa näistä (esim. Ilieva, Ivanov & Stefanova 2004, Mann & Maurer 2005) tutkimuksista asiakkaiden tyytyväisyys kehitettyyn ohjelmistoon parantui Scrumin tai XP:n käytön myötä, koska he tunsivat olevansa tiiviimmin mukana ohjelmiston kehitysprosessissa. Asiakkaan rooli nähdään osassa tutkimuksista (esim. Martin, Biddle & Noble 2004) myös joskus uuvuttavana, koska ketterissä menetelmissä asiakkaalta vaaditaan enemmän osallistumista ja panosta projektin onnistumiseksi. Tutkimuksiin osallistuneiden kehittäjien ja opiskelijoiden näkökulmasta katsottuna Scrumin ja XP:n käytänteet olivat pääosin hyödyllisiä ja he aikoivat suositella niiden käyttämistä myös tulevissa projekteissa. Pariohjelmointi tuotti tässäkin näkökulmassa eniten toisistaan poikkeavia tuloksia. Vaikka pariohjelmointi nähtiin useimmissa tutkimuksissa (esim. Mannaro, Melis & Marchesi 2004, Ilieva, Ivanov & Stefanova 2004) hyödylliseksi käytänteeksi, tuotiin niissä

esille myös tilanteita, joissa pariohjelmointi on koettu raskaaksi. (Dybå & Dingsøyr 2008)

Vertailevia tutkimuksia oli tehty eniten, 11 kappaletta (esim. Baskerville, Ramesh, Levine ym. 2003, Ceschi, Sillitti, Succi & De Panfilis 2005, Dalcher, Benediktsson & Thorbergsson 2005, Ilieva, Ivanov & Stefanova 2004, Layman, Williams & Cunningham 2004). Näissä tutkimuksissa vertailtiin jotakin ketterää menetelmää johonkin toiseen ohjelmistotuotannon menetelmään. Ketterillä menetelmillä oli positiivinen vaikutus projektinhallinnallisesta näkökulmasta katsoen, koska niiden käyttö paransi asiakassuhteita ja suunnitelmatkin olivat parempia. Erityisesti asiakassuhteet ja niiden paraneminen tuotiin useassa tutkimuksessa esille. Myös lyhyet ohjelmiston toimitussyklit auttoivat parantamaan kommunikointia, koska palautetta on helpompi antaa toimivasta ohjelmistosta kuin pelkkien dokumenttien perusteella. Perinteiseen ohjelmistokehitykseen verrattuna ketterissä menetelmissä tiimien toiminta oli tehokkaampaa ja ne tuottivat yritykselle arvokkaampaa koodia nopeammin. Toisaalta ketterien menetelmien mukainen, erittäin tiivis tiimin yhteistyö aiheuttaa sen, että tiimin jäsenten vaihtaminen kesken projektin on vaikeampaa kuin perinteisessä tiimissä. (Dybå & Dingsøyr 2008)

Neljässä tutkimuksessa (Dalcher, Benediktsson & Thorbergsson 2005, Ilieva, Ivanov & Stefanova 2004, Layman, Williams & Cunningham 2004, Wellington, Briggs & Girard 2005) keskityttiin tutkimaan ketterien menetelmien tuottavuutta, joissa yhtä tutkimusta lukuun ottamatta ketterä tiimi oli tuotteliaampi kuin verrokkiryhmänsä. Yhdessä tutkimuksessa (Wellington, Briggs & Girard 2005) perinteinen tiimi oli kuitenkin ketterää tiimiä tuotteliaampi. Myös laatua on pyritty mittaamaan ja vertaamaan tuloksia perinteisiin menetelmiin tai yrityksen aikaisempaan tilanteeseen. Tutkimuksissa tehdyt johtopäätökset tukivat useimmiten sitä oletusta, että ketterä lähestymistapa nostaa tuotteen laatua,

mutta niitäkin tutkimuksia oli, joissa laadun nousua tai laskua ei voitu todeta eri tiimien välillä. Näiden havaintojen lisäksi muutamassa tutkimuksessa todettiin ketterän lähestymistavan vaikuttavan tiimin yhteenkuuluvuuteen ja töissä viihtymiseen positiivisella tavalla. (Dybå & Dingsøyr 2008)

Tutkimuksen johtopäätöksissä Dybå ja Dingsøyr (2008) toteavat, että tarve tarkemmalle ketterän lähestymistavan tutkimukselle on suuri. Heidän läpikäymistään tutkimuksista suurin osa koski XP:tä, vaikka Scrum oli selkeästi jo tutkimuksen tekoaikana suosittu ketterä menetelmä. Empiiristen tutkimusten laatu ei ollut myöskään tarpeeksi korkea, vaan tutkijat olisivat halunneet nähdä enemmän oikeassa kehitysympäristössä tehtyjä tutkimuksia. Tehdyt empiiriset tutkimukset kaipasivat myös tieteellisempää näkökulmaa. Tarkastelluista empiirisistä tutkimuksista hyvin monesti esimerkiksi tutkimusmenetelmän kuvaus, tutkittavien kohteiden valintaperusteet, oikeellisuuden, luotettavuuden sekä puolueellisuuden arvioiminen olivat vajavaisia tai ne puuttuivat kokonaan. Kirjoittajat peräänkuuluttavatkin systemaattisemman tieteellisen tutkimuksen tekemistä ketteristä menetelmistä. (Dybå & Dingsøyr 2008)

Dybån ja Dingsøyrin (2008) koosteen jälkeen vastaavaa yhteenvetoa ketterien menetelmien tutkimuksen tilanteesta ei ole tehty, ainakaan samassa laajuudessa. Abrahamsson, Conboy ja Wang (2009) ovat samoilla linjoilla Dybån ja Dingsøyrin kanssa tutkimuksen tilanteesta ja toteavat, että ketterän lähestymistavan tutkijoiden tulisi eritoten kiinnittää huomiota siihen, miten ketterän lähestymistavan vaikutuksia voitaisiin mitata ja arvioida tieteellisellä tavalla. Heidän mukaansa tutkimuksissa olisi hyvä kiinnittää huomiota myös ketteryyden käyttöön ja vaikutuksiin myös organisaation tasolla.

Empiirisen tutkimuksen tulevaisuutta pohtinut Dag Sjøberg kertoo Tichyn (2011) tekemässä haastattelussa vievänsä ketterän lähestymistavan tutkimusta

eteenpäin myös omalta osaltaan toteuttamalla tutkimuksia nimenomaan oikeissa ohjelmistoalan projekteissa, jolloin tutkimuksen tulokset kertovat todellista tietoa tutkimuksen aiheesta. Sjøberg huomauttaa haastattelussa siitä, miten usein toteutettujen tutkimuksen kysymykset keskittyvät toteamaan, ”toimiiko joku teknologia” tai ”onko menetelmä A parempi kuin menetelmä B?”, kun todellinen kysymys pitäisi olla ”milloin A on parempi kuin B?”(Tichy 2011, 2).

Tämä tutkimus sijoittuu Dybån ja Dingsøyrin (2008) kategorisoinnissa kolmanteen ryhmään, jossa ketteriä menetelmiä tarkastellaan kehittäjän (tai asiakkaan) näkökulmasta katsottuna, vaikka osa tutkielmassa käsitellyistä asioista koskee luonnollisesti myös tiimin sisäistä dynamiikkaa ja ketteryyden sosiaalista näkökulmaa. Tällä tutkimuksella pyritään vastaamaan Dybån ja Dingsøyrin (2008) sekä Abrahamssonin ym. (2009) pyyntöön tehdä ketterään kehittämiseen liittyvää tutkimusta aidossa yritysympäristössä.

4 HAASTATTELUTUTKIMUKSEN TOTEUTTAMINEN

Edellisen luvun perusteella voidaan sanoa, että tällä tutkimuksen alueella on selkeä tarve empiiriselle tutkimukselle, joka selvittää, mitä käytäntöjä yrityksissä on todellisuudessa käytössään, millä tavalla käytänteitä hyödynnetään ja millaisia kokemuksia niiden käytöstä on saatu. Tässä tutkimuksessa pyritään valottamaan näitä kysymyksiä neljässä suomalaisessa ohjelmistoalan yrityksessä tehdyllä haastattelututkimuksella. Tässä luvussa esitellään tutkimusmenetelmän valinta, tutkimuskohteiden valinta, aihealueen rajaus sekä tiedon keruutapa ja sen käsittelymenetelmät.

4.1 Tutkimusmenetelmän valinta

Ketterien käytänteiden käyttöä ja käyttökokemuksia voidaan tutkia empiirisesti useammalla tavalla. Kysely- eli survey-tutkimuksella olisi mahdollista selvittää laajan yritysjoukon osalta, millaisia käytänteitä niissä on käytössä. Otoksen ollessa tarpeeksi suuri ja edustava, on mahdollisuuksia yleistäviin johtopäätöksiin. Kyselytutkimusten heikkoutena on kuitenkin se, että ne jäävät enemmän tai vähemmän pintapuolisiksi. Niillä on vaikea selvittää vastauksia Miksi –kysymyksiin (Hirsjärvi, Remes & Sajavaara 2000). Toisena vaihtoehtona olisi käyttää tutkimusmenetelmänä tapaustutkimusta, jolla voitaisiin porautua valitun yrityksen tilanteeseen syvällisemmin (Hirsjärvi ym. 2000). Resurssien salliessa kohteeksi voitaisiin valita useampiakin yrityksiä. Yksittäisen tapauksen, tai muutamienkaan tapausten, perusteella ei ole kuitenkaan mahdollisuutta yleistyksiin. Pro gradu –tutkielmana ei ole mahdollisuutta suorittaa tapaustutkimusta useammassa yrityksessä.

Tässä tutkimuksessa on siten päädytty käyttämään edellä mainittujen puoliväliin sijoittuvaa ratkaisua. Tutkimus toteutetaan kvalitatiivisena tutkimuksena,

jossa tiedonkeruumenetelmänä käytetään haastattelua. Tiedonkeruumenetelmäksi valittiin haastattelututkimus siksi, että tehdyn kirjallisuuskatsauksen perusteella voitiin odottaa vastausten vaativan mahdollisesti selventäviä ja asiaa syventäviä lisäkysymyksiä (Hirsjärvi ym. 2000). Haastattelun aihepiiri on myös niin laaja, että vastausten suuntaa ei voitu etukäteen tietää, jolloin haastattelututkimus antaa menetelmänä välineet käsitellä myös sellaiset vastaukset, joita ei ennen haastattelua ole osattu odottaa (Hirsjärvi ym. 2000).

Haastattelun kohteina käytetään useamman yrityksen edustajia. Ennen varsinaisia haastatteluja yritysten valitsemiin haastateltaviin oltiin yhteydessä sähköpostitse alustavan kyselyn muodossa (liite 1). Sen tarkoituksena oli rajata valmiiksi käsiteltävistä käytänteistä pois sellaiset, joista haastateltavilla ei ollut riittävästi kokemusperäistä tietoa. Alustavan kyselyn tulokset esitellään tarkemmin alaluvussa 4.3. Alustavan kyselyn tietojen perusteella koottiin haastatteluja tukeva haastattelurunko, jonka mukaan jokainen haastattelu eteni niiden käytänteiden osalta, joista haastateltavalla oli kokemusperäistä tietoa. Haastattelu oli strukturoitu haastattelu, eli lomakehaastattelu, jossa ”kysymysten ja väitteiden muoto ja esittämisjärjestys on täysin määrätty” (Hirsjärvi ym. 2000, 195). Haastattelut toteutettiin yksilöhaastatteluina. Ennen varsinaisten haastattelujen tekemistä haastattelurungon toimivuutta testattiin yhden koehaastattelun avulla, jonka perusteella haastattelurunko kirjoitettiin lopulliseen muotoonsa.

4.2 Tutkimuskohteiden valinta

Tällä tutkimuksella pyrittiin selvittämään ketterien menetelmien ja käytänteiden käyttötapoja ja –kokemuksia, ei esimerkiksi niiden levinneisyyttä. Tästä syystä tutkimuskohteiksi pyrittiin saamaan sellaisia yrityksiä, joilla oli jo kokemusta ketterien menetelmien käytöstä ohjelmistotyössä. Ketterien menetelmien

hyödyntämistä selvitettiin joko ottamalla henkilökohtaisesti suoraan yhteyttä yrityksissä työskenteleviin ihmisiin ja tiedustelemalla heiltä asiaa tai tutustumalla yritysten toimintaan heidän kotisivujensa kautta. Halukkuutta haastatteluun kysyttiin sähköpostitse tai suullisesti viideltä yritykseltä, joista neljä suostui mukaan tutkimukseen. Kustakin yrityksestä varauduttiin haastattelemaan kahta työntekijää, eli maksimimäärä haastatteluille oli kymmenen haastattelua. Kunkin haastattelun arvioitiin kestävän noin puolitoista tuntia.

Yrityksen yhteyshenkilöä kehotettiin valitsemaan yrityksestään kaksi haastateltavaa siten, että vähintään toinen haastateltavista olisi toiminut tiimin jäsenen roolin lisäksi myös projektipäällikön roolissa laajemman katsantokannan saavuttamiseksi. Haastatteluihin osallistui lopulta yhteensä kuusi haastateltavaa neljästä yrityksestä. Haastattelujen määrää voidaan pitää sopivana opinnäytetyöhön, koska kyseessä ei ollut kvantitatiivinen, vaan kvalitatiivinen tutkimus (Hirsjärvi ym. 2000).

4.3 Tutkimuksen aihealueen rajaus ja alustava kysely

Tutkimuksen aihealueen karkea rajaus tehtiin kahdessa vaiheessa. Ensimmäisessä vaiheessa tutustuttiin yleisesti ketteriin menetelmiin ja niitä koskeviin aiempiin empiirisiin tutkimuksiin pääosin luvussa 3 esiteltyjen kirjallisuusanalyysien avulla. Erityisesti tässä vaiheessa tukeuduttiin Dybån ja Dingsøyrin (2008) tekemään kirjallisuusanalyysiin, jonka ansiosta saatiin selkeä kuva tutkimuksen tasosta sekä osviittaa myös lähdemateriaalin ja vertailukelpoisten aiempien empiiristen tutkimusten saatavuudesta myös yksittäisten ketterien menetelmien kohdalta. Ketteristä menetelmistä oli tarkemmin tutkittu vain kolmea menetelmää, XP:tä, Scrumia ja Lean-ohjelmistokehitystä. Siksi nämä kolme valittiin mukaan tehtäessä haastateltaville lähetettävää alustavaa kyselylomaketta (liite 1).

Rajauksen toinen vaihe tapahtui alustavasta kyselystä saatujen tietojen perusteella. Kutakin haastatteluihin osallistuvaa henkilöä pyydettiin täyttämään sähköpostitse lähetetty alustava kysely haastateltavan yrityksessä käytössä olevista käytänteistä. Taulukossa 2 on esitetty saadut tiedot tiivistetysti. Lomakkeessa annettiin lista eri käytänteistä koskien kolmea ennalta valittua ketterää menetelmää (XP, Scrum ja Lean ohjelmistokehitys) ja pyydettiin haastatteluun osallistuvia luokittelemaan käytänteet neljään luokkaan:

- K, jos käytänne on yrityksessä käytössä vakituisesti
- O, jos käytänne on joskus ollut esimerkiksi pilottiprojektissa mukana, mutta ei enää vakituisessa käytössä
- E, jos käytänteestä ei ole minkäänlaisia käyttökokemuksia yrityksessä
- X, jos käytänne ei ole ollenkaan tunnettu.

Yritykset on taulukossa 2 eroteltu toisistaan kirjaimilla A, B, C ja D, jonka lisäksi yrityksen sisällä haastateltavat henkilöt on eritelty numeroilla 1 ja 2. Tätä käytäntöä noudatetaan tästä eteenpäin myös empiirisen tutkimuksen tuloksia tarkastellessa.

Alustavan kyselyn tuloksissa ei ollut mitenkään yllättävää se, että Scrumin käytänteet olivat ylivoimaisesti selvimmin ja laajimmin käytössä kyselyyn osallistuvissa yrityksissä. Ainoastaan sprintin arviointitapaamista ei hyödynnetty käytänteenä enää niissä projekteissa, joista haastateltavalla D1 oli kokemusperäistä tietoa. Scrumin suuri käyttöaste haastateltavien joukossa voi selittyä sillä, että Scrum toi yleistyessään konkreettisen ohjeistuksen myös ketterään projektinhallintaan (Abrahamsson, Warsta, Siponen & Ronkainen 2003), ja että se tarjoaa rooliensa avulla tukirangan menetelmän toteuttamiselle oikein ja systemaattisesti käyttöönotosta lähtien kaikkine siihen kuuluvine käytänteineen. XP:n käytänteiden kohdalla ei ole havaittavissa samanlaista systemaattisuutta, vaan yritykset ovat poimineet käytänteistä mukaan vain heille parhaiten soveltuvat.

TAULUKKO 2 Haastateltavien yrityksissä käytössä olevat käytänteet ja niiden käyttöaste

Haastateltava	**A1**	**A2**	**B**	**C**	**D1**	**D2**
XP						
Pariohjelmointi	**E**	**K**	**O**	**O**	**O**	**K**
Koodin yhteinen omistajuus	**E**	**E**	**K**	**K**	**K**	**K**
40 tunnin työviikko	E	K	K	E	K	K
Testivetoinen kehitys	**K**	**K**	**O**	**O**	**O**	**K**
Jatkuva integraatio	**K**	**E**	**K**	**K**	**K**	**K**
Refaktorointi	K	E	K	K	K	K
Suunnittelupeli	E	E	O	K	O	K
Pienet julkistukset	K	E	K	K	K	K
Metaforat	E	E	X	X	X	E
Yksinkertainen suunnittelu	K	E	X	X	K	K
Jatkuva asiakkaan läsnäolo	**K**	**E**	**E**	**E**	**K**	**K**
Avoin työtila	K	K	K	K	K	K
Vain säännöt	E	E	X	X	K	K
Scrum						
Tuotteen tehtävälista	**K**	**K**	**K**	**K**	**K**	**K**
Työmäärän arviointi	K	K	K	K	K	K
Sprintti	**K**	**K**	**K**	**K**	**K**	**K**
Sprintin suunnittelukokous	**K**	**K**	**K**	**K**	**K**	**K**
Sprintin tehtävälista	**K**	**K**	**K**	**K**	**K**	**K**
Päivittäinen Scrum-tapaaminen	**K**	**K**	**K**	**K**	**K**	**K**
Sprintin katselmointitapaaminen	**K**	**K**	**K**	**K**	**K**	**K**
Sprintin arviointitapaaminen	**K**	**K**	**K**	**K**	**O**	**K**
Lean ohjelmistokehitys						
Vältä jätettä	E	E	O	E	E	K
Paranna oppimista	E	E	X	E	E	K
Päätä niin myöhään kuin mahdollista	E	E	O	E	E	E
Toimita niin nopeasti kuin mahdollista	E	E	E	E	E	E
Anna valtaa tiimille	E	E	K	K	E	K
Tee eheää työtä	E	E	X	E	E	K
Näe kokonaisuus	E	E	X	E	E	K

Alustavan kyselyn perusteella valittiin tarkempaan tarkasteluun vain Scrumin ja XP:n käytänteitä. Lean-ohjelmistokehitys jätettiin alustavan kyselyn perusteella tutkimusalueen ulkopuolelle, koska yli puolella haastatteluun osallistuneista ei ollut sen käytänteistä konkreettisia kokemuksia. Koska haastatteluihin käytettävä aika oli rajallinen, oli Scrumin ja XP:n käytänteistä rajattava mukaan haastatteluun vain osa. Lopullisten käytänteiden valintaan vaikuttivat ensisijai-

sesti aiempien empiiristen tutkimustulosten saatavuus. Mitä enemmän aikaisemmin tehtyjä tutkimustuloksia on saatavilla, sitä paremmin pystytään vertailemaan haastattelututkimuksesta saatuja tuloksia aiempiin tutkimuksiin. Tämän vuoksi pariohjelmointi ja testivetoinen kehitys olivat ensisijaisia valintoja. Loput XP:n käytänteistä valikoituivat haastattelijan omien esitietojen ja mieltymysten mukaan sitä asiaa silmälläpitäen, että valituista käytänteistä on saatavilla lähdemateriaalia empiirisen tutkimuksen taustalle ja vertailukohdaksi. Scrumiin liittyvistä käytänteistä lähes kaikki otettiin mukaan haastattelututkimukseen, koska eroavaisuuksia Scrumin käytänteiden käytössä haastateltavissa yrityksissä ei ollut kuin sprintin arviointitapaamisen kohdalla yhden haastateltavan osalta. Scrumin käytänteistä tutkimuksesta jäi pois vain yksi, työmäärän arviointi. Työmäärän arviointiin on kirjallisuudessa esitetty todella monenlaisia tapoja, joten voitiin olettaa haastateltavien kokemusten eroavan toisistaan, jolloin käytänteen käsittelyaika haastattelussa olisi ollut ajallisesti liian pitkä. Kaikki haastatteluun mukaan otetut 12 käytännettä on korostettu taulukossa 2 tekstin lihavoinnilla.

4.4 Tiedon keruu ja käsittely

Tiedon kerääminen aloitettiin haastateltavien yritysten työntekijöiden varmistuttua alustavan kyselyn merkeissä. Kaikkiin haastateltaviin otettiin aluksi yhteyttä sähköpostitse. Sähköpostissa kerrottiin lyhyesti haastattelututkimuksen tavoitteet ja menetelmät sekä arvio haastattelun kestosta. Sähköpostin liitteenä oli myös alustava kysely (liite 1), johon kaikkien haastateltavien pyydettiin vastaamaan sähköpostitse ennen varsinaisen haastatteluajankohdan sopimista. Kaikki haastateltavaksi lupautuneet palauttivat vastauksensa alustavaan kyselyyn, ja heidän kanssaan sovittiin haastattelulle tarkempi ajankohta. Haastattelurungon toimivuutta testattiin ja muokattiin ennen varsinaisia haastatteluja yhden koehaastattelun avulla. Koehaastatel-

tava oli työskennellyt ketterää lähestymistapaa hyödyntävissä projekteissa muutaman vuoden ajan, joten koehaastattelun avulla saatiin realistinen arvio kysymysten ja haastattelulomakkeen toimivuudesta sekä haastattelujen ajallisesta kestosta. Koehaastattelun perusteella muutamaa kysymystä tarkennettiin hieman varsinaiselle haastattelulomakkeelle.

Varsinaiset haastattelut toteutettiin maalis- ja huhtikuun aikana keväällä 2010 neljässä ohjelmistoalan yrityksessä. Haastattelut toteutettiin yritysten omissa neuvottelutiloissa tai niiden läheisyydessä, haastateltavan toiveiden mukaisesti. Haastattelutilanteeseen osallistui yksi haastateltava kerrallaan, eli haastatteluja tehtiin yhteensä kuusi kappaletta. Jokainen haastateltava oli ennen haastatteluun tuloa täyttänyt edellä käsitellyn alustavan kyselyn, jonka mukaan kunkin haastateltavan haastattelurunkoon merkittiin ne käytänteet, joista haastateltavalla ei ollut kokemusperäistä tietoa, eikä kyseistä käytännettä siis käsitelty haastattelun aikana turhaan.

Haastattelun kulku ja tiedon keruu

Haastattelu eteni jokaisessa haastattelussa saman haastattelurungon (liite 2) mukaisesti. Haastattelun alussa kullekin haastateltavalle kerrottiin hieman tutkimuksen taustasta ja selvitettiin, miksi tutkimusta ollaan tekemässä. Samalla perusteltiin myös se, miksi käsiteltävät käytänteet oli valittu mukaan haastattelurunkoon. Haastateltavilta kysyttiin myös lupa haastattelun nauhoittamiseen, ja se saatiin kaikilta osallistujilta.

Ensimmäisenä haastateltavat vastasivat yrityksen ja heidän omaa työtaustaa koskeviin kysymyksiin. Näillä taustatiedoilla pyrittiin helpottamaan aiempien empiiristen tutkimusten ja haastattelututkimuksen tulosten vertailua keskenään sekä varmistamaan haastateltavien pätevyys aihealueen asiantuntijoina. Seuraa-

vaksi haastateltava täytti paperille oman arvionsa käsiteltävien käytänteiden käyttöönoton tasosta koko yrityksessä.

Käyttöönoton tasojen käsittelyn jälkeen haastatteluissa siirryttiin yksittäisten käytänteiden käsittelyyn. Ennen jokaista käytännettä haastateltavalla oli mahdollisuus tutustua sen tiivistettyyn määritelmään, jonka jälkeen haastateltava kertoi siitä havaitut hyvät ja huonot tai ongelmalliset puolet. Hyvien ja huonojen puolien käsittelemisen jälkeen kysyttiin mahdollisia tarkennuksia tai selvennyksiä annettuihin vastauksiin ja käytiin läpi ennalta mietityt lisäkysymykset. Haastateltavat arvioivat jokaisesta käytänteestä lopuksi myös sen hyödyllisyyden tasoa Rumpen ja Schröderin (2002) artikkelissa käytetyn asteikon pohjalta kehitetyn seitsenportaisen hyödyllisyysasteikon avulla (TAULUKKO 3).

TAULUKKO 3 Haastatteluissa käytetty käytänteiden hyödyllisyysasteikko

-3	-2	-1	0	1	2	3
Käytänteestä on selvästi haittaa projektille	Käytänne haittaa projektia	Käytänne vaikeuttaa hieman projektia	Käytänteestä ei ole ollut hyötyä eikä haittaa projektille	Käytänteestä on hieman hyötyä projektille	Käytänne on hyödyllinen projektille	Käytänne on erittäin hyödyllinen

Rumpen ja Schröderin (2002, 98) artikkelissa mainittu projektin tavoitteita mittaava asteikko oli (-5) - 5, mutta tehdyn haastattelututkimuksen kannalta edellä esitetty seitsenasteinen portaikko (-3) - 3 on riittävän laaja. Käytänteen hyödyllisyyden arviointi numeerisella asteikolla jätettiin haastattelussa aina viimeiseksi kunkin käytänteen kohdalla, jotta annetulle hyödyllisyysasteelle osattiin kysyä perusteluja, jos aiemmin mainitut käytänteen hyvät ja huonot puolet eivät jostakin syystä olleet saman linjan mukaisia. Käytänteiden hyödyllisyyttä arvioitiin sen mukaan, miten haastateltava hyödynsi sitä työssään. Kun kaikki haastattelulomakkeessa olevat asiat oli käsitelty, haastateltaville kerrottiin jatkotoimenpiteistä.

Aineiston analysointimenetelmä

Kustakin haastattelusta tehtiin tiivistetty tekstiversio, haastatteluraportti, kuuntelemalla haastattelun nauhoitetta ja kirjoittamalla muistiinpanoja apuna käyttäen haastattelussa esiin tulleet selkeät, tutkimuksen kannalta relevantit mielipiteet ja havainnot. Jos jokin asia jäi vielä tässä vaiheessa epäselväksi, haastattelun tiivistelmään merkittiin kyseinen kohta huomiovärillä ja pyydettiin haastateltavaa selventämään asia omin sanoin uudelleen kirjallisesti. Jokainen haastattelun tiivistelmä toimitettiin haastateltavalle muutaman työpäivän kuluessa haastattelusta, jonka jälkeen haastateltavilla oli muutama viikko aikaa tarkistaa haastattelun tiivistelmä ja vastata mahdollisiin selvennyspyyntöihin. Tutkimuksen tuloksissa esiintyvät lainaukset eivät välttämättä ole suoria sitaatteja haastattelutilanteesta, vaan suurin osa sitaateista on poimittu haastateltavien hyväksymästä haastatteluraportista, jossa haastateltavan sanomaa on tiivistetty (poistamalla siitä esimerkiksi ylimääräiset konjunktiot) sekä muokattu luettavampaan ja paremmin ymmärrettävämpään muotoon (esim. englanninkielisiä termejä on suomennettu vastaamaan tutkielmassa käytettyjä termejä). Haastateltavien kommentit on kuitenkin säilytetty niin, ettei haastateltavan alkuperäinen ajatus ole päässyt muuttumaan. Kaikki haastateltavat kävivät oman haastattelunsa tiivistelmän läpi ja hyväksyivät sen ja siinä esiintyvät siistityt kommentit sähköpostitse tai suullisesti.

Tutkimusaineiston analysointi jatkui haastatteluraporttien yhdistämisellä. Kaikista haastatteluista saadut vastaukset ryhmiteltiin haastattelurungon kysymysten mukaiseen järjestykseen ja numeerisessa muodossa olevat vastaukset koostettiin tämän lisäksi taulukkomuotoon luettavuuden ja vertailtavuuden parantamiseksi. Vastausten ryhmittelyn jälkeen saadut samankaltaiset mielipiteet ja perustelut yhdistettiin, jotta saatiin eroteltua selvästi yhtenevät mielipiteet muusta aineistosta. Tämän lisäksi vastaukset luokiteltiin positiivisiin tai negatiivisiin kokemuksiin. Haastatteluraporttien aineistosta on poimittu mu-

kaan kaikki ne vastaukset, jotka ovat relevantteja tutkittavan käytänteen kannalta katsottuna. Haastateltavan yritystä ja henkilökohtaista työtaustaa koskevien kysymysten tarkoituksena ei ole verrata yksittäisiä henkilöitä toisiinsa, vaan ne antavat ensisijaisesti pohjan saatujen vastausten vertailuun aiempien empiiristen tutkimusten kanssa. Vastaukset myös taulukoitiin haastatteluissa saatujen vastausten vertailtavuuden helpottamiseksi.

Seuraavassa luvussa esitellään haastattelututkimuksen käytännekohtaiset tulokset käytänne kerrallaan. Luvussa 6 esitetään tutkimuksesta vedetyt johtopäätökset.

5 TUTKIMUKSEN TULOKSET

Tässä luvussa käsitellään tarkemmin haastattelututkimukseen mukaan valittuja Scrumin ja XP:n käytänteitä sekä niistä saatuja tutkimustuloksia. Haastateltavien taustatietojen esittelyn jälkeen kukin käytänne käydään läpi seuraavasti: ensin kuvataan käytänne yleisesti, jonka jälkeen käsitellään aiemmin kirjallisuudessa esiin tulleet empiiriset tutkimustulokset koskien käytänteen soveltamista ja käyttöä. Käytänteistä saadut tulokset jaotellaan karkeasti positiivisiin ja negatiivisiin kokemuksiin. Lopuksi kunkin käytänteen kohdalla käydään läpi haastattelututkimuksen tulokset. Haastattelujen tuloksissa esitellään ensin haastateltaville kerrottu lyhyt määritelmä käytänteestä sekä käytänteelle annetut arvosanat hyödyllisyysasteikolla. Hyödyllisyysasteikolla on esitetty se arvosana, jonka haastateltavat antavat käytänteelle sillä tavalla hyödynnettynä, miten he sitä yrityksessään soveltavat. Siksi mahdolliset erot annettuun määritelmään nähden on kuvattu heti kunkin käytänteen hyödyllisyyden arvioinnin jälkeen. Tämän jälkeen käsitellään käytänteestä koetut hyvät ja huonot puolet sekä muut haastateltavien kokemukset ja samalla havainnoidaan yhteneväisyydet ja eroavaisuudet aikaisemmin saatujen tutkimustulosten kanssa. Tulosten yhteenveto ja johtopäätökset esitetään luvussa 6.

5.1 Haastateltavien taustatiedot

Haastattelututkimuksen tulokset koskien haastateltavan taustaa tai yritystä on esitetty tiivistetysti taulukossa 4. Kaikki haastateltavat olivat kokeneita ohjelmistotuotannon ammattilaisia, ja kaikilla oli yleistä työkokemusta ohjelmistotuotannon alalta vähintään neljän vuoden ajalta keskimäärin noin 5 - 10 projektista. Kysyttäessä haastateltavan kokemustasoa ketteristä menetelmistä tai tiimin keskimääräisen asiantuntijuuden tasoa asteikkona käytettiin seuraavanlaista jaottelua:

- aloittelija (alle vuoden kokemus)
- kokenut (1-3 vuotta)
- ekspertti (yli 3 vuotta)

Tällä karkealla asteikolla mitattuna kaikki haastateltavat ylsivät ekspertti-tasolle. Haastateltavat olivat myös toimineet projekteissa projektipäällikön ja/tai Scrum-mestarin roolissa, joten heillä on kokemusperäistä pohjaa arvioida haastatteluissa käsiteltäviä käytänteitä myös projektin hallinnollisesta näkökulmasta katsottuna.

TAULUKKO 4 Yleiset organisaatioon ja haastateltavan taustaan liittyvät kysymykset ja saadut vastaukset

Haastateltava	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**
Haastateltavan tiedot						
Kokemus ohjelmistotuotannon töistä vuosina	5	15	4	4	6	4
Kuinka monessa ketterässä projektissa on työskennellyt	4	5-10	40	5	10	10
Kokemus ketteristä menetelmistä (Aloittelija / Kokenut / Ekspertti)	E	E	E	E	E	E
Rooli ketterää menetelmää hyödyntävässä tiimissä (Projektipäällikkö / Tiimin jäsen / Scrum-mestari)	P, T, S	P,S	T,S	T, S	T, P/S	P,T
Tiimin tiedot						
Minkä kokoisia tiimit ovat keskimäärin?	3-4	7-10	7	2-3	2-5	3-4
Tiimin jäsenten keskimääräinen kokemustaso ohjelmistokehityksessä yleensä (Aloittelija / Kokenut / Ekspertti)	K	E	K	K	A,K,E	A, K
Onko hajautettu projekti?	Ei	Ei	Kyllä	Kyllä	Ei	Kyllä
Yrityksen tiedot						
Yrityksen koko (Pieni / Keskisuuri / Suuri)	S	S	P	P	P	P
Kuinka suuri osa projekteista on ketteriä projekteja?	60%	50%	80%	100%	33%	60%

Tiimien koot projekteissa vaihtelivat suhteellisen pienistä (2-3 henkilöä) aika suuriinkin projekteihin (7-10 henkilöä). Kaikki tiimikoot olivat kuitenkin ketterille projekteille tyypillisten tiimikokojen rajoissa. Tiimin jäsenten keskimääräinen kokemustaso edellä esitetyn asiantuntijuuden asteikon avulla ilmaistuna oli lähes kaikissa yrityksissä vähintään kokenut. Poikkeuksena tästä oli yritys D,

jonka projekteissa oli yleensä mukana jäseniä kaikilta kokemustasoilta, eli projekteissa mukana olevat aloittelijat pääsivät samalla oppimaan kokeneemmilta tiimin jäseniltä. Haastattelujen alussa käsiteltiin myös sitä, ovatko haastateltavan projektit olleet hajautettuja. Jos haastateltava oli ollut mukana Suomen sisäisesti tai kansainvälisesti hajautetuissa projekteissa, häntä pyydettiin mainitsemaan tämä asia siinä vaiheessa, kun jonkin käytänteen kohdalla kokemukset liittyivät nimenomaan hajautettuun projektiin. Koska tutkielmassa ei ole tarkoitus tutkia ketterän lähestymistavan käytänteiden toimivuutta hajautetuissa projekteissa, haastatteluissa ei keskitytty niihin liittyviin havaintoihin.

Haastatteluihin osallistuvista yrityksistä kartoitettiin taustatietoja vain yrityksen koon suhteen. Yritys A luokitellaan suureksi yritykseksi, muiden yritysten henkilöstön määrä jäi alle 250 työntekijän. Haastattelun aluksi haastateltavat saivat myös arvioida ketterien menetelmien käyttöastetta yrityksessään. Tämän arvion antaminen osoittautui kuitenkin hyvin haasteelliseksi tehtäväksi lähes kaikille haastatelluille. Siksi taulukossa esitetyt lukemat ovat vain suuntaa-antavia kultakin yrityksen edustajalta.

Ennen yksittäisten käytänteiden läpikäymistä jokainen haastateltava arvioi vielä kunkin käsiteltävän käytänteen omaksumisen tasoa edustamansa yrityksen sisällä. Arviointi tehtiin aiemmin luvussa 2.6 esitellyn Guttmanin assimilaatiotasojen avulla (KUVIO 12). Guttmanin tasojen lisäksi kyselyssä käytettiin Fichmanin ja Kemererin (1997) kategorisointia sellaisissa tapauksissa, joissa käytänne on hylätty tai sen käyttöä ei ole jatkettu. Haastattelutilanteessa haastateltavalle annettiin taulukko 1, jossa nämä assimilaatiotasot ja tasoille siirtymisen kriteerit on esitetty tiivistetysti taulukkomuodossa. Tämän taulukon mukaisesti he antoivat oman arvionsa käytön tai käyttöönoton tasosta kaikkien

niiden käytänteiden osalta, joista heillä oli alustavan kyselytutkimuksen mukaan kokemusperäistä tietoa. Arvioita käsitellään alaluvussa 5.4.

5.2 Scrumin käytänteet

Scrumin käytänteet ovat seuraavassa alaluvussa käsiteltäviin XP:n käytänteisiin verrattuna enemmän projektin hallintaan liittyviä käytänteitä. Tästä syystä oli tärkeää, että haastattelututkimuksessa oli mukana sellaisia haastateltavia, joilla oli kokemusta myös projektien vetämisestä. Kuten taulukosta 4 kävi ilmi, kaikki haastateltavat ovat toimineet vähintään yhdessä projektissa projektipäällikön tai Scrum-mestarin roolissa.

Scrumin käytänteistä tässä tutkimuksessa on mukana Scrumin tärkeimmät käytänteet: sprintti, tuotteen tehtävälista, sprintin suunnittelukokous, sprintin tehtävälista, päivittäinen Scrum-tapaaminen ja sprintin katselmointitapaaminen. Näiden lisäksi mukaan on otettu vielä sprintin arviointitapaaminen (Sutherland & Schwaber 2007). Tarkastelun ulkopuolelle jäi ainoastaan työmäärän arviointi (engl. effort estimation). Käytänteet käydään läpi edellä mainitussa järjestyksessä.

Scrumiin liittyvien käytänteiden määrittely perustuu ensisijaisesti lähteisiin Schwaber ja Beedle (2002), Schwaber (2004), Sutherland ja Schwaber (2007) sekä Schwaber ja Sutherland (2010). Rajaus näihin teoksiin on perusteltavissa sillä, että näin saadaan säilytettyä Scrumin alkuperäisten kehittäjien ajatus kunkin käytänteen luonteesta ja käyttötavasta.

5.2.1 Sprintti

Sprintti (engl. Sprint) on rajattu työskentelyjakso, jonka aikana keskitytään toteuttamaan siihen valitut tuotteen toiminnallisuudet. Sprintti voi kestää ajallisesti maksimissaan kuukauden, riippuen kuhunkin sprinttiin valituista

tehtävistä ja niiden keston arvioista. Yksi sprintin tärkeimmistä säännöistä on se, että sen aikana Scrum-tiimin työntekoon ei saa puuttua kukaan ulkopuolinen henkilö millään tavalla. Scrum-tiimi saa sprintin alussa kehitettävät kohteet tuotteen tehtävälistalta, jonka jälkeen tiimillä on täysi vapaus toteuttaa pyydetyt toiminnot juuri niin kuin se itse parhaaksi näkee. Scrum-mestarin tehtäviin kuuluu tämän työskentelyrauhan takaaminen, jotta Scrum-tiimin aikaresurssit saadaan valjastettua mahdollisimman tehokkaasti itse kehitystyöhön. Käytännössä ainoat asiat, joista Scrum-tiimi ei saa tinkiä sprintin aikana, ovat aika- ja raharesurssit sekä kehitettävän ohjelmiston laatu. (Schwaber & Beedle 2002)

Scrum-tiimille annettu vapaus sprintin aikana vaatii johdolta suurta luottamusta tiimin jäsenten ammattitaitoon. Käytännössä on mahdollista, että sprintin aikana tiimi ei tuota mitään toiminnallisuutta kehitettävään ohjelmistoon, mutta Schwaber ja Beedle (2002) huomauttavat kuitenkin, että tässäkin tapauksessa tiimi on vähintäänkin oppinut paljon. (Schwaber & Beedle 2002)

Aikaisemmat empiiriset tutkimustulokset

Sprintin hyvinä puolina nähdään se, että se tuo kehitysprosessiin entistä enemmän läpinäkyvyyttä (Paasivaara, Durasiewicz & Lassenius 2008). Lyhyenä pidetyn sprintin merkitys nousee erityisesti esille hajautettujen tiimien tapauksessa (Paasivaara, Durasiewicz & Lassenius 2008). Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) tutkimuksessa tuotiin esille, että sprintti toi selkeän ja yhtenäisen tahdin eri toimipisteiden välille, mikä helpotti projektin tarkkailua ja ongelmien havaitsemista hyvissä ajoin (Paasivaara, Durasiewicz & Lassenius 2008). Marchenkon ja Abrahamssonin (2008) tutkimuksessa Scrumia oltiin vasta ottamassa käyttöön 20 hengen osastolla ja ensimmäisen kahdeksan kuukauden tarkkailujakson aikana havaittiin, että

sprinttien haasteena oli erityisesti niiden edistymisen seuraaminen. Kyseisessä tutkimuksessa osa tiimin jäsenistä oli niin tiettyyn teknologiaan erikoistuneita, että kukaan muu tiimin jäsenistä ei oikein voinut auttaa heidän työssään, jolloin esimerkiksi sprinttiin kuuluvien tarinakorttien täyttäminen ja päivittäminen tuntui kehittäjistä turhalta (Marchenko & Abrahamsson 2008).

Haastattelututkimuksen tulokset

”[Sprintti on] rajoitettu 30 päivään, jonka aikana tiimi työstää valitsemistaan tuotteen tehtävälistan tehtävistä tuotteen toiminnallisuuksia.” (Schwaber 2004, Appendix B)

Sprintin hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	2	1	3	2	2,3

Kaikki vastaajista totesivat, että sprintti harvemmin kestää heidän yrityksissään täyttä kolmeakymmentä päivää, vaan tyypillisesti sprintin kesto on vain kaksi viikkoa. Erityisesti pienissä projekteissa sprinttien pituus pidettiin mieluummin kahdessa viikossa, laajemmissa projekteissa sitä saatettiin pidentää kolmeen tai neljään viikkoon. ”Joskus saatetaan käyttää käynnistysvaiheessa kuukautta, mutta sitten kun siirrytään siihen tekemisen vaiheeseen, niin kaksi viikkoa on aika hyvä.”(A2) Kahden viikon sprintin hyvänä puolena mainittiin lyhyempi suunnittelukokousten ja katselmointitapaamisten väli, jolloin suunniteltavaa ei ole liikaa ja suunniteltavat asiat pystyy tällä aikavälillä hallitsemaan (A1). Sprinttien pituus herätti eniten keskustelua haastateltavien keskuudessa ja sen suhteen oli tehty paljon kokeiluja haastatelluissa yrityksissä. Lyhyimmillään sprinttiä oli toteutettu seitsemän päivän pituisena, mutta näin lyhyissä sprinteissä palaverit vievät turhan paljon aikaa suhteessa muuhun kehitykseen, vaikka palaveritkin olivat suhteessa lyhyempiä kuin kahden viikon sprinteissä (A1, A2, C1, D2).

Yrityksessä C sprinttiä oli muokattu muita yrityksiä enemmän Kanbanin (Kniberg 2011) oppien mukaiseksi:

> ”Sprinttejä ei aina käytetä ollenkaan, vaan tiimi tekee tarinoita tasaiseen liukuhihnatahtiin, jonka varrella on aina välillä julkaisupäiviä. Julkaisupäivinä julkaistaan ne tarinat, jotka ovat valmiita ja muuten jatketaan tekemistä eteenpäin. Asiakkaalle tämä vaihtoehtoinen tapa näyttää aika samanlaiselle kuin sprintti. Tämä liukuhihnatapa voisi olla hyödyllisempi, koska siitä puuttuu sprintin huonot puolet, mutta siinä saavutetaan kuitenkin sprintin hyvät puolet. Liukuhihnatavassa ei myöskään tule mitään puolivalmista tilannetta, kun julkaisuja tehdään tarinapaketti kerrallaan.” (C1)

Sprintin hyviksi puoliksi mainittiin esimerkiksi tekemisen jäsentyminen (A1, C1) ja rutinoituminen (A1) sekä työn tehostuminen (A1). ”Lähtökohtaisesti on hyvää, että asiat tehdään inkrementeissä, jolloin asioilla on aina tietty pituus, jolloin ihmiset osaava asennoitua siihen tekemiseen sykleittäin.” (A2) ”Timeboxing liittää selkeästi yhteen tehtävät ja ajankulun, eli nämä tehtävät on tehtävä tässä ajassa. Jos tämä puuttuisi, niin kehittäminen olisi vähän kaoottista.” (B2) Myös tiimin näkökulmasta katsottuna sprintissä on se hyvä puoli, että se voidaan mieltää tavallaan omaksi pieneksi projektikseen, jolla on selkeä tavoite, tehtävät sekä selkeä ajallinen alku ja loppu (B1, D1, D2).

> ”Sprintin lopussa saa usein sellaisen onnistumisen tunteen, kun tietää että on saanut jotain järkevää aikaan sen sprintin aikana. Jos syklit olisivat pidempiä, ei tiimille tulisi sellaista tunnetta siellä matkan varrella, että se olisi saanut jotakin valmiiksi. Ja jos edellinen sprintti on mennyt vähän huonosti, uuden sprintin voi aloittaa tavallaan puhtaalta pöydältä.” (B1)

Sprintit auttavat kommunikoimaan myös asiakkaan kanssa paremmin, koska ”sprintit ovat hyvä syy tavata asiakasta” (C1) ja tapaamisista tulee sprinttien ansiosta säännöllisiä. Tällöin kehitettävästä tuotteesta ja sen edistymisestä saadaan myös tasaisin väliajoin kommentteja, eli tiimin ja asiakkaan välinen kommunikaatiokin parantuu (D1). Kaksi haastateltavista löysi sprintistä vain positiivisia puolia (A1, A2).

Sprintin ainoaksi haasteeksi haastateltavat mainitsivat tavalla tai toisella sprintin aikataulun sovittamisen sopivaksi sekä tiimille että asiakkaalle (B1, D2). Palautetta tulisi saada asiakkaan suunnalta tarpeeksi usein, mutta sprintin kokonaiskestosta ei kuitenkaan kovin paljon saisi mennä turhiin hallinnollisiin tehtäviin sprintin alku- ja loppupäässä (B1, C1, D1) tai ylimääräiseen testaamiseen (D2). Yrityksessä B erityisesti asiakasta ”tuntuu välillä harmittavan, jos he haluaisivat tehdä jonkun suuren muutoksen, eikä sitä voida toteuttaa vasta kun vähän ajan päästä alkavassa sprintissä” (B1).

5.2.2 Tuotteen tehtävälista

Tuotteen tehtävälista (engl. Product Backlog) sisältää jatkuvasti päivittyvän kuvauksen kaikista niistä toiminnallisuuksista, joita kehitettävään ohjelmistoon tulisi toteuttaa (Schwaber & Beedle 2002). Koska tuotteen omistaja katsoo kehitettävää ohjelmistoa usein liiketaloudellisesta näkökulmasta ja Scrum-tiimi taas hahmottaa ohjelmiston helpommin teknisen näkökulman kautta, on tuotteen tehtävälista tällöin heidän yhteistyötään helpottava työkalu, jota molemmat osapuolet kehittävät yhdessä eteenpäin. (Schwaber 2004)

Ensimmäisen sprintin alkaessa tuotteen tehtävälista voi sisältää hyvinkin epämääräisiä toiveita ja suuria linjauksia tuotettavasta ohjelmistosta. Kehityksen kannalta tällä ei ole niinkään väliä, kunhan ensimmäiseen sprinttiin saadaan asiakkaan kanssa muodostettua riittävä määrä tarpeeksi tarkalla tasolla määriteltyjä vaatimuksia. (Schwaber & Beedle 2002)

Kehitysprosessin edetessä asiakkaan ymmärrys tarvitsemistaan toiminnoista kasvaa, jolloin tuotteen tehtävälista muuttaa muotoaan jatkuvasti niin toimintojensa kuin toimintojen tärkeysjärjestyksenkin puolesta. Koska konkreettisten ominaisuuksien ja teknologisten yksityiskohtien toteuttaminen ei ole aina niin suoraviivaista, tuotteen tehtävälista voi sisältää myös toistaiseksi työstettäväksi

sopimattomia vaatimuksia (engl. issues), jotka pitää ratkaista ennen kuin varsinaisen tehtävän toteuttaminen on edes mahdollista. (Schwaber & Beedle 2002)

Tuotteen tehtävälistan hallinnoiminen kuuluu vain ja ainoastaan yhdelle henkilölle, tuotteen omistajalle. Tällä pyritään välttämään kaikki ylimääräinen epäröinti siitä, mikä on tuotteen tehtävälistan tärkein tehtävä ja mitä tehtäviä listalta tullaan toteuttamaan seuraavaksi. (Schwaber & Beedle 2002)

Aikaisemmat empiiriset tutkimustulokset

Tuotteen tehtävälistaa koskevia aiempia empiirisiä tutkimuksia on hyvin harvassa. Muutamassa tutkimuksessa kuitenkin käsitellään myös tuotteen tehtävälistan vaikutusta Scrumia hyödyntävässä projektissa. Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) havaintojen mukaan tuotteen tehtävälistan hyvänä puolena nähtiin se, että tiimin jäsenet pysyivät paremmin perillä siitä, millaista kehitystä projektissa tapahtuu, kun tehtävälista oli kaikkien nähtävillä ja tehtävät kaikkien tiimin jäsenten valittavissa. Moen, Dingsøyrin ja Dybån (2010) tutkimuksessa huomattiin kuitenkin, että tehtävien jakamisessa tiimin jäsenten kesken kannattaa olla tarkkana. Jos tuotteen tehtävälistan yksittäisestä isommasta tehtävästä vastaa joku tiimin jäsenistä yksinään, tietotaito kyseisestä tehtävästä ei kulje tiimin sisällä, eikä mahdollisia virheitäkään huomata välttämättä ajoissa (Moe, Dingsøyr & Dybå 2010). Kyseisessä tutkimuksessa tiimit olivat kuitenkin ensi kertaa tekemisissä Scrumin kanssa, joten kokemuksen myötä helpottuva tehtävien pilkkominen entistä pienempiin osiin voisi parantaa tiedon liikkuvuutta tiimin sisällä.

Haastattelututkimuksen tulokset

”[Tuotteen tehtävälista on] priorisoitu lista projektin vaatimuksista sekä niiden aika-arvioista, jotka muutetaan valmiiksi tuotteen toiminnallisuudeksi. Aika-arviot ovat päivän tasolla ja ne tarkentuvat sitä myötä, mitä korkeammalla ne

ovat tuotteen tehtävälistan prioriteettiasteikolla. Lista kehittyy muuttuvien liiketoiminta- ja teknologiamuutosten mukaan. Listassa on mukana toiminnallisia ja ei-toiminnallisia vaatimuksia sekä ongelmia (issues)" (Schwaber 2004, Appendix B)

Tuotteen tehtävälistan hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	1	3	3	3	3	3	2,7

Tuotteen tehtävälista oli tutkimukseen mukaan otetuista Scrumin käytänteistä kolmanneksi suosituin haastateltavien keskuudessa ja sitä toteutettiin suhteellisen hyvin määritelmän mukaisesti. Useimmiten määritelmästä poikettiin aika-arvioiden tekemisen suhteen. Joissakin tapauksissa aika-arvioita ei tehty vielä tuotteen tehtävälistaa muodostettaessa ollenkaan (A1), vaan tarkempi ajallinen arviointi tehtiin vasta sprinttien alussa. Yrityksessä B aika-arviot oli taas tehty huomattavasti aikaisemmin, jo ennen projektin aloittamista, koska "kustakin projektista tehdään jo ennen tuotteen tehtävälistan syntymistä tarjous, jossa on arvioitu projektin ajallista määrää. Tällöin yksittäisten tehtävien aika-arvioita ei varsinaisesti tarvita" (B1). Muista yrityksistä poikkeavasti tämän yrityksen tuotteen tehtävälista myös sisälsi useamman projektin tehtäviä ja tällöin myös projektit ovat listalla prioriteettijärjestyksessä (B1).

Tehdyt aika-arviot eivät välttämättä olleet aina nimenomaan tunneissa tai minuuteissa mitattuja, vaan tehtävien laajuutta saatettiin arvioida myös esimerkiksi pisteytyksen avulla (englanninkielinen termi tälle on story points) (C1). Aika-arvioiden päivittäminen projektin edetessä mainittiin usein myös sellaiseksi asiaksi, jota ei muistettu tai jaksettu tehdä kovin usein projektin aikana (B1, C1, D2). Yrityksessä D tämä tehtiin aivan tarkoituksella, sillä projektin alussa tehdyt aika-arviot haluttiinkin säilyttää, jotta niitä voitiin projektin lopussa verrata toteutuneisiin tunteihin ja näin kehittää tehtävien arviointikykyä entisestään (D2). Haastatteluissa kävi ilmi, että muutoin tehtäviä päivitettiin tuotteen

tehtävälistaan vähintään noin kahden viikon (A2, B1, C1) tai kuukauden välein (A1, D1, D2).

Kuten jo käytänteen yleisessä osuudessa mainittiin, tuotteen tehtävälistan hallinnoiminen kuuluu vain yhdelle henkilölle projektissa, tuotteen omistajalle (Schwaber & Beedle 2002). Käytäntö ei kuitenkaan ollut haastatelluissa yrityksissä aivan näin yksiselkoinen ja riippui usein projektista. Useimmiten tuotteen tehtävälistaa hallinnoi joko projektipäällikkö tai tuotteen omistaja, varsinkin silloin, kun asiakas ei itse halua tai pysty asettamaan tehtäviä (A2, B1, C1, D1, D2). Joissakin tapauksissa tiimi kirjasi tehtävät tuotteen tehtävälistaan, mutta päätti kuitenkin asiakkaan kanssa, mitä tehtäviä sprintteihin otetaan mukaan (A1). Tehtävien priorisoinnin osalta lopullisen päätöksen teki kuitenkin projektin asiakas kaikissa muissa haastatelluissa yrityksissä, paitsi yrityksessä B, jossa tämänkin tehtävän hoitaa tuotteen omistaja tai projektipäällikkö.

Haastateltavien käytännön kokemukset tuotteen tehtävälistasta olivat pääosin myönteisiä ja haastatteluissa nousi esille mm. tuotteen tehtävälistan rooli kommunikoinnin välineenä sekä tiimin sisäisesti, että tiimin ja asiakkaan välillä (A2, B1, C1). ”On olemassa vain yksi lista, josta näkee kaikki projektiin kuuluvat vaatimukset, jolloin sitä on helpompi käsitellä myös asiakkaan kanssa” (C1). Yrityksessä B, jossa tuotteen tehtävälistalla on muista yrityksistä poiketen samalta asiakkaalta useampia projekteja ja niihin kuuluvia tehtäviä, tuotteen tehtävälista selkeyttää varsinkin asiakkaan kanssa kommunikointia:

> ”Jos asiakas haluaa lisätä listaan jotakin uutta tehtävää tai uuden projektin, niin tuotteen tehtävälistan avulla on helppo näyttää, miten muut tehtävät sitten putoavat listalla alaspäin. Asiakas näkee silloin selkeästi haluamiensa muutosten vaikutuksen muihin tilaamiinsa projekteihin.” (B1)

Tuotteen tehtävälistan mainittiin myös selkeyttävän ja jäsentävän tekemistä, koska toiminnalliset ja ei-toiminnalliset tehtävät joudutaan kirjoittamaan konkreettisesti listalle (A2) ja ne myös joudutaan jakamaan pienempiin osiin (D1), jolloin kokonaisuuden hahmottaminen on helpompaa. Tuotteen tehtävälistan kokoaminen kullekin projektille takaa myös sen, että kaikki ne asiat, jotka projektissa pyritään toteuttamaan, tulee käytyä myös isommalla tasolla läpi (D2). Positiiviseksi puoleksi mainittiin myös se, että tehtävä työ on tuotteen tehtävälistan ansiosta aina yhdessä paikassa (B1) ja se on aina asiakkaan priorisoimaa (C1).

Haastateltavat mainitsivat myös muutamia tuotteen tehtävälistaan suoraan tai epäsuorasti liittyviä huonoja puolia. Yksi ongelma liittyy tapaan, jolla tuotteen tehtävälistaa työstetään:

> ”Jos siinä alussa on tehty joku väärä teknologinen rajaus tai vastaava ja sen perusteella tehty se karkea jaottelu, niin sitten ollaan siinä niin sanotusti jumissa niin kauan kun me aletaan sitten ottaa sieltä ylhäältä asti uudestaan sitä hommaa.” (D1)

Eli kun tehtäviä pilkotaan isommista kokonaisuuksista pienempiin, on tuotteen tehtävälistan hallinnoijan pidettävä huolta edelleen myös laajemman kuvan säilymisestä järkevänä ja toteuttamiskelpoisena. Kaikkia tuotteen tehtävälistan tehtäviä ei myös haastateltavan mukaan kannata pilkkoa kovin pieniin osiin, vaan listan hallinnoijalta vaaditaan ammattitaitoa myös siinä, että tehtävät osataan pitää tarpeeksi yleisellä tasolla.

> ”Jos tehtäviä tulee pilkottua ja rajattua liian aikaisin ja liian pieniksi palasiksi, niin silloin on vaarana, että ne eivät vastaakaan asiakkaan toiveita, jolloin on tehty turhaa työtä. Mutta tätä yritetään hallita sillä, että alussa ei tehdä kovin pieniä tehtäviä.” (D1)

Koska tuotteen tehtävälista sisältää aivan kaikki projektiin tai projekteihin liittyvät tehtävät, on sen laajuuden hallitseminen haastavaa ja aikaavievää sekä lis-

tan koostamisen vaiheessa (D2) että listan ylläpidon aikana. Jotta lista olisi todella ajantasainen, tulisi sitä päivittää tarpeeksi usein, koska muuten listalle jää roikkumaan myös sellaiset tehtävät, joita asiakas ei lopulta halunnutkaan toteutettavan (B1).

Yksi haastateltavista toi tuotteen tehtävälistan läpinäkyvyyden esille niin positiivisena asiana kuin myös käytänteen haasteena:

> ”Scrum tekee menetelmänä organisaatiosta paljon avoimemman, jolloin se näyttää oikeasti sen, miten paljon menee aikaa siihen itse työn tekemiseen ja paljonko menee aikaa kaikkeen muuhun, mutta ei silti kerro, miten sen kaiken muun saa pois. Se saattaa olla välillä jopa epämukavaa kertoa asiakkaalle, että meillä muuten vaan pystytään tekemään tämmöinen ja tämmöinen osa tästä meidän työpäivästä tätä itse työtä ja kaikki nää loput menee sitten opetteluun ja niihin kaikkeen muuhun. Tämä on haaste, ei oikeastaan hankaluus tai varsinainen huono puoli.” (A2)

5.2.3 Sprintin suunnittelukokous

Sprintin suunnittelukokous (engl. Sprint Planning meeting) on jaettu kahteen erilliseen osaan (Schwaber & Beedle 2002). Kokous on tarkoitus rajata niin, että noin 5 % koko sprintille varatusta ajasta käytetään sen suunnitteluun (Schwaber & Sutherland 2010). Tämä tarkoittaa kuukauden mittaisessa sprintissä korkeintaan kahdeksaa tuntia ja vastaavasti kahden viikon sprintissä vain neljää tuntia suunnittelua (Schwaber & Sutherland 2010).

Ensimmäisessä osassa Scrum-tiimi keskustelee tuotteen omistajan, johdon ja loppukäyttäjien kanssa siitä, mitä toiminnallisuuksia seuraavan sprintin aikana olisi hyvä toteuttaa tuotteen tehtävälistasta. Tuotteen tehtävälistasta valitaan sopivat tehtävät, jotka muodostavat sprintin tavoitteen (engl. Sprint Goal). (Schwaber & Beedle 2002)

Sprintin suunnittelukokouksen toisessa osassa Scrum-tiimi miettii keskenään, mitä kaikkea heidän tulee tehdä, jotta sprintin tavoite saadaan täytetyksi

(Schwaber & Beedle 2002). Tämä tarkoittaa valittujen tehtävien pilkkomista yksityiskohtaisemmiksi, pienemmiksi tehtäviksi (Schwaber & Beedle 2002), jolloin yhden tehtävän tekeminen saa korkeintaan viedä yhden työpäivän (Schwaber & Sutherland 2010). Nämä tehtäväkuvaukset muodostavat sprintin tehtävälistan, joka on seuraava käsiteltävä käytänne (Schwaber & Beedle 2002).

Aikaisemmat empiiriset tutkimukset

Sprintin suunnittelukokoukseen keskittyvien empiiristen tutkimusten määrä on Scrumin muiden käytänteiden tavoin hyvin alhainen verrattuna esimerkiksi joihinkin XP:n käytänteistä tehtyihin tutkimusmääriin. Muutamassa tutkimuksessa nostettiin kuitenkin esille juuri sprintin suunnittelukokoukseen liittyviä tutkimustuloksia tai huomioita. Mannin ja Maurerin (2005) kaksi vuotta kestäneessä Scrumia käsittelevässä tapaustutkimuksessa erityistä kiitosta sprintin suunnittelukokous sai projektissa mukana olleelta asiakkaalta (Mann & Maurer 2005). Asiakas oli tyytyväinen näihin tapaamisiin, koska niissä koko kehitystiimi ja asiakas ovat läsnä, jolloin kaikki tietävät, mitä heiltä odotetaan (Mann & Maurer 2005). Myös Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) tutkimuksessa todettiin yhteisen suunnitteluhetken auttavan tehtävän ymmärtämisessä ja yhteisen tavoitteen saavuttamisessa. Suunnittelukokousten ansiosta ohjelmistoa ei kehitetä niin herkästi väärään suuntaan, eikä tällöin kallisarvoista kehitysaikaa mene hukkaan (Mann & Maurer 2005). Myös kehitystiimi piti suunnittelukokouksia hyödyllisinä, koska niissä päätettiin selkeästi kehitettävän ohjelmiston rajaus (Mann & Maurer 2005). Haittapuoleksi tutkimuksessa nostettiin taas suunnittelupalaverien venyminen ja aika-arvioiden tekemisen vaikeus (Mann & Maurer 2005), vaikkakin aika-arvioiden vaikeus liittyy ennemminkin sprintin tehtävälistan haasteisiin.

Haastattelututkimuksen tulokset

”[Sprintin suunnittelukokous on] yhden päivän mittainen tapaaminen, joka kestää 8 tuntia jokaisen sprintin alussa. Tapaaminen on jaettu kahteen 4 tunnin

mittaiseen osaan. Ensimmäisessä osassa tuotteen omistaja esittelee tärkeimmät tuotteen tehtävälistan tehtävät tiimille, jonka jälkeen he miettivät yhdessä, kuinka monta tehtävää seuraavan sprintin aikana pystytään toteuttamaan. Tiimi sitoutuu tapaamisen päätteeksi tekemään sovitut tehtävät. Suunnittelukokouksen toisessa osassa tiimi suunnittelee, kuinka he toteuttavat sovitut tehtävänsä määrittelemällä ne tarkemmalla tasolla sprintin tehtävälistaan." (Schwaber 2004, Appendix B)

Sprintin suunnittelukokouksen hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	3	2	2	3	2,7

Tähän sprintin suunnittelukokouksen määritelmään nähden useammassa haastatellussa yrityksessä oli tehty pieniä muutoksia, suurimmaksi osaksi koskien suunnittelukokouksen pituutta. Koska suurin osa haastatelluista kertoi sprinttien kestävän vain puolet määritelmän mukaisesta kuukauden ajanjaksosta, on luontevaa, että sprinttien suunnittelukokouksien ajallinen kesto on suhteessa myös noin puolet pienempi verrattuna määritelmän mukaiseen kahdeksaan tuntiin (A1, A2 B1, C1, D1, D2), eikä tapaamista välttämättä eroteta kahteen erilliseen osaan (A1, C1, D1). Kaikki haastateltavat kertoivat, että sprintin suunnittelukokouksille ei ole tarvinnut määrittää erikseen jotakin tiettyä maksimipituutta, vaan suunnittelukokousten pituudet ovat pysyneet sopivissa rajoissa ilmankin.

Jos projektissa mukana oleva asiakas ymmärtää myös teknisiä yksityiskohtia, suunnittelukokous saatetaan pitää kokonaan asiakkaan kanssa yhdessä (A1). Tuotteen omistaja ei ollut aina haastateltavien projekteissa mukana sprintin suunnittelupalaverissa (A2, D1), mutta kaikkien tiimin jäsenten läsnäolo oli kuitenkin tärkeää suunnittelukokouksessa.

> "Joskus on ollut sellaisia tehtäviä käsiteltävänä, että se ei koske kaikkia tiimin jäseniä, joten on kyseenalaistettu sitä, onko kaikkien tiimin jäsenten tällöin oltava paikalla. Tiimi on kuitenkin tullut siihen lopputulokseen, että kaikkien läsnäolo on hyödyllistä, koska silloin kaikki pysyy paremmin kärryillä siitä, missä projektissa mennään." (A1)

”Tuotteen omistajan olisi hyvä olla käytettävissä myös suunnittelukokouksen toisen vaiheen aikana, koska kaikkia täsmentäviä kysymyksiä ei osata ennen tehtävien varsinaista pilkkomista edes kysyä.” (B1) Yrityksessä C tätäkin käytännettä oli kevennetty esimerkiksi siten, että tiimille ei erikseen esitelty tuotteen tehtävälistan tehtäviä. Tapaamisen tärkein tarkoitus oli varmistaa, ettei mikään asia ollut tiimille epäselvä ja että tehtävien ja vaatimusten prioriteetit ja aika-arviot olivat kohdallaan (C1). Samoilla linjoilla oltiin myös haastateltavan D2 kokemissa projekteissa.

Sprintin suunnittelukokouksen kulku meni yrityksissä hyvin saman mallin mukaisesti kuin Scrumissa on tarkoitettukin. Sprintin suunnittelukokouksen aikana päätetyt asiat kirjattiin kaikissa yrityksissä suoraan ylös sprintin tehtävälistalle, jonka asiakkaat näkivät joko suunnittelukokouksen aikana tai se lähetettiin heille tapaamisen jälkeen hyväksyttäväksi. Suunnittelukokouksen jälkeen kaikille tiimin jäsenille oli selvää, mitä tulevan sprintin aikana on tarkoitus tehdä ja mikä on sprintin konkreettinen tavoite.

Sprintin suunnittelukokous sai haastateltavilta hyvin paljon positiivista palautetta juuri tästä sen tuomasta konkreettisesta tavoitteesta ja käytännönläheisestä lähestymistavasta. Palaute oli hyvin samanlaista kuin aiemmin saaduissa tutkimustuloksissa.

> ”Ilman suunnittelua ei pärjäisi. Sprintin suunnittelukokous on ihan ehdoton edellytys tälle koko työskentelylle.” (A1)
>
> ”Tosi hyvää, että tätä suunnitellaan pienissä pätkissä, eikä niin, että suunnitellaan koko projekti etukäteen.” (A2)
>
> ”Jos tällaista suunnittelukokousta ei olisi, olisi vaikea kuvitella, miten projekti toimii. Suunnittelukokouksessa saadaan tiimille siirrettyä se tieto, että mitä ollaan tekemässä ja osataan vähän arvioida, että mitkä ne päivät on, lasketaan kuinka paljon porukalla on aikaa tehdä projektia. Eli käytän-

> nössä saadaan kasaan sellainen paketti, joka voidaan tehdä siinä parin viikon sisään.” (D2)

> ”[Sprintin suunnittelukokous] on mahdollista rakentaa niin, että se tiimi oikeasti sitoutuu siihen, mitä ne tekee. Mitä vähemmän Scrum-mestari tai tuotteen omistaja sekaantuu siihen, sitä parempi, jotta se tiimi sitten oikeasti ottaisi vastuun ja omistajuuden siinä.” (A2)

Sprintin suunnittelukokous toi useissa projekteissa myös asiakkaan tiimin kanssa saman pöydän ääreen (A1, B1, C1, D2), jolloin myös asiakas on paremmin perillä kehityksen tilanteesta ja seuraavan sprintin tavoitteesta.

Negatiivisia kokemuksia kysyttäessä huonot kokemukset riippuivat aika paljon suunnittelukokoukseen osallistuvien henkilöiden omasta osaamisesta ja jopa tunnetiloista.

> ”Tämä on helppo tehdä huonosti ja jos tämä osio tehdään huonosti, se värittää koko sitä projektia. Jos Scrum-mestari on kokematon, tiimin jäsenet ovat kokemattomia ja myös tuotteen omistaja on kokematon, niin tämä on vähän sellaista opettelua.” (A2)

Haastateltava B1 kertoi suunnittelukokousten onnistumisen riippuvan usein ihmisten tunnetilasta. Jos tiimi on väsynyt, eikä oikein jaksa keskittyä, tiimin jäsenet eivät esitä kokouksen ensimmäisessä vaiheessa tarpeeksi kysymyksiä. Tämän seurauksena he eivät tehtäviä pilkkoessaan tiedä, missä mennään ja miten tehtävät tulisi jakaa (B1). Haastateltava D1 mainitsi myös, että suunnittelukokoukseen osallistuvat henkilöt eivät aina ole välttämättä hirveän hyvin valmistautuneita kokoukseen, jolloin suunnittelukokouksesta ei saada niin suurta hyötyä irti. Asiakkaalle sprintin suunnittelukokouksessa voi olla vaikea lukita vain muutamat tietyt tehtävät seuraavaan sprinttiin ja antaa tiimille täysi työrauha niiden toteuttamiseen, vaan sprinttiin haluttaisiin lisätä tehtäviä myös sprintin aikana (A1).

5.2.4 Sprintin tehtävälista

Sprintin tehtävälista (engl. Sprint Backlog) koostuu yksityiskohtaisista sprintissä toteutettavista tehtävistä, joiden suorittamiseen arvioidaan menevän noin neljästä kuuteen tuntiin (Schwaber & Beedle 2002). Scrum-tiimi muokkaa ja päivittää sprintin tehtävälistaa koko sprintin ajan (Schwaber & Beedle 2002). Esimerkiksi joku listan tehtävistä voi vaatia lisäselvitystä jossakin asiassa, jolloin Scrumin tehtävälistaan lisätään tämä lisäselvitys uudeksi tehtäväksi (Schwaber & Beedle 2002). Myös tehtävien ajallista arviointia säädetään kokemuksen karttuessa (Schwaber & Beedle 2002). Olennaista Scrumin kannalta sprintin tehtävälistassa on se, että sprintin tehtävälistaa hallinnoi vain ja ainoastaan Scrum-tiimi (Schwaber & Sutherland 2010).

Aikaisemmat empiiriset tutkimukset

Juuri sprintin tehtävälistaa koskevia aiempia empiirisiä tutkimustuloksia oli hyvin harvassa aiemmassa kirjallisuudessa. Moe, Dingsøyr ja Dybå (2010) käsittelivät artikkelissaan Scrumin käyttöönottoa tiimityön näkökulmasta katsottuna. He tarkkailivat tutkimuksessaan käyttöönottoprosessia yhdeksän kuukauden ajan oikeassa ohjelmistoalan projektissa (Moe ym. 2010). Projektin ongelmat eivät johtuneet sprintin tehtävälistan kohdalla käytänteestä itsestään, vaan oikeastaan siitä, että sprintin tehtävälistaa ei toteutettu Scrumin määritelmän mukaisesti. Kyseisessä projektissa tuli viidennen ja kuudennen sprintin välisenä aikana alihankkijasta johtuva yllättävä tilanne, jonka ratkaiseminen vei tiimiltä kohtuuttoman kauan aikaa (Moe ym. 2010). Koska tiimi ja Scrum-mestari olivat kuitenkin vastuussa projektin ja sprintin etenemisestä, kaunisteltiin totuutta merkitsemällä puolivalmiita tehtäviä tehdyiksi (Moe ym. 2010). Tämä kuitenkin kostautui seuraavissa sprinteissä, koska tiimi tiesi joutuvansa viimeistelemään sprintin aluksi tekemättömät tehtävät loppuun tietäen, että he eivät ehdi tekemään tämän vuoksi sprinttiin suunniteltuja tehtäviä loppuun asti (Moe ym. 2010). Tällöin sprintin tehtävälista ei enää ollut tiimin jäsenille ja

projektin vetäjälle sellainen ajantasainen työkalu, jollaiseksi se on Scrumissa suunniteltu.

Haastattelututkimuksen tulokset

”[Sprintin tehtävälista on] lista tehtävistä, jotka määrittävät tiimin työtehtävät seuraavan sprintin aikana. Lista kehittyy sprintin aikana. Jokaisen tehtävän yhteydessä mainitaan, kuka on vastuussa tehtävän tekemisestä ja mikä on jäljellä olevan tehtävän aika-arvio. Aika-arviota päivitetään joka päivä.” (Schwaber 2004, Appendix B)

Sprintin tehtävälistan hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	3	2	3	3	2,8

Sprintin tehtävälistaa pidettiin XP:n jatkuvan integraation kanssa hyödyllisyysasteikolla mitattuna kaikista hyödyllisimpänä käytänteenä tutkimukseen mukaan otettujen käytänteiden joukossa. Erona Schwaberin (2004) tekemään määritelmään haastateltavat mainitsivat sen, että tehtäville ei määrätty heidän projekteissaan erikseen vastuuhenkilöä (B1, C1, D2), vaan tehtäviä teki kuka vain tiimin jäsenistä prioriteettilistan mukaisessa järjestyksessä tai jakaminen tehtiin käytettävien teknologioiden mukaan. Myöskään aika-arvioita ei päivitetty projekteissa määritelmän mukaisesti joka päivä, vaan päivityksiä tehtiin vain tarvittaessa (D1). Kaksi haastateltavista mainitsi, että heidän projekteissaan kustakin tehtävästä kirjattiin sen arvioitu kesto ennen sprintin aloittamista, mutta sitä aika-arviota ei ollut tarkoituskaan korjailla sprintin edetessä, vaan alkuperäistä arviota verrattiin toteutuneeseen aikaan sprintin loputtua (B1, D2). Yrityksessä C taas aika-arvioista oli luovuttu kokonaan:

> ”Alussa aika-arviot tehtiin tunteina, mutta aika-arvioiden arpomiseen meni liikaa aikaa ja paperi-kynä-systeemillä merkittyjen arvioiden päivittäminen oli hidasta. Nykyinen systeemi on sellainen, että aikaa arvioidaan vain sillä, että montako tehtävää on mukana. Tehtävät on pilkottu niin pieniin osiin, että yhden tehtävän tekemiseen ei menisi yhtä työpäivää enempää.” (C1)

Kun toteutettavat tehtävät valitaan sprintin suunnittelukokouksessa tuotteen tehtävälistalta sprintin tehtävälistalle, ne eivät tule tuotteen tehtävälistalta sellaisenaan, vaan niitä tarkennetaan ja niiden ajallista kestoa arvioidaan kaikissa yrityksissä tuntitasolla, kuten Scrumin teoriassa on aikaisemmin määriteltykin (Schwaber 2004). Tärkeintä tässä prosessissa on työtehtävien konkretisoituminen, koska usein tuotteen tehtävälistan tehtävät ovat hyvin suurpiirteisiä kokonaisuuksia tai esimerkiksi käyttötapauskuvauksia (B1, C1). ”Tehtävien pilkkominen ei ole tässä se varsinainen arvo, vaan se, että tässä vaiheessa se työ suunnitellaan.” (A2) Sprintin tehtävälista toimii tiimillä kommunikoinnin välineenä: se auttaa tiimin jäseniä suunnittelemaan tehtäviä paremmin ja laatimaan parempia vaatimusmäärittelyjä (A2). Sprintin tehtävälistan laatiminen on hyvä tapa analysoida sitä, että ”mitä kaikkea tässä tuotteen tehtävälistan tehtävän tekemisessä itse asiassa pitääkään tehdä.” (A2)

Aika-arvioiden tekeminen tapahtui suurimmassa osassa yrityksiä ihan vain tiimin kesken tapaamalla ja juttelemalla, mutta yhdessä yrityksessä arvioiden tekemiseen käytettiin suunnittelupokeria (engl. planning poker):

> ”Kahta päivää ennen sprintin vaihtumista pidetään estimointitapaaminen, jossa tiimi istuu pelaamaan Planning Pokeria jos tuotteen tehtävälistassa on joitakin sellaisia tehtäviä, joita ei ole vielä arvioitu.” (C1)

Suunnittelupokerissa tehtäville annetaan suhteellinen pisteytys, eikä tunteihin perustuvaa arviota.

> ”Tästäkin huolimatta tehtävien arviointiin menee liian paljon aikaa, joten yrityksessä aiotaan muokata tätä vielä siten, että tehtävät arvioidaan vain ns. T-paita-asteikolla. Tehtävien kooksi voidaan siis sanoa joko S, M tai L.” (C1)

Kun sprintti on saatu yrityksissä rullaamaan, näyttäisi siltä, että sprintin tehtävälistan ääreen palataan joko keskimäärin kerran päivässä (A1, A2, D1) tai

sitten vain muutaman kerran sprintin aikana (B1, D2). Asiakkaan ei pitäisi päästä vaikuttamaan enää sprintin kuluessa sprintin tehtävälistan tehtäviin, mutta yrityksessä B tässä on jouduttu taipumaan ja aktiivisen asiakkaan suunnalta lisäyksiä tulee sprintin tehtävälistaan vielä noin kerran tai kaksi sprintin aikana:

> ”Tiimi on pääosin vastuussa tehtävien päivittämisestä, mutta tosiaan asiakkaan suunnalta muutosehdotuksia tulee myös. Tiimillä on kyllä oikeus päättää, otetaanko muutosehdotukset mukaan sprinttiin vai ei, mutta käytännössä asiakas saa yleensä tahtonsa läpi.” (B1)

Sprintin tehtävälistasta haastateltavilla oli paljon hyvää sanottavaa, kuten korkealle arvioidusta hyödyllisyysasteestakin voi päätellä. Päällimmäisenä hyvänä asiana sprintin tehtävälistassa nähtiin se, että se antoi selkeän kuvan tiimille siitä, mitä heiltä minäkin päivänä odotetaan, mikä on sprintin tavoite ja projektin vetäjien kannalta sprintin tehtävälista tarjoaa reaaliaikaisen ja konkreettisen mittarin sille, missä vaiheessa tehtävien tasolla ollaan sprintissä menossa (A1, A2, B1, C1, D1). Kun tiimillä on tavoite kirkkaana mielessä ja selkeät kuvaukset päivittäisistä työtehtävistä, kehittäjät ovat vähemmän stressaantuneita (D1). Tällöin tekeminen ei myöskään ala rönsyilemään (D1), eikä kenelläkään tiimin jäsenellä pitäisi olla liikaa työtehtäviä (A1). Jos projektissa pystytään pitämään kiinni siitä, että sprintin tehtävälistaan ei koske sprintin aikana enää kukaan muu kuin tiimi itse, tehtävälista tuo rauhaa työskentelyyn ja parantaa tiimin keskittymistä (C1).

Sprintin tehtävälistan hyödyntäminen tarkoittaa myös sitä, että tiimin tehtävät löytyvät aina yhdestä paikasta. Yrityksessä B sprintin tehtävälista on fyysisesti esillä tauluna projektitiloissa, jolloin sen seuraaminen on koettu helpommaksi kuin jos se olisi esimerkiksi yrityksen intranetissä. Yrityksessä todettiin, että ”tikettien” tekeminen oli hitaampaa ja tehottomampaa kuin nykyinen tehtävien merkitseminen post-it lapuille. Laput jaotellaan neljään eri lokeroon

niiden valmiusasteen mukaan: ”ei vielä aloitetut / työn alla / vaatii testausta / tehdyt”. Tältä taululta näkee myös poistetut tehtävät, työssä esiintyneet esteet (engl. impediments) sekä sprintin edistymiskäyrän (engl. burndown chart). (B1)

Kysyttäessä huonoja puolia sprintin tehtävälistasta, mainitut ongelmat liittyivät osittain jo sprintin suunnittelukokouksen yhteydessä mainittuun asiakkaan liiankin innokkaaseen osallistumiseen projektiin (A1, B1, C1, D2). ”Asiakas puuttuu hyvin usein sprintin kulkuun ja haluaa uusia toiminnallisuuksia tai muuta vastaavaa kesken sprintin.” (D2) ”Asiakas ei aina ymmärrä sitä, että mitä hyötyä siitä on, ettei siihen [sprintin tehtävälistaan] kosketa.” (C1) Tehtävien lisääminen kesken sprintin paitsi hidastuttaa sprintin edistymistä, on myös erittäin vaativaa tehtävien uudelleenpriorisoinnin kannalta (A1, D2). Yhdellä haastateltavista oli huonoja kokemuksia sprintin tehtävälistan tehtävien liiallisesta pilkkomisesta:

> ”Jos tehtävät jaetaan todella pieniin osiin ja niiden sisältö on aika rajattua, niin silloin tehtäviä toteutetaan aika mekaanisesti ’laput silmillä’, eikä muisteta pitää silmällä sitä kokonaiskuvaa niin hyvin. Tämä ohjaa niin tarkasti johonkin tiettyyn työntekotyyliin, että taitavampien tyyppien luovuudelle ja osaamiselle ei ehkä ole niin paljon tilaa. Tämä tosin on hyvin pieni huono puoli ja on helposti ratkaistavissa sillä, että tehtävien liiallista pilkkomista vältetään.” (D1)

Onnistuneen sprintin suunnittelu ja sprintin tehtävälistan kokoaminen vaatii tiimiltä harjoitusta, jotta sprintissä ei ole liikaa, eikä liian vähän tekemistä (C1) ja että sprintin tehtävälistan koostamiseen ei mene liian paljon aikaa (D2). ”Oppimiskäyrä on tässäkin asiassa pitkä, mitä pidempään ihmiset ovat olleet projekteissa mukana, sitä helpommin esimerkiksi aika-arvioiden tekeminen sujuu.”(A2)

5.2.5 Päivittäinen Scrum-tapaaminen

Scrum-tiimi ja Scrum-mestari kokoontuvat yhteen joka päivä noin 15 minuutin ajaksi *päivittäiseen Scrum-tapaamiseen* (engl. Daily Scrum), jossa tiimin jokainen jäsen vastaa omalta kohdaltaan seuraaviin kysymyksiin:

- Mitä olet tehnyt viimeisen Scrum-tapaamisen jälkeen?
- Mitä teet ennen seuraavaa tapaamista?
- Mitä vaikeuksia tai esteitä on ollut työssäsi?

Tapaamisen tarkoituksena on pitää Scrum-mestari ja muu tiimi ajan tasalla siitä, mitä projektissa ollaan sillä hetkellä tekemässä (Boehm & Turner 2004). Scrum-mestari saa myös tietää (Boehm & Turner 2004) viimeistään tässä vaiheessa, jos tiimillä tai jollakin sen jäsenistä on ongelma, jonka ratkaisemisessa tarvitaan ulkopuolista apua. (Schwaber & Beedle 2002)

Scrum-mestarin tehtävänä on valvoa, että päivittäiset Scrum-tapaamiset eivät veny turhaan liian pitkiksi (Schwaber & Sutherland 2010). Yksi tapa pitää kokousten pituus kurissa on pitää päivittäiset Scrum-tapaamiset nk. stand-up – tapaamisina (engl. Daily Stand-up), eli kaikki osallistujat seisovat koko tapaamisen ajan (Sutherland, Viktorov, Blount & Puntikov 2007). Koska Scrum-tapaamiseen saa tulla kuka vain halutessaan, on Scrum-mestarin myös varmistettava, etteivät mahdolliset ulkopuoliset osallistujat häiritse tapaamisen etenemistä (Schwaber & Sutherland 2010).

Aikaisemmat empiiriset tutkimukset

Yleinen mielipide päivittäisestä Scrum-tapaamisesta on pääosin positiivinen aiemmissa empiirisissä tutkimuksissa. Tapaamisen mainittiin parantavan tiimin yhteenkuuluvuuden tunnetta (Grossman, Bergin, Leip ym. 2004) ja tietoisuutta muiden tiimin jäsenten edistymisestä (Begel & Nagappan 2007). Scrum-tapaamisen ansiosta kaikki olivat perillä siitä, mitä kukakin tekee sprintin tavoitteen saavuttamiseksi (Begel & Nagappan 2007). Tapaamisen ansiosta tii-

min jäsenillä oli myös parempi yleiskuva projektin tilanteesta (Paasivaara, Durasiewicz & Lassenius 2008) tai koodista (Li, Moe & Dybå 2010) ja ihmiset kommunikoivat keskenään aiempaa enemmän (Begel & Nagappan 2007, Li, Moe & Dybå 2010, Paasivaara, Durasiewicz & Lassenius 2008). Lin, Moen ja Dybån (2010) yli kolme vuotta kattaneessa tapaustutkimuksessa päivittäisen Scrum-tapaamisen kerrottiin auttavan virheiden korjaamisessa ja havaitsemisessa, koska asioista keskusteltiin päivittäin ja virheet tulivat esille siten nopeammin. Näin ongelmalliset kohdat voitiin myös korjata nopeammin (Li, Moe & Dybå 2010). Tutkimuksessa todettiin, että ennen Scrumin käyttöönottoa kehitettävän ohjelmiston virheet saatettiin huomata vasta hyvin myöhäisessä vaiheessa projektia, jolloin ohjelmoijilla ei ollut enää tuoretta muistikuvaa siitä koodin osasta, johon virheellinen tilanne voisi liittyä ja aikaa meni turhaan muistelemiseen projektin aikana (Li, Moe & Dybå 2010). Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) tutkimuksessa päivittäinen Scrum-tapaaminen oli hajautetuissa tiimeissä työskentelevien haastateltavien mielestä hyödyllisin Scrumin käytänteistä, koska se toi läpinäkyvyyttä kahden toimipaikan välille ja paljasti projektin ongelmat ajoissa.

Päivittäisten Scrum-tapaamisten venähtäminen liian pitkiksi on mainittu käytänteen ongelmalliseksi puoleksi useammassa aiemmassa empiirisessä tutkimuksessa (Begel & Nagappan 2007, Mann & Maurer 2005). Begelin ja Nagappanin (2007) tekemässä kyselytutkimuksessa mainittiin, että joistakin tiimin jäsenistä päivittäinen Scrum-tapaaminen tuntui ennemmin tapaamiselta, jonka tarkoituksena oli raportoida projektin johdolle, mitä kukin on viimeisen vuorokauden aikana saanut tehdyksi (Begel & Nagappan 2007). Tiimeissä, joissa on mukana ihmisiä eri kulttuureista, voi jollekin tiimin jäsenistä oman työn haasteista kertominen olla hyvin vierasta ja vaikeaa, jolloin projektin johto taas ei pysty reagoimaan näihin ongelmiin tarpeeksi ajoissa (Paasivaara, Durasiewicz & Lassenius 2008).

Haastattelututkimuksen tulokset

”[Päivittäinen Scrum-tapaaminen on] rajoitettu 15 minuuttiin. Lyhyt tiimin tapaaminen tilanteen selvittämiseksi, jossa tiimin jäsenet synkronoivat tehtävänsä ja edistymisensä sekä raportoivat mahdolliset esteensä Scrum-mestarille, jotta niihin voidaan puuttua.” (Schwaber 2004, Appendix B)

Päivittäinen Scrum-tapaaminen -hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	1	3	1	3	2,3

Suurin osa haastateltavista totesi, että tapaamisen kulku menee juuri määritelmän mukaisesti. Yrityksessä C kysytään henkilökohtaisten kysymysten lisäksi tiimin jäseniltä yhteisesti, ollaanko sprintissä aikataulussa. Tähän kysymykseen pitäisi pystyä vastaamaan kuka tahansa tiimin jäsenistä (C1). Haastateltavan D1 projekteissa päivittäinen Scrum-tapaaminen ei ole päivittäistä, koska tapaamiset ovat yhteisiä useammalle projektille. Järjestely johtuu siitä, että samat ihmiset työskentelevät usein useammassa pienessä projektissa samanaikaisesti (D1). Samasta syystä johtuen näissä tapaamisissa ei myöskään käytetä siihen yleensä liittyviä kolmea kysymystä (D1). Muut haastateltavat kertoivat, että tapaamisissa kolme kysymystä ovat kyllä käytössä, mutta yrityksessä B kysyttäessä oman työn haasteita, niitä ei kovin usein mainita. Yrityksen B tiimeissä jäseniä löytyy useammasta eri kansallisuudesta, jolloin syy saattaa olla samankaltainen kuin mitä Paasivaara, Durasiewicz ja Lassenius (2008) aiemmin mainitussa tutkimuksessaan havaitsivat.

Käytänteen hyväksi puoleksi mainittiin mm. se, että tiimi ja tiimin vetäjä / Scrum-mestari pysyvät hyvin perillä siitä, missä projektissa mennään, kun tiimin tekemisiä käydään läpi päivittäin (A1, B1, C1, D2). Päivittäinen Scrum-tapaaminen auttaa tiimiä todella pääsemään yhteiseen päämäärään (A2). Työssä olevien esteiden selvittäminen avoimesti muulle tiimille on koettu myös hyväksi asiaksi tässä käytänteessä (A1, A2, C1, D2).

> ”Kun kokoonnutaan yhteen kerran päivässä, se suo keskitetyn tilaisuuden kertoa työssä eteen tulleista esteistä. Kun esteistä oikein kysymällä kysytään, niin niihin on helpompi vastata kuin jos esteen ilmetessä se täytyisi lähteä erikseen kertomaan jollekin. Joskus nämä esteet ovat myös tiedostamattomia, jolloin vasta päivittäisessä Scrum-tapaamisessa Scrum-mestari tajuaa jollakin tiimin jäsenellä olevan ongelmia, vaikkei tiimin jäsen itse sitä heti tajuaisikaan.” (C1)

Tapaaminen toimii myös kannusteena työnteolle, kun tiimin jäsen tietää joutuvansa raportoimaan tehdyt asiat seuraavana päivänä muille (B1) ja joutuu tavallaan lupaamaan muulle tiimille seuraavaksi tehtävät asiat (C1).

Haastateltava D1 totesi, että hänellä ei ole mainittavia hyviä puolia käytänteestä. Kuten jo aikaisemmin mainittiin, heillä päivittäiseen Scrum-tapaamiseen osallistuu muista haastateltavien tilanteista poiketen tiimin jäseniä useasta pienemmästä projektista. Tapaamisten luonnetta ei ole määritelty tarkasti, joten haastateltava toteaa, ettei niistä ole heille niin paljon hyötyä. Toinen oleellinen ero muihin projekteihin oli Scrum-mestarin puuttuminen kokoontumisen kokoonpanosta. Tämä vaikuttaa niin kokoontumisten säännöllisyyteen, kuin ongelmatilanteiden selvittelyynkin.

> ”Koska tapaamiset pidetään vapaalla aikataululla ja sovitaan vain sisäisesti, ei niitä tule sitten aina pidettyä, varsinkin jos osallistujia ei ole tarpeeksi. Ja kun tapaaminen pidetään pienen projektiryhmän kesken ja joku tarvitsisi sellaista apua, jossa muut projektin jäsenet eivät voi auttaa, on vaikea etsiä apua sitten muista projektiryhmistä.” (D1)

Haastateltavan mukaan tapaamisen osallistujilla on joskus vaikeuksia pysytellä aiheessa, mutta tämä taas johtui varmaankin siitä, ettei tapaamisessa hyödynnetty kolmea peruskysymystä (D1). Haastateltava B1 oli toinen, joka määritteli käytänteen hyödyllisyysasteen muita haastateltuja alhaisemmaksi. Tähän hän mainitsee syyksi yrityksen työskentelyolosuhteet:

> ”Tätä käytännettä ei ole koettu ihan niin hyödylliseksi, koska tiimin jäsenet istuvat niin lähellä toisiaan, että he tietävät kyllä, missä mennään ilman päivittäistä kokoustakin. Ongelmien kanssa ei jäädä odottelemaan, vaan niihin haetaan apua ilman päivittäistä Scrum-tapaamistakin. Tästä syystä tapaamisia saatetaan harventaa tulevaisuudessa tai käytännettä muokata jotenkin muuten parempaan ja toimivampaan suuntaan.” (B1)

Muut haastateltavat kokivat päivittäisen Scrum-tapaamisen vaikuttavan pääosin positiivisesti heidän projekteihinsa, joten merkittäviä huonoja puolia ei mainittu. Projekteissa esiin tulleet, käytännettä koskevat negatiiviset asiat ovat hoituneet joko puhumalla tai ajan kanssa kokemuksen karttuessa:

> ”Yhden tiimin jäsenen mielestä päivittäinen Scrum-tapaaminen tuntui siltä, että siinä projektipäällikkö hengittää niskaan. Asiasta on kuitenkin selvitty, kun tiimin jäsenille on selvitetty, mikä tapaamisen tarkoitus todella on.” (A1)

> ”Kokeneen ja kokemattoman tiimin välinen ero on tässäkin käytänteessä suuri. Kokemattomat tiimit keskittyvät enemmän omaan tehtäväänsä, mutta kokeneet tiimit oikeasti katsovat mittareita, suunnittelevat ja miettivät, miten saadaan se sprintin tavoite toteutettua, eli todellakin synkronoivat tekemisensä.” (A2)

Haastateltava D2 mainitsi myös, että heillä päivittäisen Scrum-tapaamisen aamuinen ajankohta tuotti joskus tiimin jäsenille ongelmia, jolloin välillä tiimin jäsenten puuttumisen takia tapaamisia jouduttiin perumaan.

Koska aikaisemmissa tutkimuksissa mainittiin tapaamisten usein venyvän yli tämän sallitun 15 minuutin aikarajan, haastateltavilta tiedusteltiin, venyvätkö tapaamiset sekä käsitelläänkö ilmenneitä ongelmia myöhemmin jossakin jatkopalaverissa. Suurimmalle osalle haastateltavista palaverien venyminen ei ollut liian usein toistuvaa tai muuten ongelmallista (A2, B1, C1, D2). Haastateltava A1 totesi, että jos jollakin tiimin jäsenistä on ollut esteitä työssään, asia halutaan usein käsitellä heti päivittäisessä Scrum-tapaamisessa tai välittömästi sen jälkeen (A1). Ylimääräisiä kokouksia ei juurikaan tämän lisäksi pidetty (A1).

Yrityksessä C on venymisen estämiseksi käytetty jo teoriassa aiemmin mainittua stand-up tapaamista, jossa siis tapaamisen osallistujat seisovat koko tapaamisen ajan. Yrityksessä kokeiltiin myös muunlaisia tapoja pitää tapaamisen sisältö tarpeeksi tiiviinä: ”Esimerkiksi munakelloa on käytetty mittaamaan kahden minuutin aika, jonka piti riittää yhden tiimin jäsenen asioiden kertomiseen.” (C1) Jatkopalaverin tarvetta arvioitiin yrityksissä järkevästi päivittäisen Scrum-tapaamisen aikana ja se pidettiin tarvittaessa tiimin kesken esimerkiksi tapaamisen päätteeksi ”riippuen vähän siitä, että pystyykö tiimi tekemään asialle jotain, vai pitääkö siihen ongelmaan hakea ulkopuolista apua. Jos asia voidaan selvitellä tiimin kesken, niin asia saatetaan käsitellä päivittäisen Scrum-tapaamisen yhteydessä.” (B1)

5.2.6 Sprintin katselmointitapaaminen

Sprintin katselmointitapaamisessa (engl. Sprint Review meeting) tärkein tarkoitus on tarkastella kuluneen sprintin aikana aikaansaatuja tuloksia, sen aikana esiin tulleita ongelmia ja sille asetettuja tavoitteita siten, että niiden avulla voidaan nähdä, mihin suuntaan projekti on menossa. Sprintin katselmointitapaaminen on tarkoitus pitää aina sprintin viimeisenä päivänä, ja siihen saavat osallistua tiimin lisäksi kaikki projektiin liittyvät tahot: asiakas, loppukäyttäjät, tuotteen omistaja ja projektin johdon edustajat. (Schwaber & Beedle 2002)

Scrumin hengen mukaisesti tässäkin tapaamisessa pyritään välttämään kaikkea turhaa tekemistä, joten Schwaberin ja Beckin (2002) mukaan sprintin katselmointitapaamiseen valmistautumiseen ei saa käyttää missään nimessä kahta tuntia kauempaa. Tuloksia ei esitellä minkään esitysgrafiikan avulla, vaan ne käydään läpi konkreettisesti näyttämällä muille, mitä toiminnallisuuksia tuotteeseen on sprintin aikana lisätty. Tämä auttaa asiakasta ymmärtämään kehitettävää tuotetta paremmin ja antaa tällöin myös hyvän pohjan seuraavan sprintin suunnitteluun. (Schwaber & Beedle 2002)

Aikaisemmat empiiriset tutkimukset

Sprintin katselmointitapaamista koskevia aikaisempia tutkimuksia on muiden Scrumiin liittyvien käytänteiden tavoin hyvin niukasti. Mannin ja Maurerin (2005) tapaustutkimuksessa mainittiin positiiviseksi puoleksi se, että katselmointitapaaminen auttoi hahmottamaan kehitteillä olevaa projektia paremmin ja katselmointien aikana ohjelmoijien oli helppo levittää tietoa myös muille tiimin jäsenille. Tapaamisten ajallinen venyminen raportoitiin katselmointitapaamisen huonoksi puoleksi (Mann & Maurer 2005). Berczukin (2007) empiirisessä tutkimuksessa tarkkailtiin hajautetun tiimin sopeutumista Scrumiin ja katselmointitapaaminen koettiin positiiviseksi asiaksi projektissa. sprintin katselmointitapaaminen tarjosi selkeän kuvan siitä, missä projektissa ollaan todella menossa ja vastaavatko toiminnallisuuksien vaatimukset toteutettua tuotetta (Berczuk 2007). Berzuk (2007) mainitsee myös, että vaikka kaikki katselmointitilaisuudet eivät aina menneet suunnitelmien mukaan, ne loivat asiakkaaseen luottamusta tiimin taitoihin.

Haastattelututkimuksen tulokset

”[Sprintin katselmointitapaaminen on] rajoitettu 4 tuntiin. Jokaisen sprintin lopussa tiimi esittelee tuotteen omistajalle sekä muille kiinnostuneille tahoille, mitä se on saanut aikaiseksi kuluneen sprintin aikana. Vain valmiita toiminnallisuuksia voidaan demonstroida.” (Schwaber 2004, Appendix B)

Sprintin katselmointitapaamisen hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	2	2	3	-	2,6

Sprintin katselmointitapaaminen koettiin joko hyödylliseksi tai erittäin hyödylliseksi projektille. Haastateltava D2 ei vastannut kysymyksiin lainkaan, koska web-kehityksen luonteesta johtuen asiakas pystyi testaamaan tuotetta itse internetin välityksellä, jolloin varsinaisia katselmointeja ei projektissa pidetty. Haastateltavat eivät kertoneet merkittäviä eroja oman katselmointitapaamisen ja määritelmän välillä. Katselmointitapaamisten ajalli-

nen kesto ei ollut läheskään niin pitkä kuin mitä määritelmässä kerrotaan, vaan tapaamiseen saatettiin käyttää vain noin tunnin verran aikaa (B1, C1). ”Katselmointitapaamisen jälkeen on yrityksessä järjestetty aikaa vielä asiakkaan ideoinnille, koska ne parhaat ideat tulevat katselmoinnin jälkeen.” (C1) Kaksi haastateltavaa kertoi, että heillä katselmointitapaamisessa voidaan yhtä hyvin käydä läpi myös keskeneräisiä toiminnallisuuksia, koska niistäkin voidaan saada asiakkaalta arvokasta palautetta (A1, B1).

Sprintin katselmointitapaamiseen osallistui yleensä tiimi, asiakas, projektin johdon edustaja sekä tuotteen omistaja. Kysyttäessä kunkin osallistujan roolia, eivät vastaukset eronneet juurikaan toisistaan, vaan yleensä kukin tiimin jäsen esitteli vuorollaan oman osuutensa tuotoksesta, joista tuotteen omistaja ja/tai asiakas saattoi tehdä sitten esityksen perusteella tarkentavia kysymyksiä. Scrum-mestarin harteille jäi kokouksen yleisen kulun seuranta ja siitä varmistuminen, että tapaamisessa pysytään asiassa (A1, A2, C1, D1). Yrityksessä B käytännettä on muokattu kokemuksen karttuessa siten, että tiimin jäsenet eivät kerro yksitellen asioita, joita he ovat itse olleet kehittämässä, vaan esittelystä vastaa testaaja (B1).

> ”Testaaja on kuitenkin joutunut testaamaan esiteltävän ohjelmiston osan kokonaisuudessaan, joten hän on parhaiten perillä siitä kokonaisuutena. Aikaisemmin ongelmana oli myös se, että katselmointeihin ei juurikaan valmistauduttu, mutta nyt kun testaaja käy kaiken läpi, hänellä on paremmat mahdollisuudet myös miettiä asioiden järkevää esittämistapaa ja -järjestystä.” (B1)

Kaikki haastateltavat kertoivat, että sprintin katselmointitapaamisessa tuotetta käydään läpi Scrumin teorian mukaisesti konkreettisesti demoamalla, eli käyttämällä toteutettua ohjelmistoa ja näyttämällä näin toteutetut toiminnallisuudet. Usein demoamisen tukena toimii kuluneen sprintin tehtävälista, jonka avulla näytetään, miten mikin tehtävälistan tehtävä on muuttunut

sprintin aikana ohjelman toiminnallisuudeksi (A1, A2, B1, C1, D1). Asiakkaan näkökulmasta katsottuna katselmoinnit ovat olleet mieluinen tapa käydä tilattua tuotetta läpi ja näin nähdä, mitä kehityksessä tapahtuu.

Käytänteen hyvinä puolina nähtiin mm. paremman yhteyden saaminen asiakkaaseen, koska katselmointitapaamisten ansiosta asiakasta nähdään säännöllisin väliajoin, asiakkaalta saadaan arvokasta palautetta tehdystä työstä ja itse asiakaskin on paremmin perillä kehitettävän tuotteen tilanteesta (A1, A2, B1, C1, D1). Tuotteen läpikäyminen asiakkaan kanssa toteutettuja toiminnallisuuksia esitellen antaa myös asiakkaalle vinkkejä siihen, minkälaisia vaatimusmäärittelyjä kehittäjät häneltä odottavat:

> ”Vaikka kuinka hyvin asiakas kertoisi etukäteen, mitä hän on halunnut tuotteelta, tulee paras palaute ja parhaimmat ideat vasta sen jälkeen, kun katselmoinnissa on käyty läpi valmiiksi saatuja vaatimuksia. Tämän jälkeen asiakas osaa vielä paremmin ilmaista oikeat tarpeensa.” (C1)

Useampi haastateltava mainitsi myös sen, että sprintin katselmointitapaaminen luo tiimille tietyn takarajan ja kannustaa näin tekemään tuotteen todella siihen pisteeseen, että sen kehtaa hyvillä mielin asiakkaalle esitellä (B1, C1, D1). Lisäksi ”sisäisen stressinhallinnan ja työnohjauksen työkaluna sprintin katselmointitapaaminen on toimiva.” (D1)

Huonoja puolia kysyttäessä kaksi haastateltavista ei osannut mainita mitään huonoa puolta käytänteestä. Kaksi vastanneista totesi, että asiakkaan saaminen paikanpäälle on haastavin asia käytänteeseen liittyen, koska kasvotusten tapaaminen on asiakkaalle kallista (D1) ja kahden viikon välein tapaaminen on hyvin aikaavievää (C1). Lievänä miinuspuolena mainittiin myös se, että tiiviisti yhteistyötä tekevässä tiimissä kaikki tiimin jäsenet kyllä tietävät, mitä tuote pitää sisällään, joten lähinnä asiakkaalle suunnatussa katselmointitapaamisissa istuminen voi tiimin mielestä tuntua välillä turhauttavalta (B1).

5.2.7 Sprintin arviointitapaaminen

Sprintin arviointitapaamisen (engl. Sprint Retrospective meeting) tarkoituksena on jatkuvasti parantaa ja tehostaa projektin etenemistä oppimalla kuluneen sprintin aikana tehdyistä virheistä ja onnistumisista. Sprintin arviointitapaaminen liittyy eritoten Scrumin hallinnollisiin käytänteisiin. (Schwaber 2004)

Sprintin arviointitapaamiseen voivat osallistua Scrum-mestari ja kaikki Scrum-tiimit, jotka työskentelevät saman projektin ja sprintin parissa. Tapaamisessa jokainen tiimin jäsen miettii kulunutta sprinttiä ja kertoo muille, mitä parannettavaa heidän mielestään sprintissä oli. Myöhemmin arviointitapaamisessa ainakin kriittisimmille ongelmallisille asioille pyritään löytämään yhdessä jokin ratkaisu, jotta seuraavassa sprintissä työskenteleminen olisi jouhevampaa. Huonosti menneiden asioiden lisäksi jokainen tiimin jäsen kertoo, mikä meni sprintissä hyvin. Tämä edesauttaa hyväksi havaittujen työskentelytapojen ja muun tietotaidon leviämistä myös muihin Scrum-tiimeihin ja Scrum-mestarin kautta mahdollisesti myös muihin organisaatiossa toteutettaviin projekteihin. (Schwaber 2004)

Aikaisemmat empiiriset tutkimukset

Aikaisemmin julkaistuissa tutkimuksissa tuotiin käytänteen hyvänä puolena esiin se, että tapaamisissa ohjelmoijat saivat arvokasta palautetta ja parannusehdotuksia kanssakoodaajilta (Mann & Maurer 2005) sekä projektia tulee mietittyä sprintin arviointitapaamisen ansiosta ihan toisella tavalla: ”Toimitimmeko tosiaan sen, minkä lupasimmekin? Korjasimmeko kaikki puutteet? Näihin ongelmiin keskityttiin aiempaa enemmän koko projektin tasolla.” (Li, Moe & Dybå 2010, 13:5). Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) haastattelututkimuksessa kerrottiin niin ikään tiimien käsitelleen sprintissä hyvin ja huonosti menneitä asioita, mutta näihin havaintoihin ei kuitenkaan reagoitu kovin vahvasti. Sprintin katselmointitapaamisista saatu tieto ei myöskään

levinnyt tehokkaasti eri tiimien välillä, joten tiimit kehittelivät siirtymälokin (engl. transition backlog), jonne sprintin arviointitapaamisessa esille nostetut asiat kirjattiin ylös kaikkien näkyville (Paasivaara, Durasiewicz & Lassenius 2008).

Haastattelututkimuksen tulokset

"[Sprintin arviointitapaaminen on] rajoitettu 3 tuntiin. Scrum-mestarin johtama tilaisuus, jossa tiimi keskustelee juuri päättyneestä sprintistä ja määrittelee, mitä voitaisiin muuttaa, jotta seuraava sprintti olisi miellyttävämpi tai tehokkaampi." (Schwaber 2004, Appendix B)

Sprintin arviointitapaamisen hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	2	3	3	3	0	2	2,2

Useimmiten erona tähän määritelmään mainittiin tapaamisen kesto, joka vaihteli 15 minuutista 1,5 tuntiin. Määritelmä sprintin arviointitapaamisesta ei rajaa tilaisuuden luonnetta kovinkaan tarkasti, joten haastatteluissa tulikin esille hyvin erilaiset tavat lähestyä ja käsitellä tätä käytännettä: sprintin arviointitapaamista sovellettiin ehkä yhtä monella eri tavalla kuin mitä oli haastateltaviakin.

"Tää on sellainen ehkä, mikä vaatii semmoista sosiaalista älyä eniten Scrum-mestarilta." (A2) Haastateltavan kommentti sivuaa sitä asiaa, että sprintin arviointitapaaminen on Scrumin käytänteistä ehkä yksi arimmista sosiaalisessa mielessä, koska siinä otetaan suurennuslasin alle kuluneen sprintin tapahtumat "ihmisten, muodostuneiden ihmissuhteiden, prosessien ja työkalujen näkökulmasta" katsottuna (Schwaber & Sutherland 2010, 15). Scrum-mestarin taidosta riippuu siis paljonkin se, millä tasolla asioita pystytään tiimissä käsittelemään. Jos tiimin tunteita ja kokemuksia yritetään kalastella esille soveltamalla esimerkiksi sellaisia lähestymistapoja, jotka eivät sovi suomalaiseen kulttuuriin, "se saattaa tuntua tosi naiivilta ja hölmöltä toimia jotenkin niin tai kerätä sitä palautetta jollain sellaisella tyylillä." (A2) Haastateltava A2 oli haastateltavista ainoa,

joka kertoi sprintin arviointitapaamisen olevan useimmin epävirallisempi tapaaminen, jossa yleensä mennään johonkin yhdessä istumaan ja pohtimaan asiaa, jolloin Scrum-mestarin rooli korostuu entisestään. Muut haastateltavat kertoivat sprintin arviointitapaamisen olevan enemmän virallinen kuin epävirallinen tapahtuma, joka toteutettiin usein heti sprintin katselmointitapaamisen jälkeen (A1, B1, D1) tai seuraavan sprintin suunnittelukokouksen yhteydessä (D2). Yrityksessä C sprintin arviointitapaaminen pidetään ihan vain palaverina ja asialle on varattu aikaa noin puoli tuntia (C1).

Kysyttäessä haastateltavilta, toteutuuko arviointitapaaminen jokaisen sprintin jälkeen, vain kaksi vastaajaa antoi kieltävän vastauksen (A2, D1). Haastateltava A2 selvitti tämän syyksi joidenkin tiimien kokemattomuuden:

> ”Jos tiimi on kovin kokematon, niin sitten kaikki tällaiset ylimääräiset käytännöt vaan lisää sitä vaikeutta ja tuskaa siinä. Kannattaa alussa aloittaa mahdollisimman helpoista ja pienistä ja sitten lisätä niitä elementtejä.” (A2)

Haastateltava D1 kertoi, että sprintin arviointitapaamista ei ole pidetty enää pitkään aikaan missään projektissa. Tämän takana on oletettavasti haastateltavan negatiiviset kokemukset arviointitapaamisista: tapaamisissa esiin tulleisiin kehitysehdotuksiin ei reagoida tarpeeksi hyvin ja tehokkaasti, eikä asioiden ylös kirjaaminenkaan auta siirtämään arviointitapaamisissa kertynyttä tietotaitoa eri projektien ja ihmisten välillä (D1). Tällöin on täysin ymmärrettävää, että tiimin jäsenet näkevät sprintin arviointitapaamisen turhana ja ylimääräisenä taakkana sprintin lopussa. Haastateltavan D1 näkemys käytänteen haasteista on hyvin samankaltainen kuin mitä Paasivaara, Durasiewicz ja Lassenius (2008) toivat esille omassa tutkimuksessaan. Haastateltavan A2 mukaisesti uskon, että tässäkin tapauksessa ”tämä on eniten Scrum-mestarista kiinni, miten hyvin se [sprintin arviointitapaaminen] onnistuu”.

Tiedonsiirto tiimiltä toiselle ja projektien välillä on osa sprintin arviointitapaamisen tarkoitusta sen lisäksi, että tiimi oppisi itse omista virheistään ja onnistumisistaan. Mutta se, miten tämä tiedon ja tietotaidon siirtyminen toteutetaan, on aivan oma juttunsa. Yrityksissä oli monenlaisia tapoja kirjata arviointitapaamisissa esitettyjä parannusehdotuksia ja kompastuskiviä. Virallisin tapa oli kirjoittaa arviointitapaamisesta muistio, joka lähetettiin tapaamisen jälkeen vielä sähköpostitse kaikille osallistujille (yleensä tapaamisessa mukana tiimin ja Scrum-mestarin lisäksi myös asiakas) (A1). Haastateltava A1 kertoi kuitenkin, että ”useimmiten ei tule konkreettisia muutosehdotuksia, joten tapaaminen toimii silloin eräänlaisena fiilismittarina, että miten projektissa menee”. Jos muutosehdotuksia tuli esille, ne pyrittiin ottamaan mahdollisimman nopeasti käyttöön (A1). Haastateltava A2, jonka projekteissa arviointitapaamisen luonne on muihin haastateltujen kertomuksiin verrattuna rennompi, kertoo Scrum-mestarin tekevän näistä tapaamisista kuitenkin itselleen muistiinpanot, joiden mukaan hän pystyi toimimaan tulevissa sprinteissä paremmin (A2). Useimmiten tapaamisista kerätty tieto kirjoitettiin kuitenkin jonnekin yrityksen sisäiseen intraan tai wikiin, josta tieto on saatavilla yrityksen työntekijöille (B1, D1, D2). Näissä ongelmana on kuitenkin se, että näitä dokumentteja tulee harvemmin kuitenkaan kenenkään kovin aktiivisesti luettua. Yrityksessä C tämä on ratkaistu sijoittamalla ihan konkreettinen lista toimivista ja toimimattomista projektien käytänteistä tiimin yleisiin tiloihin kaikkien nähtäville, jolloin sitä tulee myös paremmin seurattua (C1). Tämän lisäksi projektien välistä tiedonsiirtoa parannettiin pitämällä uutta projektia aloittaessa pieni tilaisuus, jossa käydään läpi muissa projekteissa opittuja asioita (C1). Ratkaisu on samansuuntainen Paasivaaran, Durasiewiczin ja Lasseniuksen (2008) havaintojen kanssa.

Sprintin arviointitapaamisen hyviksi puoliksi haastateltavat toivat esille asioita laidasta laitaan. Sprintin arviointitapaamisen rooli hyvänä prosessin kehittämi-

sen työkaluna tuli ilmi useammassa haastattelussa (A2, B1, C1, D2). Käytänteen avulla ”saatiin asiat tuotua esille niin, että niistä voi oppia jotain ja saadaan sitten sprintti pysymään seuraavalla kerralla paremmin kasassa ja tehdään asioita ehkä paremmin.” (D2)

> ”Silloin kun tämä osataan vetää oikein, jokainen sprintti on edellistä parempi. Tämä auttaa poistamaan työssä olevia lukkoja ja auttaa ihmisiä tekemään oman työnsä paremmin, ymmärtämään toisiaan ja tuntemaan toisiaan paremmin, jne.” (A2)

Toisena hyvänä puolena mainittiin se, että käytänne tarjoaa tiimin jäsenille tilaisuuden kertoa, ”mikä hiertää ja mikä taas toimii” (C1) ja millaisia konkreettisia kehitysehdotuksia heillä on (A1). Tällaista tilannetta ei perinteisemmissä projekteissa tule vastaan välttämättä missään vaiheessa projektin elinkaarta. Suora palaute antaa myös projektipäällikölle paremman valmiuden puuttua tiimiä haittaaviin asioihin (A1). Jos tapaamisessa on mukana myös asiakkaan edustaja, niin kuin haastateltavan A1 projekteissa on, käytänne tuo entistä enemmän avoimuutta ja läpinäkyvyyttä niin asiakkaan suuntaan, kuin tiiminkin sisäisesti (A1).

> ”Tämä on paras ja olennaisin osa koko Scrumissa. Ei tosin itse tapaaminen, vaan siinä saatava tieto ja sen hyödyntäminen. Projektipäällikölle / Scrum-mestarille tämä on ehkä haasteellisin alue, koska arviointitapaamisessa on vaikea saada tiimin jäsenistä ulos tuntemuksia ja miten projektissa mukana olevat voisivat kehittyä tiiminä.” (B1)

Haastateltava B1 oli siis hyvin samoilla linjoilla haastateltava A2:n kanssa, jonka kokemukset sprintin arviointitapaamisen haasteista projektipäällikön näkökulmasta katsottuna käytiin jo aikaisemmin läpi. Yrityksessä B sprintin arviointitapaamisessa oli kokeiltu rohkeasti eri lähestymistapoja:

> ”Joskus sprintin arviointitapaamisiin otetaan jokin tietty aihe tai aihealue, josta sitten arviointitapaamisessa keskustellaan. Aihe voi tulla Scrum-mestarilta tai joskus myös tiimi saa päättää, mistä he haluavat keskustella.

> Joskus tapaamisissa käytetään jotain Scrumin periaatteisiin tai toimintatapoihin liittyviä pelejä tai muuta toiminnallista tekemistä, koska niiden avulla ihmiset saadaan normaalia paremmin mukaan. Aina semmoinen retro (sprintin arviointitapaaminen), jossa on joku toiminta, on parempi kuin semmoinen missä ei ole. Sprintin arviointitapaamisessa esiin noussut yksi toteutettu kehitysehdotus on ollut esimerkiksi se, että 'tikettijärjestelmästä' luovuttiin ja siirryttiin merkitsemään tehtävät post-it lapuille." (B1)

Käytänteen huonoja puolia kysyttäessä viitattiin täälläkin siihen, että kokematon tiimi ja ennen kaikkea kokematon Scrum-mestari voi tehdä sprintin arviointitapaamisesta todella helposti hyödyttömän tapaamisen (A2, B1, C1). Huonoimmillaan sprintin arviointitapaamisesta tulee tiimin jäsenille todella ahdistava tilaisuus (A2), joissa arviointitapaamisiin tottumattomat eivät välttämättä uskalla kertoa todellisia mielipiteitään (C1). "Siitä saa paremman hyödyn, jos kaikki uskaltaa todella sanoa, mikä on ongelmallista. Tilanteen tulisi olla tarpeeksi vapaamuotoinen, jotta tähän päästäisiin." (C1) Haastateltava B1 mainitsee tiimin aktivoimisen haasteelliseksi erityisesti sellaisissa tiimeissä, joissa on mukana jäseniä eri kulttuureista. Haastateltava mainitsee ongelmalliseksi myös toisen ääripään:

> "Jos Scrum-mestari ei ole tarkkana ja yritä saada tapaamisesta jotakin konkreettisia päätöksiä, voi tapaamisesta tulla vain sellainen tilaisuus, jossa käydään vaan valittamassa asioista tai siellä vain heitellään ilmoille joitakin ajatuksia, joille ei sitten kuitenkaan tehdä mitään." (B1)

Haastateltavat A1 ja D2 eivät ole kokeneet mitään sen suurempia ongelmia sprintin arviointitapaamisen kanssa, kuin että tapaamisissa ei aina tule kaikilla mieleen mitään järkevää sanottavaa kun samat asiat kysytään kuitenkin kaikilta tiimin jäseniltä (A1).

5.3 XP:n käytänteet

XP:n käytänteiden haasteena tässä tutkimuksessa oli se, että haastateltavilla oli huomattavasti vähemmän kokemusperäistä tietoa XP:hen liittyvistä käytänteistä Scrumin käytänteisiin verrattuna. XP:ssä on tiettyjä ominaispiirteitä, joita ei kaikissa ketterissä menetelmissä käytetä. XP hyödyntää esimerkiksi pariohjelmoinnin tuomia etuja, virheiden minimoimiseksi kaikki kirjoitettu koodi on syntynyt kahden ohjelmoijan yhteistyönä (Paulk 2001). Täten millään kirjoitetulla koodin osalla ei ole yksittäistä omistajaa, vaan kaikilla on oikeus parannella ja muokata kirjoitettua koodia (Paulk 2001). Rajoittamalla työviikon tiukasti 40 tuntiin, toisin sanoen kieltämällä ylityötunnit, XP ottaa huomioon myös työntekijöiden jaksamisen (Vanderburg 2005). Lisäksi testivetoisen kehityksen ja jatkuvan integraation uskotaan lisäävän koodin laatua (Beck 1999b, Leffingwell 2007a).

Tutkielman seuraavissa alaluvuissa käsitellään XP:n käytänteistä tarkemmin seuraavia: pariohjelmointi (engl. pair programming), koodin yhteinen omistajuus (engl. collective code ownership), testivetoinen kehitys (engl. test-driven development), jatkuva integraatio (engl. continuous integration) ja jatkuva asiakkaan läsnäolo (engl. on-site customer). Tarkastelun ulkopuolelle jäävät 40 tunnin työviikko (engl. 40-hour week), refaktorointi (engl. refactoring), suunnittelupeli (engl. plannig game), pienet julkistukset (engl. small/short releases), metaforat (engl. metaphor), yksinkertainen suunnittelu (engl. simple design), koodausstandardit (engl. coding standards), avoin työtila (engl. open workspace) ja vain säännöt (engl. just rules). XP:hen liittyvien käytänteiden määrittely perustuu ensisijaisesti julkaisuihin Abrahamsson, Salo ja Ronkainen (2002) sekä Beck (1999a, 1999b).

5.3.1 Pariohjelmointi

Pariohjelmointi (engl. pair programming) on nimensä mukaisesti pareittain ohjelmoimista. Työparit istuvat saman koneen äärellä ja toinen parista, ajaja (engl. driver), kirjoittaa koodia sillä aikaa kun toinen, navigaattori (engl. navigator), tarkkailee ja pohtii, onko tekeillä oleva ratkaisu paras mahdollinen, läpäiseekö se tarvittavat testit ja onko ratkaisua vailla oleva ongelma mahdollista poistaa kokonaan yksinkertaistamalla koodia (Arisholm, Gallis, Dybå & Sjøberg 2007, Beck 1999b). Tällöin ajajana toimiva ohjelmoija saa välitöntä palautetta työstään ja arvokkaan toisen mielipiteen työstettävän ongelman kannattavimmasta ratkaisutavasta.

Aikaisemmat empiiriset tutkimukset

Pariohjelmointi on eniten tutkittu tässä tutkielmassa käsiteltävä käytänne. Koska pariohjelmoinnista on tehty runsaasti empiirisiä tutkimuksia, löytyy kirjallisuudesta myös aikaisempia tutkimustuloksia kokoavia teoksia, esimerkiksi Hulkon ja Abrahamssonin (2005) artikkeli. Vaikka tutkimusta aiheesta on tehty paljon ja niistä saadut tulokset ovat usein pariohjelmointia puoltavia, käytännettä on siihen nähden otettu käyttöön oikeissa projekteissa todella harvoin. Begelin ja Nagappanin (2007) tekemän laajan (488 hyväksyttyä vastauslomaketta) kansainvälisen kyselytutkimuksen mukaan vain noin 5 % kyselyyn vastanneista hyödynsi pariohjelmointia projekteissaan vakituisesti. Samassa tutkimuksessa pariohjelmointi kuitenkin koettiin mielenkiintoiseksi vaihtoehdoksi, koska noin 30 % vastanneista hyödynsi pariohjelmointia joskus ja noin 4 % aikoi testata sitä jossakin vaiheessa (Begel & Nagappan 2007). Hyvin monessa pariohjelmointia koskevassa empiirisessä tutkimuksessa tutkimuksen kohdeprojektit olivat yliopistoprojekteja, joissa mukana olevilla koehenkilöillä ei ollut varsinaista kokemusta ohjelmistoalan töistä (Hulkko & Abrahamsson 2005). Koska opiskelijaprojekteista saadut tulokset eivät ole välttämättä suoraan verrattavissa oikeisiin ohjelmistoalan projekteihin, kaiva-

taan pariohjelmoinnistakin empiiristä tutkimusta oikeassa ympäristössä (Dybå & Dingsøyr 2008). Seuraavaksi käsiteltävien empiiristen tutkimustulosten osalta ilmaistaan erikseen, onko johtopäätökset tehty opiskelijaprojektien, vai oikeiden projektien perusteella.

Pariohjelmoinnin positiiviset vaikutukset aiempien empiiristen tutkimusten mukaan

Pariohjelmoinnin uskotaan nostavan eritoten kehitettävän ohjelmakoodin laatua (Leffingwell 2007a), koska ohjelmakoodia tavallaan katselmoidaan ensimmäisen kerran jo kirjoitusvaiheessa, jolloin virheet voidaan huomata mahdollisimman aikaisin (Hulkko & Abrahamsson 2005). Empiirisissä tutkimuksissa on saatu ristiriitaisia tuloksia koodin laadun paranemisesta tai koodin virheiden vähenemisestä: Cockburn ja Williams (2001), Fitzgerald, Hartnett ja Conboy (2006) sekä Tessem (2003) havaitsivat tutkimuksissaan koodin laadun paranemista, kun taas Arisholm, Gallis, Dybå ja Sjøberg (2007) sekä Auvinen, Back, Heidenberg ym. (2005) eivät todenneet tutkimuksissaan merkittävää laadun nousua.

Virheiden vähenemistä onkin vaikea todeta juuri pariohjelmoinnin ansioksi, koska usein pariohjelmointia hyödynnettäessä käytetään projektissa myös testivetoista kehitystä, jolloin mahdollisiin virheitä aiheuttaviin kohtiin osataan kiinnittää jo ohjelmointivaiheessa enemmän huomiota (Auvinen ym. 2005). Arisholm ym. (2007) totesivat tutkimuksessaan myös, että pariohjelmoinnin vaikutus virheiden määrään riippuu kehitettävän ohjelmiston kompleksisuudesta ja mahdollisesti myös parin asiantuntijuuden tasosta. Kyseisessä tutkimuksessa tehdyn kontrolloidun kokeen mukaan kompleksisen ohjelmiston kehittämisessä pariohjelmoinnista on selkeästi eniten hyötyä, jos parin asiantuntijuuden taso on molemmilla aloittelija (engl. junior) (Arisholm ym. 2007). Pareittain ohjelmoivat aloittelijat saivat tutkimuksessa 73 % enemmän

oikeita vastauksia kuin vastaavalla tasolla olevat yksittäin ohjelmoijat (Arisholm ym. 2007). Parin tietotason noustessa pariohjelmoinnin hyöty samantasoisiin yksittäin ohjelmoiviin kehittäjiin verrattuna laski: keskitason parilla pariohjelmoinnista oli vain 4 % hyöty ja senioritasolla pariohjelmointi vaikutti negatiivisesti tulokseen, eli yksittäin ohjelmoivat seniorit saivat 8 % enemmän oikeita vastauksia kuin pareittain ohjelmoivat seniorit (Arisholm ym. 2007). Näiden tutkimustulosten mukaan ammattilaisten kohdalla ei voida sanoa, että olisi järkevää ns. ”lyödä kaksi viisasta päätä yhteen”. Parien muodostamisen tärkeyteen on kiinnittänyt huomiota myös Auvinen ym. (2005) ja he esittävätkin, että jokaiselle tiimin jäsenelle määriteltäisiin oma pätevyystaso (engl. competence level) ja samoin myös kaikista tehtävistä arvioitaisiin niiden vaikeustaso. Tämän avulla kuhunkin tehtävään osattaisiin valita siihen tehtävään sopiva pariyhdistelmä, esimerkiksi keskitasoiseen tehtävään voitaisiin valita opetusmielessä korkean ja matalan pätevyystason tiimin jäsenet pariksi (Auvinen ym. 2005).

Laadun paraneminen näkyy myös laajemmalla alalla kuin yksittäisten virheiden määrän vähenemisellä. Eräässä opiskelijaprojektissa havaittiin, että pareittain ohjelmoitu toiminnallisuus sisälsi keskimäärin vähemmän rivejä, jonka tutkijat olettivat johtuvan paremmasta suunnittelusta (Cockburn & Williams 2001). Parempaa suunnittelua edistää empiiristen havaintojen mukaan myös se, että useampi ohjelmoija on pariohjelmoinnin myötä paremmin perillä kehitteillä olevasta kokonaisuudesta, eli tiedonsiirto sujui pariohjelmoinnin ansiosta jouhevammin. Myös oppiminen oli tehokkaampaa pariohjelmoinnin ansiosta. Näihin päätelmiin oli tullut useampi tutkimus niin opiskelijaprojektien pohjalta (Back, Milovanov, Porres & Preoteasa 2002, Canfora, Cimitile & Visaggio 2004, Cockburn & Williams 2001) kuin aidossa kehitysympäristössä tehdyissä tutkimuksissa (Auvinen ym. 2005, Chong 2005, Hulkko & Abrahamsson 2005, Rumpe & Schröder 2002, Tessem 2003) sekä Dybå, Arisholm, Sjøberg ym. (2007)

aikaisempia tutkimustuloksia kokoavassa teoksessaan. Esimerkiksi testaajien työtä helpotti Auvisen ym. (2005) tekemässä tutkimuksessa se, että testaajat pystyivät kommunikoimaan ja kyselemään lisätietoja useammalta ohjelmoijalta tietystä ohjelmakoodin osasta. Laajemmassa, ympäri maailmaa lähetetyssä kyselytutkimuksessa 17,8 % vastaajista mainitsi pariohjelmoinnin suurimmaksi syyksi ketterän projektin menestykseen testivetoisen kehityksen ohella: ”Koodaaminen tapahtui paljon nopeammin ja tiedonsiirto ohjelmoijien välillä oli tiivistä” (Rumpe & Schröder 2002, 99).

Kirjoitettavan koodin laadun paranemisen lisäksi pariohjelmoinnilla uskotaan olevan myös positiivinen vaikutus työilmapiiriin, koska pareittain työskentely on monen tutkittavan mielestä ollut mielekkäämpää kuin yksin ohjelmoiminen (Cockburn & Williams 2001, Hulkko & Abrahamsson 2005, Mannaro, Melis & Marchesi 2004, Nosek 1998). Nopeampi virheiden huomaaminen ja täten laadukkaamman koodin kirjoittaminen nostaa myös monen tutkimuksen mukaan ohjelmoijien itsetuntoa (Hulkko & Abrahamsson 2005).

Pariohjelmoinnin negatiiviset vaikutukset aiempien empiiristen tutkimusten mukaan

Pariohjelmoinnin negatiivisina puolina on mainittu mm. että se vie resursseja enemmän kuin yksittäin ohjelmointi. Esimerkiksi Cockburnin ja Williamsin (2001) tutkiman opiskelijaprojektin perusteella todettiin, että pariohjelmointina tuotetun koodin tekemiseen meni 15 % enemmän aikaa kuin yksin ohjelmoijilla, mutta tuotettavan koodin laatu oli vastaavasti 15 % parempaa kuin verrokkiryhmällä pariohjelmoinnin ansiosta. Hulkko ja Abrahamsson (2005) huomauttavat kuitenkin neljän tapaustutkimuksensa keskinäisen hajonnan perusteella, että merkittävää kasvua tai laskua ajallisessa hyödyssä ei voida todeta puoleen tai toiseen, tätä tukee myös Nosekin (1998) aikaisemmin tekemä tutkimus.

Hulkko ja Abrahamsson (2005) nostavat esille myös sen negatiivisen puolen, että pariohjelmointi työskentelymuotona ei vain yksinkertaisesti sovi kaikille, jolloin se väkisinkin tuo esille huonot henkilökemiat tai muut mahdolliset konfliktit (Hulkko & Abrahamsson 2005). Pariohjelmoinnissa sosiaaliset taidot ovatkin koetuksella. Esimerkiksi kritiikin antaminen ja sen vastaanottaminen voi olla joillekin ohjelmoijille hyvin vaikeaa, mutta ilman näitä taitoja on tiimillä edessään vakavia kommunikaatio-ongelmia (Tessem 2003). Pareittain ohjelmointi voi olla myös joillekin ohjelmoijille hyvin raskasta, koska se vaatii normaalia enemmän keskittymistä (Ilieva, Ivanov & Stefanova 2004, Tessem 2003).

Pariohjelmoinnissa kannattaakin pitää mielessä, millaisiin tilanteisiin se parhaiten sopii. Esimerkiksi Grossmanin, Bergin, Leipin ym. (2004) tekemässä empiirisessä tutkimuksessa huomattiin, ettei pariohjelmointi ollut luonnollista web-pohjaisen ohjelmiston kehittämisprojektissa, koska jokaisella ohjelmoijalla oli siinä selkeä oma osaamisalueensa. Tällaisessa tilanteessa pariohjelmointi ei ole perusteltua, ellei sitä käytetä esimerkiksi opetus– tai perehdytysmielessä. Yksinkertaiset tehtävät koettiin myös sellaisiksi, joiden tekemiseen pariohjelmointia ei empiiristen tutkimustulosten mukaan kannata käyttää (Fitzgerald, Hartnett & Conboy 2006, Hulkko & Abrahamsson 2005, Sutherland, Viktorov, Blount & Puntikov 2007). Yksinkertaisiin ongelmiin onkin turha tuhlata kahden ohjelmoijan aikaa, jos taitava ohjelmoija pystyy ratkomaan ongelman jo koodia kirjoittaessaan (Fitzgerald ym. 2006).

Ketterän projektityöskentelyn luonne aiheuttaa osaltaan ongelmia pariohjelmoinnin tehokkaalle hyödyntämiselle. Jos pariohjelmointia toteutetaan esimerkiksi avokonttorissa, on keskeytyksiin, taukoihin ja aikataulutukseen liittyviltä ongelmilta vaikea välttyä. Chongin (2005) etnografisesta tutkimuksesta ilmenee, että tarkkailun alla olleet parit eivät saaneet työskennellä rau-

hassa yli kahden - kolmen tunnin aikajaksoja. Työskentely keskeytyi esimerkiksi muiden tiimin jäsenten kysyessä jotakin parin toiselta jäseneltä (Chong 2005). Fitzgeraldin ym. (2006) tekemissä haastatteluissa nousi esille tarve yhteisten pelisääntöjen kehittämiselle ja noudattamiselle. Haastatteluun osallistuneissa yrityksissä kaavailtiinkin pariohjelmoinnin pelisäännöiksi mm., että pariohjelmointia tehdessä otetaan myös muut samassa tilassa olijat huomioon pitämällä keskustelun taso hiljaisena, pariohjelmointisessioille asetetaan sen alussa selkeät tavoitteet ja että sessiot priorisoidaan korkealle, jotta yhteistä aikaa saadaan järjestettyä (Fitzgerald ym. 2006, 206).

Haastattelututkimuksen tulokset

"[Pariohjelmointi tarkoittaa sitä, että] kaikki tuotettu koodi on kirjoitettu kahden ohjelmoijan istuessa yhdellä koneella" (Beck 1999b, 48)

	A1	A2	B1	C1	D1	D2	KA
Pariohjelmoinnin hyödyllisyysaste	1	3	2	0	3	2	1,8

Pariohjelmointi oli hyödyllisyysasteikolla mitattuna vähiten arvostettu käytänne haastateltavien keskuudessa, mutta silti yksikään haastateltavista ei pitänyt sen hyödyntämistä haitallisena projektille. Haastattelussa kaksi vastaajista totesi, että heillä oli kokemusperäistä tietoa pariohjelmoinnista vain hyvin rajoitetusti (A1, C1).

Pariohjelmoinnin määritelmä on näin tiivistetyssä muodossa aika rajoittava, joten ei ollut yllätys, ettei yksikään haastateltavista ollut toteuttanut käytännettä täysin tämän määritelmän mukaisesti. Suurin ero annettuun määritelmään oli siinä, että pariohjelmointia käytettiin vain tapauskohtaisesti tarpeen niin vaatiessa ja tällaisessa käytössä pariohjelmointi koettiin hyödylliseksi käytänteeksi yritykselle. Esimerkiksi haastateltava D1 mainitsi, että pariohjelmointi saisi hyödyllisyysasteikolla arvosanan 1, jos sitä toteutettaisiin täsmälleen määritel-

män mukaisesti, mutta vapaammin toteutettuna vain tarpeen mukaan pariohjelmointi on yritykselle todella hyödyllinen käytänne.

Haastatteluissa tuli selvästi ilmi, että pariohjelmointia hyödynnettiin yleensä vain sellaisissa tilanteissa, joissa oli jokin kriittinen kohde, joka vaati laajaa ymmärrystä ja kaksia aivoja. Tämä tulos on samansuuntainen aiemmin kirjallisuudessa havaittujen empiiristen tulosten kanssa (Fitzgerald, Hartnett & Conboy 2006, Hulkko & Abrahamsson 2005, Sutherland, Viktorov, Blount & Puntikov 2007). Pariohjelmoiden ratkaistavat ongelmat saattoivat koskea esimerkiksi tekniseen toteutukseen liittyviä haasteita (D1), haastavien virheiden löytämistä (B1), laajojen kokonaisuuksien muutostöitä (B1) tai uusien työntekijöiden perehdyttämistä (D2). Yhdessä haastattelussa pariohjelmointia käytettiin osittain myös projektiryhmän keskustelun välineenä ja avaajana, eli jos jokin sellainen tärkeä asia tulee esille pariohjelmoinnin aikana, jota kannattaisi käydä läpi koko porukan kanssa, kutsuttiin paikalle useampi ihminen miettimään kyseistä asiaa (D1). Yrityksessä C pariohjelmointia oli muokattu niin paljon, että on kyseenalaista, voidaanko tässä enää puhua pariohjelmoinnista: kaikki kirjoitettu koodi menee erityiselle ”katselmointihyllylle”, josta joku toinen käy lukemassa ja kommentoimassa sen ja palauttaa sen jälkeen kommentoidun koodin kirjoittajalle (C1). Kaiken kirjoitetun koodin on tarkastellut vähintään kaksi silmäparia, se vain ei tapahdu samassa tilassa tai samanaikaisesti.

Pariohjelmointia hyödynnettiin haastatelluissa yrityksissä aika monenkirjavasti, käyttöaste vaihteli lähes päivittäisestä käytöstä muutamaan kertaan koko projektin aikana, silloinkin vain tiukoissa tilanteissa. Pääosin pariohjelmointiin turvauduttiin kerran tai kaksi kertaa Scrumiin liittyvän sprintin aikana, eli toisin sanoen keskimäärin kerran viikossa. Yhdessä haastattelussa mainittiin

avotoimiston edesauttavan pariohjelmoinnin luontevaa hyödyntämistä päivittäisessä työssä (D1).

Parien muodostamiseen ei ollut missään yrityksessä erityisiä vaatimuksia (vrt. Arisholm, Gallis, Dybå & Sjøberg 2007, Auvinen, Back, Heidenberg ym. 2005), mikä johtunee pariohjelmoinnin käyttöasteesta projekteissa. Kun pariohjelmointia hyödynnetään vain sitä vaativien ongelmien tullessa esiin, pariksi pyydetään haastattelujen mukaan poikkeuksetta vain muista kehittäjistä sellainen henkilö, joka osaa kyseisessä ongelmassa auttaa. Luultavasti tilanne olisi kuitenkin toinen, jos pariohjelmointia ryhdyttäisiin tekemään näissä yrityksissä pysyvänä työmuotona, jolloin parien valinnalla olisi suurempi merkitys työn etenemiseen. Henkilökemioiden on kuitenkin toimittava parin välillä lyhyissäkin pariohjelmointihetkissä: ”Todella tehokas tapa toimiessaan. Tämä (pariohjelmointi) on hyvä, kun siihen löytyy oikeanlaiset ihmiset ja jokin tiukka juttu, mikä pitää vääntää.” (A2)

Kysyttäessä pariohjelmoinnin positiivisia vaikutuksia ja hyviä puolia, vastauksissa nostettiin esille mm. kehitettävän koodin laadun parantuminen, perusteenaan se, että kaksi silmäparia huomaa paremmin virheitä kuin yksi (A1, B1, C1, D1). Tämä on samoilla linjoilla aiemmin käsiteltyjen tutkimustulosten kanssa (esim. Hulkko & Abrahamsson 2005). Aikaisempien tutkimusten mukaisesti pariohjelmoinnin koettiin auttavan myös tiedon ja uusien ideoiden jakamisessa tiimin kesken (B1, C1, D1), sekä uuden työntekijän perehdyttämisessä (D2, C1). Pariohjelmointi pakottaa arvioimaan sitä, mitä ollaan tekemässä (D1), jolloin tehdyt päätökset ja ratkaisut ovat paremmin perusteltuja kuin yksin ohjelmoidessa.

> ”Uusi työntekijä oppii pariohjelmoinnin avulla uusia käytänteitä ja siinä tulee paremmin kyseltyä enemmän asioista sekä työntekijä oppii parhaiten, mitä kaikkia asioita pitäisi tehdä ja millä tavalla.” (D2)

Kuten jo tämän alaluvun alussa todettiin, yksikään haastatelluista yrityksistä ei käyttänyt pariohjelmointia käytänteen määritelmän mukaisesti jatkuvasti. Kaikkien tehtävien ratkaiseminen pariohjelmoiden oli haastateltavien mielestä suorastaan järjetöntä (D2), koska se vaatii luonnollisesti enemmän tiimin aikaa kuin yksin ohjelmointi (B1, D1). Tämä havainto on myös aiempien tutkimusten kanssa samoilla linjoilla, vaikka merkittäviä tutkimustuloksia suuntaan tai toiseen ei ole vielä saatavilla.

Hulkon ja Abrahamssonin (2005) havaintojen mukaisesti myös ihmisten henkilökemioiden kohtaaminen pariohjelmoinnissa koettiin joissakin tapauksissa ongelmalliseksi. Pariohjelmointi ei vain oikein toimi hyvin kaikkien ihmisten kanssa (A1, A2, B1) tai missä tilanteessa tahansa (A2, B1, D1). Ohjelmointitehtävään keskittyminen voi olla hyvin vaikeaa, esimerkiksi jos pari on kovin puhelias, puhumattakaan siitä, jos kaikilla koneilla avotoimistossa hyödynnettäisiin pariohjelmointia ja keskusteltaisiin (D1). Kyseiseen ongelmaan voisi kuitenkin auttaa aiemmin mainitut Fitzgeraldin, Hartnettin ja Conboyn (2006) hahmottelemat yhteiset pariohjelmoinnin pelisäännöt.

5.3.2 Koodin yhteinen omistajuus

Koodin yhteinen omistajuus (engl. collective code ownership) on hyvin luonnollinen lisä pariohjelmoinnille. ”Kaikki, jotka näkevät jossakin koodin osassa mahdollisuuden lisätä koodin arvoa, ovat velvoitettuja myös tekemään niin, missä vaiheessa tahansa” (Beck 1999b). Yhteinen koodin omistajuus tarkoittaa myös sitä, että kaikki kehittäjät ovat vastuussa kehitteillä olevasta ohjelmistosta kokonaisuudessaan. Jotta tämä käytänne voisi oikeasti toimia käytännössä, on jokaisen kehittäjän sitouduttava yhteisiin koodauskäytänteisiin, joka on myös yksi XP:n käytänteistä. (Beck 1999b)

Aikaisemmat empiiriset tutkimukset

Rumpen ja Schröderin (2002) tekemän kyselytutkimuksen mukaan koodin yhteinen omistajuus oli kaikista laajimmin käytössä XP:n eri käytänteistä, koska se oli helpoin asia toteuttaa XP-projektissa. Begelin ja Nagappanin (2007) vielä laajemmalti toteutetussa tutkimuksessa saatiin samankaltaisia tuloksia: noin 45 % vastaajista käytti koodin yhteistä omistajuutta aina projekteissaan, jonka lisäksi noin 20 % hyödynsi käytännettä joskus. Salon ja Abrahamssonin (2008) tekemään kyselytutkimukseen osallistui 35 projektia 18:sta eri sulautettuihin järjestelmiin erikoistuneesta yrityksestä ympäri Euroopan. Kyseisessä tutkimuksessa kyselyn tulokset olivat hyvin selkeät koodin yhteisen omistajuuden kannalta: kyselyssä tuli käytänteestä ainoastaan positiivista palautetta ja käytänteen käyttöaste oli samaa luokkaa edellä mainitun Begelin ja Nagappanin (2007) tutkimuksen kanssa. Käytänteen positiivisiksi puoliksi luokiteltiin koodin laadun paraneminen (Back, Milovanov, Porres & Preoteasa 2002) ja että useammalla tiimin jäsenellä on parempi käsitys kehitettävästä koodista, jolloin esimerkiksi tiimin jäsenten vaihtuvuus ei ole yhtä suuri ongelma kuin aiemmin (Back ym. 2002, Fitzgerald, Hartnett & Conboy 2006, Gittins, Hope & Williams 2002).

Koodin yhteisen omistajuuden haasteena nähtiin taas isojen tiimikokojen yhteensovittaminen yhteisen koodin omistajuuden kanssa (Fitzgerald, Hartnett & Conboy 2006) ja yhteisomistajuus sellaisissa tiimeissä, joissa osaamisalueet ovat hyvin erilaiset (Grossman, Bergin, Leip ym. 2004). Gittinsin ja Hopen (2001) tekemässä tutkimuksessa yksi yhdeksästä tutkimuksen osallistuneesta huomautti, että käytänteen käyttöönoton alkuvaiheessa hänestä tuntui siltä, että koodin laatu vain huononi, kun joku muu kävi tekemässä siihen muutoksia. Tutkimuksessa ei kuitenkaan kerrottu, tapahtuiko tilanteessa muutoksia käytänteen vakiintumisen myötä (Gittins & Hope 2001). Koodin yhteinen omistajuus aiheutti myös Tinglingin ja Saeedin (2007) havaintojen mukaan

aluksi ongelmia tutkittavien keskuudessa: kehittäjät pitivät tiukasti kiinni ”omasta koodistaan”. Käytännettä alettiin kuitenkin ajan myötä arvostaa kehittäjienkin keskuudessa enemmän (Tingling & Saeed 2007).

Haastattelututkimuksen tulokset

”[Koodin yhteinen omistajuus on sitä, että] kuka vain voi muuttaa mitä tahansa koodin osaa järjestelmässä milloin vain.” (Beck 1999b, 48)

Koodin yhteisen omistajuuden hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	-	-	3	3	3	3	3

Koodin yhteinen omistajuus oli käytössä kolmessa haastatellussa yrityksessä B, C ja D, joissa kaikki haastateltavat pitivät käytännettä erittäin hyödyllisenä. Yrityksissä käytännettä myös hyödynnettiin lähes poikkeuksetta määritelmän mukaisesti. Yrityksessä C erona määritelmään oli se, että muutoksia ei voi tehdä ihan milloin vain ja minne vain, vaan jo pariohjelmoinnin yhteydessä mainittu ”katselmointihylly” takaa, että muutosehdotuksen on nähnyt myös joku muu tiimin jäsenistä. Samankaltainen tehtyjen muutosten järkevyyden varmistamiskeino on myös yrityksessä D. Siellä jokainen tiimin jäsen voi merkitä yhteiseen koodiin epäilyttävät ratkaisut, joiden vakavuutta mietitään sen jälkeen yhdessä. Jos havaittu asia on riittävän paha, se voidaan merkitä koodiin niin, ettei se mene järjestelmän integraatiotestauksesta läpi ennen kuin kyseinen kohta on korjattu (D1). Haastateltavien projekteissa kaikentyyppiset koodit olivat yhteisomistajuudessa lukuun ottamatta sellaisia koodin osia, nk. ”peruspalikoita”, joita hyödynnetään useissa eri projekteissa (D2).

Koodin yhteisen omistajuuden hyviksi puoliksi haastateltavat listasivat mm. koodin luettavuuden paranemisen (B1, C1, D1, D2), laadun paranemisen (B1, D1), kommentoinnin ja selkeyden lisääntymisen (C1, D1), tekemisen läpinäkyvyyden lisääntymisen (D1) sekä järjestelmän integroimisen helpottumisen (B1). Kun koodi on tiimin yhteisessä omistajuudessa, yksi

selkeimmistä eduista on se, että kokonaisuuden hallinta on helpompaa. Tiimin jäsenet ovat tällöin paremmin tietoisia kehitettävän systeemin suuremmista linjoista, esimerkiksi arkkitehtuuriratkaisuista ja luokkarakenteista (D1). Helpotusta koodin yhteisomistajuus tuo myös muuttuviin tiimeihin, kun kukaan ei lähtiessään vie kaikkea omaa koodia koskevaa tietoa mennessään, vaan useampi ihminen on osallistunut sen kehittämiseen (B1). Myös tieto siitä, että muutkin tarkastelevat ja mahdollisesti myös korjailevat kirjoittamaasi koodia, auttaa automaattisesti noudattamaan mahdollisia koodausstandardeja, kirjoittamaan koodiin enemmän selventäviä kommentteja sekä kyseenalaistamaan omia ja muiden tekemiä ratkaisuja entistä paremmin (B1). Toiselta kantilta katsottuna rehellisen palautteen antaminen helpottuu yhteisomistajuuden myötä, koska kaikki ovat vastuussa koodista kokonaisuutena (B1), jolloin jotakin tiettyä koodin osaa koskevaa kritiikkiä ei tarvitse kenenkään ottaa itseensä (C1).

Huonoja puolia kysyttäessä ei niitä juurikaan mainittu. ”Mitään huonoja puolia ei tule mieleen, koska tämä on ainoa järkevä tapa, jolla tämä asia kannattaa hoitaa.” (B1) Tämä tulos on samanlainen, mitä (Salo & Abrahamsson 2008) ovat saaneet myös omassa laajemmassa kyselytutkimuksessaan, jossa käytännettä hyödyntävillä yrityksillä oli vain ja ainoastaan positiivista sanottavaa koodin yhteisestä omistajuudesta (Salo & Abrahamsson 2008). Ainoana huonona puolena yksi haastateltavista näki sellaisen tilanteen, että koodiin tehdään muutoksia tietämättä, mihin kaikkeen kyseiset muutokset voivat loppujen lopuksi vaikuttaa (D2). Tämä mahdollisuus on kuitenkin kompensoitu automaattisilla testeillä, joten tämäkään ei ole vakava puute haastateltavan mielestä koodin yhteisessä omistajuudessa (D2). Haastatteluissa saadut tulokset ovat samoilla linjoilla aikaisempien empiiristen tutkimusten kanssa.

5.3.3 Testivetoinen kehitys

Testivetoisella kehityksellä (engl. test-driven development, lyh. TDD) tarkoitetaan XP:ssä sitä, että testit pyritään kirjoittamaan ennen varsinaisen ohjelmakoodin kirjoittamista. Nämä testit voidaan jakaa kahteen luokkaan: yksikkötestaukseen (engl. unit testing), jossa testien kirjoittamisesta vastaavat ohjelmoijat, sekä asiakkaan kirjoittamiin toiminnallisiin testeihin (engl. functional tests). (Beck 1999a)

Yksikkötestauksella tarkoitetaan ohjelmoijien kirjoittamia testejä, jotka testaavat nimensä mukaisesti jonkin yksittäisen toiminnallisuuden toimivuutta. Yksikkötestit toimivat ohjelmoijien tukena ja varmistuksena siitä, että kirjoitettu koodi on toimivaa. Beck (1999b) vertaa yksikkötestien kirjoittamista vedonlyöntiin, joka kannattaa, jos jostakin syystä ohjelmoijan olettamukset osoittautuvat vääriksi, koskivatpa olettamukset testien toimivuutta silloin kun niiden ei oletettu toimivan tai sitä, kun jokin testi ei mene läpi silloin kun sen oletetaan menevän. Kirjoitetun koodin tulee läpäistä sataprosenttisesti kaikki automaattiset yksikkötestit, koska niitä kirjoitetaan vain sellaisissa tapauksissa, joissa ohjelmoija on epävarma toteutuksen monimutkaisuudesta, rajapinnoista, olosuhteista tai muusta vastaavasta. (Beck 1999b)

Asiakkaan kirjoittamat toiminnalliset testit ovat yksikkötesteihin verrattuna hieman vähemmän kriittisiä ohjelmiston toiminnan kannalta. Testejä kirjoitetaan testaajan avulla tarina kerrallaan, ja niiden avulla pyritään varmistamaan, että tuotteessa on kaikki vaaditut toiminnallisuudet ja että ne toimivat niin kuin niiden pitääkin. (Beck 1999b)

Testivetoisen kehityksen hyötyjä perustellaan myös sillä, että se pakottaa kehittäjät miettimään kehitettävän koodin tarkoitusta ja toiminnallisuutta ennen itse ohjelmakoodin kirjoittamista. Testivetoisen kehityksen uskotaankin

parantavan kirjoitettavan koodin laatua ja se varmistaa myös sen, että kaikki kirjoitettava koodi tulee varmasti testattua. (Leffingwell 2007a)

Aikaisemmat empiiriset tutkimukset

Testivetoinen kehitys on pariohjelmoinnin ohella yksi käsitellyimmistä yksittäisistä käytänteistä alan kirjallisuudessa, joten siitä löytyy runsaasti empiirisiä tutkimustuloksia sekä tuloksia kokoavia teoksia. Usein käytännettä ei kuitenkaan toteuteta aivan edellä kirjoitetun määritelmän mukaisesti. Useimmiten ero on siinä, että asiakkaan kirjoittamista testeistä ei edes puhuta testivetoisen kehityksen yhteydessä. Tämä johtunee siitä, että TDD:tä käsitteenä ei aina mielletä yhdeksi XP:n käytänteeksi, vaan erilliseksi testaustavaksi, jolloin asiakkaan rooli testausprosessissa ei ole niin merkityksellinen.

Testivetoinen kehitys jakaa pariohjelmoinnin tavoin voimakkaasti mielipiteitä aiemmin tehdyissä empiirisissä tutkimuksissa. Tämän huomasivat myös Siniaalto ja Abrahamsson (2007) tekemässään tutkimuksessa, jossa kartoitettiin aikaisempia tutkimustuloksia testivetoisesta kehityksestä sekä toteutettiin kolme omaa tapaustutkimusta eri ohjelmistoprojekteissa. Käsitys testivetoisen kehityksen hyödyllisyydestä tuntui riippuvan paljolti siitä, oliko käytännettä käytetty tutkitussa yrityksessä tai tiimissä jo kauan, vai oliko se otettu käyttöön vasta siitä tehtävän tutkimuksen myötä. Uutena käytänteenä testivetoinen kehitys vaatii aivan uudenlaista ajattelua ja enemmän totuttelemista kehittäjiltä, ja tämä voi aiheuttaa haluttomuutta opetella koko käytännettä (Grossman, Bergin, Leip ym. 2004, Melnik & Maurer 2004, Siniaalto & Abrahamsson 2007). Testivetoisuus myös vie enemmän aikaa normaaliin testaukseen verrattuna (Siniaalto & Abrahamsson 2007). Melnikin ja Maurerin (2004) tutkimassa opiskelijaprojektissa testivetoista kehitystä kuvailtiin takaperoiseksi käytänteeksi ja sen omaksuminen koettiin hyvin vaikeaksi, koska testivetoisessa kehityksessä on oikeastaan enemmän kyse suunnittelusta kuin testauksesta.

Oppilailla oli myös vaikeuksia hahmottaa testivetoisuuden hyödyllisyyttä lyhyellä tähtäimellä (Melnik & Maurer 2004). Jos testivetoista kehitystä otetaan käyttöön sellaisessa yrityksessä, jossa vastaavaa testaustapaa ei olla koskaan aikaisemmin hyödynnetty, on ensiarvoisen tärkeää, että kehittäjät todella sitoutuvat käytänteeseen (Gittins & Hope 2001). Testivetoisen kehityksen haastavuudesta kertoo myös se, että tehdyissä kyselytutkimuksissa testivetoisen kehityksen käyttöprosentti oli selvästi alhaisempi kuin muilla XP:n käytänteillä: Begelin ja Nagappanin (2007) laajan kyselytutkimuksen mukaan testivetoista kehitystä hyödynsi jatkuvasti vain 15 % vastaajista, vastaava luku Salon ja Abrahamssonin (2008) tutkimuksessa oli 18 %. Edelleen Salon ja Abrahamssonin (2008) tutkimuksessa listattiin testivetoinen kehitys yhdeksi vähiten arvostetuksi käytänteeksi; 12 % vastaajista oli käytänteestä vain huonoa sanottavaa.

On testivetoisesta kehityksestä empiirisissä tutkimuksissa myös hyvää sanottavaa. Menetelmän alkuperäisen idean mukaan sen tulisi nostaa koodin laatua (Leffingwell 2007a) ja tämän onkin todettu käytänteen hyväksi puoleksi muutamassa tutkimuksessa (Begel & Nagappan 2007, Melnik & Maurer 2004, Siniaalto & Abrahamsson 2007, Tessem 2003). Vaikka Siniaallon ja Abrahamssonin (2007) kokoavassa tutkimuksessa testivetoisen kehityksen uskottiin vaikuttavan koodin laatuun positiivisesti, sen positiivinen vaikutus koodin suunnitteluun on kuitenkin kyseenalaista kirjoittajien omien tapaustutkimusten perusteella. Erdogmuksen, Morision ja Torchianon (2005) tutkiman opiskelijaprojektin perusteella tultiin siihen lopputulokseen, että vaikka testivetoisuus ei tuottanut parempilaatuista koodia, sen ansiosta laatu oli tasaisempaa kuin verrokkiryhmällä. Koodin yhtenäisyys ja suunnitelmallisuus eivät välttämättä ole niin selviä seurauksia testivetoisen kehityksen käytöstä kuin kirjallisuudessa annetaan ymmärtää (Siniaalto & Abrahamsson 2007), mutta esimerkiksi Begelin ja Nagappanin (2007) tutkimuksessa testivetoisuuden koettiin kuitenkin paranta-

van ohjelmiston suunnittelua, arkkitehtuuria ja suorituskykyä (Begel & Nagappan 2007, 6).

Koodin hahmottamiseen testivetoisesta kehityksestä tuntuu kuitenkin olevan hyötyä, sen koettiin antavan kehitettävälle koodille selkeän fokuksen (Erdogmus, Morisio & Torchiano 2005, Fitzgerald, Hartnett & Conboy 2006) ja ”se antoi ohjelmoijille paremman ymmärtämyksen siitä, mitä toiminnallisuuksia ohjelmalta vaaditaan asiakkaan näkökulmasta katsottuna.” (Fitzgerald ym. 2006, 206). Virheiden määrällinen väheneminen ja niiden nopeampi korjaaminen mainittiin useammassa Siniaallon ja Abrahamssonin (2007) käsittelemässä lähteessä sekä Begelin ja Nagappanin (2007) tutkimuksessa testivetoisuuden positiiviseksi puoleksi, jonka ansiosta testausprosessin koettiin myös nopeutuvan (Melnik & Maurer 2004). Muutoin testivetoisesta kehityksestä saadut tutkimustulokset ovat hyvin hajanaisia. Mainittuja hyötyjä oli mm. nopeampi oppiminen (Erdogmus, Morisio & Torchiano 2005), pienempi uudelleenmuokkaamisen vaiva (Erdogmus, Morisio & Torchiano 2005), testikattavuuden parantuminen (Siniaalto & Abrahamsson 2007), testien toimiminen osittain ohjelman dokumentaationa (Siniaalto & Abrahamsson 2007) sekä suorituskyvyn parempi ennustettavuus ja yksikkötestaamisen lisääntyminen (Siniaalto & Abrahamsson 2007).

Haastattelututkimuksen tulokset

”Testivetoisessa kehityksessä testaajat kirjoittavat koko ajan yksikkötestejä, jotka pitää läpäistä virheettömästi, jotta kehitystyö voi jatkua. Asiakkaat kirjoittavat testejä, joidenka läpäisseitä toiminnallisuuksia voidaan pitää valmiina.” (Beck 1999b, 47)

Testivetoisen kehityksen hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	2	3	1	2	3	3	2,3

Suurin muutos menetelmän määritelmään nähden oli sinä, että asiakas harvemmin kirjoitti mitään testejä. Asiakkaan kirjoittamien testien suhteen haastatel-

luissa yrityksissä oli siis hyvin samansuuntainen linjaus kuin suurimmalla osalla edellä käsitellyistä empiirisen tutkimuksen kohteista. Ainoastaan yksi haastateltavista ilmoitti, että myös asiakas osallistuu testausprosessiin kirjoittamalla testejä, jotka valmistuvan ohjelmiston tulee läpäistä (D2). Asiakkaan kanssa saatettiin kuitenkin miettiä, kuinka toteutettava tehtävä todennetaan tai kuinka toteutetut toiminnallisuudet asiakkaalle näytetään (C1). Osa vastaajista totesi, että testivetoisen kehityksen hyödyntäminen on kunkin kehittäjän oma valinta, kunhan testit tulevat tavalla tai toisella tehtyä (B1, D1). Yrityksessä B tyypillisin tilanne, jossa tukeudutaan testivetoiseen lähestymistapaan, on ohjelmassa olevien virheiden korjaaminen. Yrityksessä A testivetoisuutta hyödynnettiin arviolta yhdessä projektissa neljästä, riippuen projektin luonteesta ja tiimin halukkuudesta (A1, A2). Vain yrityksen D haastateltavat kertoivat käyttävänsä testivetoista lähestymistapaa projekteissaan lähes poikkeuksetta. Toinen heistä arvioi, että noin 50 % – 75 % koodista kirjoitetaan testit etukäteen (D1). ”Osa koodista on sellaista, jota ei nykyteknologialla edes pysty testaamaan. Kaikki liiketoimintalogiikkaa ja tietokantoja koskeva koodi testataan.” (D1)

Testivetoisen kehityksen hyvänä puolena haastateltavat näkivät sen, että testien kirjoittamisen jälkeen voi levollisemmin mielin koskea myös koodiin, koska etukäteen kirjoitetut testit pitävät huolen koodin toimivuudesta myös lisäysten jälkeen (B1, C1, D1). ”Ei ole mitään hyvää syytä siihen, miksei yksikkötestejä kirjoitettaisi. Jos niitä ei olisi, koodin ylläpito olisi ihan älytöntä. Silloin mihinkään ei uskaltaisi tehdä mitään muutoksia, ettei sovellus hajoaisi.” (D2) Koodin laadun todettiin parantuvan, jos testit on kirjoitettu fiksusti ja ne ovat tarpeeksi kattavia (A1). Kuten aikaisemmissakin empiirisissä tutkimuksissa, yksi koetuista positiivisista vaikutuksista liittyi koodin parempaan ymmärtämiseen ja asioiden tarkasteluun uudelta kantilta (A2, D1). Testivetoinen kehitys koettiin myös luontevaksi pariksi jatkuvan integraation kanssa (C1). Testivetoisesta

kehityksestä sanotaan olevan hyötyä siinä, että se auttaa hahmottamaan seuraavaksi kirjoitettavaa koodia ja että sen avulla löydetään virheet helpommin. Kun haastateltavia pyydettiin arvioimaan, kuinka heidän mielestään hyöty jakaantuu näiden kahden väittämän välillä, keskimäärin vastaukset olivat sen kannalla, että testivetoisuus auttaa virheiden löytämiseen enemmän (66 %) kuin koodin hahmottamiseen (33 %).

Käytänteen huonoksi puoleksi lähes kaikki haastateltavat mainitsivat sen vaativuuden ja siitä johtuvan hitauden (A1, A2, C1, D1, D2). Testivetoista kehitystä on myös vaikea perustella asiakkaalle, koska asiakkaan näkökulmasta katsottuna testien etukäteen suunnittelu ei ole tuottavaa työtä (A1). Testien laajuutta ja monimutkaisuutta voi olla myös vaikea arvioida, jotta testeistä tulisi tarpeeksi kattavat, mutta kuitenkin ylläpidettävät (D1). Kuten aikaisemmin kirjallisuudessakin on todettu, testivetoinen kehitys on osittain kehitettävän ohjelman etukäteen suunnittelua, joka ei ole välttämättä yhtä helppoa (A2) tai mielekästä (D2) kuin suunnittelu koodin edetessä.

5.3.4 Jatkuva integraatio

Jatkuva integraatio (engl. continuous integration) tarkoittaa sitä, että päivän aikana tuotettu koodi integroidaan ja testataan vähintään kerran päivässä, mieluummin useampaan kertaan pienemmissä erissä. Tällä pyritään takaamaan se, että ohjelmisto on vähintään jokaisen työpäivän lopussa varmasti toimiva ja täysin testattu. Koska integraation ja siihen liittyvän testauksen tekevät ne kehittäjät, jotka ovat integroitavan koodinpätkän kirjoittaneet, ei ole myöskään epäselvää, kenen vastuulla mahdollisten esiin tulevien virheiden korjaaminen on. (Beck 1999b)

Aikaisemmat empiiriset tutkimukset

Jatkuvasta integraatiosta löytyy hyvin vähän empiirisiä tutkimustuloksia, joissa olisi keskitytty nimenomaan tämän yksittäisen käytänteen vaikutukseen

ohjelmistotuotantoon. Tämä on yllättävää siinä mielessä, että esimerkiksi työkaluja tämän toteuttamiseen on tarjolla runsain määrin ja käytänne on laajalti hyödynnetty (van der Storm 2008). Begelin ja Nagappanin (2007) kyselytutkimukseen vastanneista lähes 60 % hyödynsi jatkuvaa integraatiota osana kehitystyötään. Tutkimuksessa mainittiin jatkuvan integraation auttavan erityisesti siinä, että julkaisut saatiin myös asiakkaan nähtäville nopeammassa aikataulussa, jolloin oli enemmän aikaa antaa palautetta kehitettävästä tuotteesta ja testata toiminnallisuuksia ja pohtia vielä tekemättömien toiminnallisuuksien todellista tarvetta (Begel & Nagappan 2007). ”Kun integroidaan aikaisessa vaiheessa ja usein, tuotetta voidaan myös testata aikaisessa vaiheessa ja usein.” (Begel & Nagappan 2007, 6). Ferreiran ja Cohenin (2008) tekemässä tapaustutkimuksessa todettiin niin ikään jatkuvan integraation vaikuttavan positiivisesti siihen, että ohjelmiston virheet havaittiin aikaisemmin ja tämän ansiosta ohjelmistoprojektien hallittavuus oli myös parempaa.

Haastattelututkimuksen tulokset

”[Jatkuvan integraation periaate on:] integroi ja rakenna järjestelmä monta kertaa päivässä aina kun jokin tehtävä on saatu valmiiksi” (Beck 1999b, 47)

Jatkuvan integraation hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	-	3	3	2	3	3	2,8

Haastateltavat olivat sitä mieltä, että jatkuva integraatio oli XP:n käytänteistä toiseksi hyödyllisin ja sitä myös toteutettiin yrityksissä usein esitetyn määritelmän mukaisesti. Muutamassa yrityksessä erona määritelmään oli ainoastaan se, että integrointia ei välttämättä tehty montaa kertaa päivän aikana. Haastateltavien yrityksissä jatkuva integrointi ja testaus kulkivat hyvin tiukasti yhdessä paketissa, joka näkyi myös vastauksista. Integrointi ja testaus olivat hyvin pitkälle automatisoituja yrityksissä, ja koko järjestelmän integrointi ja sen yhteyteen liittyvät testit ajettiin vähintään kerran vuorokaudessa. Yleensä raskaammat integraatiotestaukset tehtiin öisin, jolloin testattavana oli koko järjestelmä,

mukaan lukien siihen liittyvät tietokannat (A2, C1), mutta joissakin tapauksissa integraatiotestaus tehtiin aina, kun järjestelmään lisätään joku uusi pala koodia (B1, D1, D2).

Jatkuvan integraation (ja siihen tiiviisti liittyvän integraatiotestauksen) hyvinä puolina nähtiin mm. se, että sen avulla testaus ei jää pelkästään kehittäjien ajamien testien varaan, vaan jatkuva integraatio pitää huolen testien ajamisesta vähintään kerran päivässä (B1). Tämän lisäksi testit ajetaan tällöin ”täysin puhtaalta pöydältä joka kerta, joten se on luotettavampaa testausta kuin yksittäisten kehittäjien omalla koneella tehty testaus.” (B1) Jatkuvan integraation ansiosta kehittäjät ovat myös paremmin ajan tasalla siitä, missä mennään ja toimiiko kaikki odotusten mukaisesti (A2, C1). Tämä lisää myös luottamusta tehtyyn ohjelmistoon (B1). Turhat huolimattomuusvirheet huomataan ajoissa jatkuvan integraation ja siihen liittyvän testauksen ansiosta (D2), jolloin voidaan olla varmoja riittävästä laadusta (D1). Jatkuvaan integraatioon liittyvien työkalujen avulla työskentely helpottuu muullakin tavalla. Erityisen hyvänä koettiin järjestelmä, joka ilmoitti asianosaisille, että mikä meni järjestelmässä integraation jälkeen rikki (A2). Yhdessä yrityksessä jatkuvassa integraatiossa testattiin, löytyikö koodista kullekin kirjoitetulle metodille oma testitapaus ja kommentit, jolloin voitiin varmistua, että koodi pysyy luettavana ja että se tulee myös testattua (D1). Tämän lisäksi koodiin sai lisätä erityisen merkinnän ”review tagin”, jos kyseinen koodinpalanen oli huonosti kirjoitettu tai muuten epäilyttävä. Tällöin se ei mene integraatiotesteistä läpi ja koodin kirjoittajan on pakko miettiä ratkaisunsa uudelleen (D1).

Jatkuvan integraation haasteeksi muutama haastateltava mainitsi kehitettävien ohjelmistojen laajuuden. ”Jatkuva integraatio olis hienoa, jos sen sais tehtyä täydellisenä, mutta siihen menee varsin paljon ponnistelua, että pääsee edes lähelle.” (C1) Usein järjestelmät ovatkin niin laajoja, että niiden ajaminen ja

testaaminen tuotantoympäristöä vastaavassa ympäristössä ei ole enää tehokasta (C1). Laajoissa hankkeissa integroitavat palaset saattavat olla niin laajoja, että ne helposti rikkovat kokonaisuuden.

> ”Jos joku osa toteutetaan kokonaan uudestaan, niin voi olla ettei siinä toimi pitkään aikaan mikään, eikä sitä saa käännettyä eikä testattua. Silloin pahimmillaan koko jatkuvan integraation koneisto seisoo.” (A2)

Muutoin haastateltavilla ei ollut itse jatkuvasta integraatiosta negatiivista sanottavaa.

5.3.5 Jatkuva asiakkaan läsnäolo

Jatkuvalla asiakkaan läsnäololla (engl. On-Site Customer) tarkoitetaan lähinnä sitä, että asiakas on hyvin tiiviisti mukana ohjelmiston kehitysprosessissa. Alaluvussa 2.3.2 käsiteltiinkin jo asiakkaan roolia XP-projektissa, jolloin mainittiin asiakkaan yhdeksi tehtäväksi testitapausten kirjoittamisen. Ilman asiakkaan kanssa kirjoitettuja toiminnallisia testejä XP-tiimi ei voi varmasti tietää, mitä asiakas todella kehitettävältä ohjelmistolta vaatii. Kirjoitettujen testitapausten avulla XP-tiimi voi todellakin keskittyä vain asiakkaalle olennaisten toiminnallisuuksien kirjoittamiseen. Kun asiakas on projektiryhmän käytettävissä jatkuvasti, tiimi voi kysyä asiakkaan mielipidettä esimerkiksi tehtävien prioriteettiasteeseen päivittäisellä tasolla. (Beck 1999b)

Aikaisemmat empiiriset tutkimustulokset

Rumpen ja Schröderin (2002) kyselytutkimukseen osallistuneet hyödynsivät jatkuvaa asiakkaan läsnäoloa hyvin harvakseltaan, asteikolla nollasta yhdeksään käyttöaste oli vain 4,09. Begelin ja Nagappanin (2007) kyselytutkimuksessa noin 33 % vastaajista hyödynsi jatkuvaa asiakkaan läsnäoloa aina. Useimmiten syynä vähäiseen käyttöasteeseen oli se, että asiakas ei ollut yksikertaisesti käytettävissä niin usein kuin projekti olisi vaatinut (Rumpe & Schröder 2002). Vaikka Tessemin (2003) tutkimuksessakin oli välillä ongelmia asiakkaan

tavoittamisessa (asiakas ei ollut läsnä projektissa koko ajan), kaikki kuusi kyseiseen tutkimukseen osallistunutta ohjelmoijaa kokivat asiakkaan osallistumisen projektiin hyvin arvokkaana lisänä. Asiakkaan läsnäolo toi parhaimmillaan välitöntä palautetta päivittäin ohjelmiston kehityksestä, joka koettiin erittäin myönteiseksi asiaksi Gittinsin ja Hopen (2001) tutkimuksessa. Suhteessa muihin XP:n käytänteisiin asiakkaan jatkuvan läsnäolon hyödyt jäivät kuitenkin pieniksi (Svensson & Höst 2005a).

Jatkuvasti tiimin käytettävissä olevan asiakkaan rooli todettiin myös hyvin haastavaksi. Asiakkaan piti pystyä vastaamaan kehittäjien kysymyksiin hyvin nopeasti (Koskela & Abrahamsson 2004), ja hänen oli opittava esimerkiksi kirjoittamaan sopivia XP-tarinoita (Grossman, Bergin, Leip ym. 2004) ja toiminnallisuustestejä (Tessem 2003) sekä ohjailemaan niiden avulla projektia oikeaan suuntaan (Grossman, Bergin, Leip ym. 2004).

Koskelan ja Abrahamssonin (2004) toteuttamassa tapaustutkimuksessa asiakas oli todella läsnä sataprosenttisesti. Tapaustutkimuksen nelihenkinen tiimi koki tämän hyvin myönteisenä asiana ja arvosti todella asiakkaan sitoutumista projektiin. Vaikka asiakas oli koko ajan läsnä ja tavoitettavissa projektin jäsenille, todellista työaikaa projektin asioiden parissa kului vain 21 % käytetystä ajasta. Projektin lopussa kehittäjät arvioivat, että asiakkaan ei ehkä tarvitsisi olla käytettävissä aivan koko ajan, mutta kuitenkin vähintään kerran päivässä. Kehittäjät mainitsivat myös, että asiakkaan ei tarvitse olla välttämättä samassa tilassa kehittäjien kanssa, koska esimerkiksi pariohjelmointi voi häiritä asiakkaan omaa työskentelyä, mutta asiakkaan luokse tulisi kuitenkin päästä nopeasti tarpeen vaatiessa. (Koskela & Abrahamsson 2004)

Haastattelututkimuksen tulokset

”[Jatkuva asiakkaan läsnäolo:] ota tiimiin mukaan asiakas, joka voi vastata kysymyksiin milloin vain on tarvetta.” (Beck 1999b, 48). Beck (1999b) määritte-

lee asiakkaan sellaiseksi, joka todella käyttää kehitettävää tuotetta sen valmistumisen jälkeen.

Jatkuva asiakkaan läsnäolo -hyödyllisyysaste	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
	3	3	-	2	1,5	2	2,3

Haastatteluissa tuli ilmi, että mukana olevien asiakkaiden rooli ja tausta vaihteli suuresti projektista toiseen. Kaikki kuusi haastateltavaa kuitenkin totesivat yleensä projektin asiakkaan olevan talon ulkopuolinen asiakas. Muutoin asiakkaat vaihtelivat ohjelmistoalan ammattilaisista julkishallinnon edustajiin, yhdessä tapauksessa projektissa asiakasta edusti lähinnä talon sisällä oleva tuotteen omistaja (engl. product owner).

Aiempien tutkimusten tavoin haastateltavissa yrityksissä asiakkaan jatkuva läsnäolo oli harvemmin käytössä. Yhdellä haastateltavista (D2) oli kuitenkin konkreettista kokemusta työskentelystä asiakkaan kanssa muutaman vuoden takaa. Muissa yrityksissä asiakas ei ollut ollenkaan paikanpäällä ja pääsääntöisesti yhteydenpito hoidettiin joko puhelimitse tai sähköpostitse, riippuen hieman projektin luonteesta (A1, C1). Yrityksessä D molemmat haastateltavat kertoivat näkevänsä asiakasta fyysisesti paikanpäällä silloin tällöin muutaman Scrumin sprintin välein ja asiakkaaseen oltiin useammin yhteydessä puhelimitse kuin sähköpostitse, jotta ongelmat saatiin selvitettyä nopeammin (D1, D2). Yrityksessä D oltiin asiakkaaseen yhteydessä viikoittain (D1) tai keskimäärin noin kerran päivässä (D2), yrityksessä C viikoittain (C1), yrityksessä A taas useamman kerran päivässä (A2) tai hektisissä projekteissa jopa tuntitasolla (A1). Jos asiakas ei ollut ollenkaan tiimin tavoitettavissa, saattoi asiakkaan roolia hoitaa esimerkiksi Scrumissa määritelty tuotteen omistaja (A2, C1). Pääasiallisesti päivittäinen yhteydenpito koettiin riittäväksi asiakkaan ja tiimin välillä (A1, A2, C1, D2), yksi haastateltavista totesi, että on oikeastaan ihan hyvä, että asiakas ei

ole fyysisesti yrityksen tiloissa (A2). Jos yhteydenpito oli vain viikoittaisella tasolla, se oli haastateltavien mielestä liian vähäistä (D1).

Asiakkaan jatkuvan läsnäolon tai riittävän tavoitettavuuden hyvinä puolina nostettiin erityisesti esille työskentelyn tehostuminen, kun asiakas pystyy vastaamaan kysymyksiin nopeasti, jolloin myös kehitystyöhön ei tule turhia odottelujaksoja (A1, A2, C1, D2). Asiakkaan riittävä osallistuminen projektiin on myös helpottava tekijä projektipäällikön tai esimerkiksi Scrum-mestarin näkökulmasta katsottuna, koska jos jokin asia ei menekään suunnitelmien mukaan, asiakkaalta voidaan varmistaa kehitettävien ominaisuuksien prioriteettijärjestys (C1). Tämän lisäksi ei tarvitsisi käyttää niin paljon aikaa esimerkiksi Scrumiin liittyvän katselmoinnin jälkeiseen ideointitapaamiseen, jos asiakas olisi aina läsnä ja pystyisi kertomaan heti, kuinka asiat kannattaisi toteuttaa (C1, D1). Jatkuva asiakkaan läsnäolo sopii erityisen hyvin sellaisiin projekteihin, joissa on tärkeää tuntea asiakkaan toimiala hyvin (D2).

Jatkuva yhteys asiakkaan kanssa ei kuitenkaan ole haastateltavien mielestä aina tarpeellista tai edes hyödyllistä.

> ”Välillä pitää miettiä, että kannattaako aivan aina ottaa asiakkaaseen yhteyttä […], koska asiakkaalla on paljon muutakin tekemistä kuin meidän projektimme, jolloin yhteydenotto väärässä tilanteessa saattaa häiritä.” (A1)

Tässä kommentissa tulee esille yksi syy lisää siihen, miksi asiakkaan jatkuva läsnäolo on haastavaa toteuttaa yrityksissä. Asiakas saattaa olla niin kiireinen omien työtehtäviensä tai muiden samanaikaisesti meneillä olevien projektien kanssa, että hänen on mahdotonta olla aina projektin käytettävissä. Toisaalta jatkuvasti läsnä oleva asiakas voi häiritä tiimin työskentelyrauhaa esittämällä ylimääräisiä kysymyksiä tai väärinkäyttämällä suoraa yhteyttään tiimiin ja vaatimalla esimerkiksi sprinttiin ”asioita, mitkä ei kuuluis siihen” (C1). Toinen

merkittävä huomio on se, että usein tiimin jäsenet voivat tehdä useampaa projektia yhtäaikaisesti, jolloin muista projekteista keskusteleminen ja niiden tekeminen ei olisi helppoa muiden projektien asiakkaiden ollessa kuunteluetäisyydellä (D1). ”Päivittäisessä työssä idea tuntuu mahdottomalta, eli jos käytännettä toteutettaisiin täysin määritelmän mukaisesti, hyödyllisyysasteikolla sen hyödyllisyys olisi -3.” (D1) Yhden haastateltavan mielestä käytänteessä ei ole kuitenkaan merkittäviä huonoja puolia (D2).

5.4 Ketterien menetelmien käyttöönoton taso haastateltavissa yrityksissä

Tässä alaluvussa pyritään arvioimaan haastateltuja organisaatioita ja niiden tasoa aiemmin alaluvussa 2.6 esitettyjen mallien ja arvosteluasteikkojen avulla. Haastatteluihin osallistuneet yritykset olivat eri kokoluokissa. Yksi yrityksistä luokiteltiin suureksi, kun taas loppujen kolmen yrityksen henkilöstön määrä jäi alle 250 henkilön, eli ne luokiteltiin pieniksi yrityksiksi. Näissä pienissäkin yrityksissä työntekijöitä saattoi kuitenkin olla yli sata, jolloin pelkästään jo sen arviointi, kuinka laajalti yrityksessä yleensä hyödynnettiin ketteriä menetelmiä, koettiin haastavaksi tehtäväksi. Tämän lisäksi saman yrityksen sisällä kahden eri haastateltavan arviot samasta asiasta saattoivat poiketa toisistaan, joten haastattelututkimuksen tuloksia on kyseenalaista yleistää koskemaan koko yrityksen tilannetta tai katsantokantaa. Tästä syystä käytänteiden käyttöönoton tasoa ei tutkimuksessa syvennytty tutkimaan tarkemmin Scrumin tai XP:n omilla mittareilla.

Sen sijaan tutkimuksessa käytettiin Fichmanin ja Kemererin (1997) arviointiasteikkoa (TAULUKKO 1) yleisen näkemyksen selville saamiseksi ketterän kehittämisen käytänteiden käyttöönotosta. Kukin taso ja tasolle siirtymisen kriteeri on esitelty taulukossa, joka annettiin kullekin haastatelta-

valle haastattelun alkuvaiheessa ja jonka mukaisesti haastateltavat arvioivat yrityksessä käytössä olevia tai jo käytöstä poistettuja käytänteitä.

Haastattelun tulokset on esitetty tiivistetysti taulukossa 5. Monessa haastattelussa tuli esille, että tason määrittäminen ei välttämättä koske koko yritystä, vaan lähinnä haastateltavan omia projekteja ja haastateltavan omaan teknologiaan liittyviä tiimejä, kuten esimerkiksi yrityksen matkapuhelinosastoa tai vaikkapa Java-osastoa. XP:n käytänteiden kohdalla haastateltava A2 totesi asteikolta poiketen niitä kaikkia käytänteitä käytettävän aina silloin, kun se on ”mahdollista ja järkevää”.

TAULUKKO 5 Käytänteiden omaksumisen taso haastatelluissa yrityksissä

Käytänne	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**
Scrum						
Tuotteen tehtävälista (Product backlog)	5	6	6	6	3	3
Sprintin tehtävälista (Sprint backlog)	4	6	6	5	3	3
Sprintti (Sprint)	6	6	6	5	3	3
Sprintin suunnittelukokous (Sprint planning meeting)	6	6	6	5	3	3
Päivittäinen Scrum-tapaaminen (Daily Scrum)	5	6	6	6	3	3
Sprintin katselmointitapaaminen (Sprint review meeting)	5	6	6	5	3	3
Sprintin arviointitapaaminen (Sprint retrospective)	4	6	6	5	3	3
XP						
Pariohjelmointi (Pair programming)	1	-	3	7	3	2
Koodin yhteinen omistajuus (Collective code ownership)	1	-	6	6	6	5
Testivetoinen kehitys (Test-driven development)	4	-	3	5	5	2
Jatkuva integraatio (Continuous integration)	1	-	5	6	6	4
Jatkuva asiakkaan läsnäolo (On-site customer)	1	-	1	6	2	5

Taulukossa 5 esiintyvät numerot viittaavat taulukossa 1 esitettyihin tasoihin ja ne tarkoittavat seuraavaa: 1) tietoisuus, 2) kiinnostus, 3) arviointi / kokeilu, 4) sitoutuminen, 5) rajoitettu käyttöönotto, 6) yleinen käyttöönotto, 7) hylkääminen ja 8) epäjatkuvuus.

Scrumissa haastateltavien arviot käytänteiden käytön tasosta koko yrityksessä olivat käytänteiden kesken lähes kaikilla täsmälleen samalla tasolla, eli yrityk-

sissä tai haastateltavan projekteissa kaikki Scrumin käytänteet oli nähtävästi otettu samalla kertaa käyttöön ja niiden kanssa edettiin käyttöönoton tasolta toiselle hyvin samaan tahtiin (A2, B1, D1, D2). Yrityksessä D Scrumin käytänteiden käyttöönotto oli molempien haastateltavien mielestä kaikkien käytänteiden kohdalta vielä arviointi- tai kokeiluvaiheessa. Muissa yrityksissä Scrumin kaikkiin käytänteisiin oli vähintään sitouduttu noudattamaan niitä merkittävästi yhdessä tai useammassa projektissa. Kahden haastateltavan (A2 ja B1) projekteissa Scrumin kaikkia käytänteitä hyödynnettiin uudessa kehityksessä merkittävässä osassa (taso 6). Scrumin käytänteistä parhaiten yrityksissä oli omaksuttu tuotteen tehtävälista, sprintti, sprintin suunnittelukokous ja päivittäinen Scrum-tapaaminen.

XP:n osalta käytänteiden käyttöönoton tasot vaihtelivat huomattavasti Scrumia enemmän. Mielenkiintoista tuloksissa oli se, että ainoastaan yhdessä yrityksessä oli käytetty arvosteluasteikon hylkäämisen tai epäjatkuvuuden kategorisointia. Yrityksessä C pariohjelmointia oli testattu käytössä, mutta käytänteen käyttö on hylätty nykyisessä tilanteessa. Pariohjelmointia käsittelevässä alaluvussa kerrottiin, kuinka yrityksessä oli muokattu pariohjelmoinnin pohjalta oma tapa (ns. katselmointihylly) varmistaa, että tehty koodi kulkee vähintään kahden eri kehittäjän silmien kautta ennen tuotantoa. Pariohjelmoinnin kohdalla kiinnostus menetelmään sitoutumiseen merkittävissä määrin oli kuitenkin alhaista. Kaikki haastateltavat totesivat, että pariohjelmointi oli ennemminkin kokeiluasteella tai sitä hyödynnettiin vain tietyissä tilanteissa.

Koodin yhteisen omistajuuden omaksumisen taso yrityksissä oli keskiarvoisesti kaikista korkeimmalla tasolla. Kolmessa yrityksessä neljästä oli sitouduttu hyödyntämään koodin yhteistä omistajuutta säännöllisesti yrityksen projekteissa. Muiden XP:n käytänteiden osalta käyttöönoton tasot vaihtelevat suu-

resti myös saman yrityksen eri haastateltavien välillä, joten tuloksista on vaikea vetää mitään perusteltuja johtopäätöksiä.

5.5 Yhteenveto

Tässä luvussa on esitelty tutkimuksen empiirisen osan tulokset kokonaisuudessaan. Haastattelututkimuksen tuloksia on peilattu kunkin käytänteen kohdalla niiden alkuperäiseen määritelmään ja niitä on vertailtu aikaisempien tutkimustulosten kanssa. Haastatteluista ilmenee, että kaikkia käytänteitä sovelletaan haastatelluissa yrityksissä jollakin tapaa, jotta ne palvelisivat paremmin yrityksen omia tarpeita. Haastattelututkimuksella saadut tulokset ovat hyvin samansuuntaisia aiemmin toteutettujen empiiristen tutkimusten kanssa. Seuraavassa luvussa esitetään diskussio tutkielman tuloksista.

6 DISKUSSIO

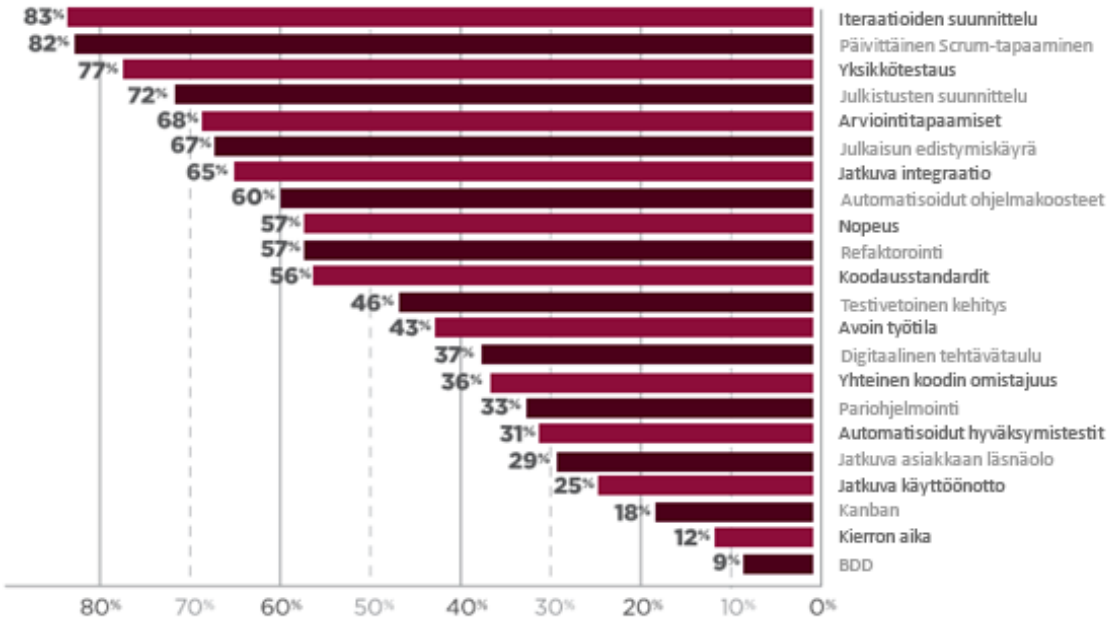
Tässä luvussa esitetään johtopäätöksiä haastattelututkimuksen tuloksista. Johtopäätökset on jaettu neljään osaan: a) ketterien käytänteiden tunnettuus ja käytön aste, b) Scrumin ja XP:n käytänteiden hyödyllisyys, c) Käytänteiden muokkaus ja d) Scrumin ja XP:n käytänteiden väliset suhteet. Luvun lopussa arvioidaan tutkimuksen reliabiliteettia ja validiteettia.

6.1 Ketterien käytänteiden tunnettuus ja käytön aste

Haastateltaville lähetettiin ennen haastattelua alustava kysely, jonka avulla kartoitettiin kolmen ketterän menetelmän tunnettuutta (kyselyn tulokset on esitetty taulukossa 2). Alustavan kyselyn perusteella voitiin todeta, että Scrumin ja XP:n käytänteet olivat tunnetumpia kuin Lean-ohjelmistokehitykseen liittyvät periaatteet. Scrumin käytänteistä kaikki olivat tuttuja haastateltaville ja kaikki olivat olleet haastateltavien käytössä. XP:n käytänteistä vähiten tunnettuja olivat metaforat, vain säännöt ja yksinkertainen suunnittelu. Läheskään kaikki XP:n käytänteet eivät olleet haastateltavien projekteissa aktiivisessa käytössä. Lean-periaatteita hyödynnettiin haastateltavien projekteissa huomattavasti harvemmin, vaikka suurin osa periaatteista oli haastateltaville kyselyn mukaan kuitenkin tuttuja.

Käytänteiden tunnettuutta ja käytön astetta on arvioitu vuosittain VersionOnen tekemässä kyselytutkimuksessa. Vuoden 2010 tulokset on esitetty kuviossa 13 (VersionOne 2010). Tutkimuksen kohteena on ollut käytänteitä kaikista ketteristä menetelmistä, joten kuviossa 13 on myös käytänteitä, jotka eivät olleet mukana haastattelututkimuksen alustavassa kyselyssä. Vuoden 2010 tuloksissa on kuitenkin samansuuntaisia viitteitä kuin mitä haastatteluissa on saatu. Esimerkiksi iteraatioiden suunnittelu, eli haastattelututkimuksessa käsitelty sprintin

suunnittelukokous, on molemmissa tutkimuksissa laajalti hyödynnetty käytänne. Myös päivittäinen Scrum-tapaaminen, arviointitapaamiset (sprintin arviointitapaaminen), jatkuva integraatio ja refaktorointi ovat molemmissa tutkimuksissa käytössä yli puolella haastateltavista. Vähemmän käytettyjä käytänteitä (alle puolet haastateltavista hyödynsi) olivat molemmissa tutkimuksissa pariohjelmointi, testivetoinen kehitys ja jatkuva asiakkaan läsnäolo. Tutkimukset erosivat toisistaan ainoastaan koodin yhteisen omistajuuden ja avoimen työtilan osalta, joita molempia hyödynnettiin haastattelututkimuksessa yli puolessa haastatteluista, kun taas VersionOnen (2010) kyselytutkimukseen osallistujien keskuudessa kyseiset käytänteet eivät olleet yhtä laajalti käytössä.



KUVIO 13 Ketterien käytänteiden käytön aste VersionOnen vuoden 2010 kyselytutkimuksessa (VersionOne 2010, 4).

6.2 Scrumin ja XP:n käytänteiden hyödyllisyys

Tutkimuksessa rajauduttiin tarkastelemaan lähemmin seitsemää Scrumin ja viittä XP:n käytännettä. Haastatteluilla pyrittiin selvittämään käsityksiä niiden hyödyllisyydestä. Yhteenveto vastauksista on esitetty taulukossa 6. Käsitykset hyödyllisyydestä on ilmaistu vaihteluvälillä -3 (selvästi haittaa) ja 3 (erittäin hyödyllinen). Scrumin ja XP:n yksittäisistä käytänteistä saadut kokemukset haastateltavien keskuudessa olivat suurimmilta osin positiivisia. Vastauksissa ei ollut yhtään miinusmerkkistä arviota. Asteikon miinuspuolta eivät hyödyntäneet myöskään Rumpen ja Schröderin (2002) tutkimukseen osallistuneet henkilöt, joilla arvosteluasteikko ulottui miinus viidestä viiteen. Negatiiviseen arvioon olisi ollut tilaisuus oikeastaan vain yhdessä haastattelussa, jossa organisaation tasolla pariohjelmointia oli arvioitu ja se oli hylätty käytänteenä nykyisessä tilanteessa. Mutta kyseisessä haastattelussa (C1) pariohjelmointi arvioitiin nollan arvoiseksi, eli siitä ei ollut erityistä haittaa, mutta ei erityistä hyötyäkään. Tällä tavalla arvioi myös haastateltava D1 sprintin arviointitapaamista sillä perusteella, että haastateltavan kokemuksen mukaan arviointitapaamisessa kerättyjä havaintoja ja parannusehdotuksia ei otettu seuraavissa projekteissa tai sprinteissä tarpeeksi hyvin huomioon, eikä tapaamisessa kerätty tieto siirtynyt riittävän hyvin projektien tai ihmisten välillä.

Arviot käytänteiden hyödyllisyydestä olivat pääosin erittäinkin positiivisia. Scrumin kaikkien käytänteiden keskiarvoinen arvio oli vähintään 2,2, johon ylsivät myös XP:n käytänteet pariohjelmointia lukuun ottamatta. Kaikki käytänteet olivat siis haastateltavien mielestä vähintään jonkin verran hyödyllisiä siten kuin he niitä projekteissaan sovelsivat.

TAULUKKO 6 Scrumin ja XP:n käytänteiden hyödyllisyyden arviointi asteikolla -3 (selvästi haittaa) ja 3 (erittäin hyödyllinen).

Käytänne	**A1**	**A2**	**B1**	**C1**	**D1**	**D2**	**KA**
Koodin yhteinen omistajuus (XP)	-	-	3	3	3	3	3
Sprintin tehtävälista (Scrum)	3	3	3	2	3	3	2,8
Jatkuva integraatio (XP)	-	3	3	2	3	3	2,8
Tuotteen tehtävälista (Scrum)	1	3	3	3	3	3	2,7
Sprintin suunnittelukokous (Scrum)	3	3	3	2	2	3	2,7
Sprintin katselmointitapaaminen (Scrum)	3	3	2	2	3	-	2,6
Päivittäinen Scrum-tapaaminen (Scrum)	3	3	1	3	1	3	2,3
Sprintti (Scrum)	3	3	2	1	3	2	2,3
Testivetoinen kehitys (XP)	2	3	1	2	3	3	2,3
Jatkuva asiakkaan läsnäolo (XP)	3	3	-	2	1,5	2	2,3
Sprintin arviointitapaaminen (Scrum)	2	3	3	3	0	2	2,2
Pariohjelmointi (XP)	1	3	2	0	3	2	1,8

Koodin yhteinen omistajuus oli käytänteistä ainoa, joka sai täydet pisteet kaikilta niiltä, joilla oli käytänteestä omakohtaista kokemusta, ja se oli täten haastateltavien mielestä hyödyllisin käsitellyistä käytänteistä. Haastateltavat kertoivat, että koodin yhteisen omistajuuden ansiosta koodin luettavuus paranee, kommentointi lisääntyy ja siten koodi on myös selkeämpää. Koodin yhteisen omistajuuden ansiosta kokonaisuuden hallinta on myös helpompaa. Ainoaksi haittapuoleksi haastateltavat mainitsivat, että yhteiseen koodiin voidaan joskus tehdä muutoksia ymmärtämättä täysin, mihin kaikkeen muutokset voivat vaikuttaa. Myös jatkuva integraatio oli suosituimpien käytänteiden joukossa. Käytänne keräsi haastateltavilta kehuja siitä, että sen ansiosta esimerkiksi testien ajamisesta vähintään kerran päivässä tulee rutiinia ja tämä taas lisää luottamusta toteutettuun ohjelmistoon. Jatkuvaa integraatiota voi olla kuitenkin vaikea toteuttaa täydessä mittakaavassa laajoissa ohjelmistoprojekteissa, varsinkin jos integraation on mentävä aina läpi. Sprintin tehtävälista oli jaetulla toisella sijalla jatkuvan integraation kanssa. Sprintin tehtävälistan arvioitiin selkeyttävän tiimin tekemistä ja tuovan rauhaa työskentelyyn. Tämän lisäksi sprintin tehtävälistan ansiosta tiimin tehtävät ovat aina samassa paikassa ja sen avulla on myös helpompi analysoida tekemistä.

Loppujen käytänteiden hyödyllisyyden arviot vaihtelivat eri haastateltavien välillä hieman enemmän, vaikka keskimäärin loputkin käytänteistä koettiin vähintään hyödyllisiksi. Pariohjelmointi sai haastateltavilta vähiten pisteitä hyödyllisyysasteikolla mitattuna, vaikkakin 1,8 pisteen keskiarvolla käytänteestä on muokattuna silti hyötyä projektille. Kaikilla haastateltavilla ei ollut konkreettista omaa kokemusta pariohjelmoinnista, eikä kukaan haastateltavista toteuttanut käytännettä määritelmän osoittamalla tavalla. Pariohjelmoinnin osalta saadut tulokset olivat aiempien empiiristen tutkimusten (esim. Grossman, Bergin, Leip ym. 2004, Hulkko & Abrahamsson 2005 ja Fitzgerald, Hartnett & Conboy 2006) kanssa hyvin pitkälti samankaltaisia, eli yritykset hyödynsivät pariohjelmointia vain tietyissä erityistilanteissa, esimerkiksi uuden työntekijän perehdytyksessä tai ns. tiukoissa tilanteissa. Pariohjelmoinnista ei nähty olevan hyötyä määritelmän mukaisesti hyödynnettynä. Toinen käytänne, jonka hyödyllisyyden arviot vaihtelivat muita käytänteitä enemmän, oli sprintin arviointitapaaminen. Yksi haastateltavista koki, että sprintin arviointitapaamisesta ei saa erityistä hyötyä, koska hänen kokemuksensa mukaan tapaamisissa kerätty tieto ei kuitenkaan siirry riittävän hyvin projektista tai tiimistä toiseen.

Kun haastattelujen tuloksia verrattiin aiemmin saatuihin empiirisiin tutkimustuloksiin, ei vertailussa havaittu mitään suuria eroja, vaan tutkielmassa kerätyn aineiston voidaan katsoa tukevan aiemmin saatuja tuloksia. Liitteessä 4 olevasta kokoavasta taulukosta ilmenee tutkimuksessa esiintyneet selkeimmät hyvät ja huonot puolet käytännekohtaisesti, ja siitä voidaan nähdä, että useasti käytänteestä on nostettu aiemmissa tutkimuksissa esille samansuuntaisia vahvuuksia tai heikkouksia, vaikka asioita haastatteluissa oli usein tarkasteltu hieman eri näkökulmasta ja käsitelty asiaa hieman eri termein.

6.3 Käytänteiden muokkaus

Haastattelututkimuksesta ilmenee, että yrityksillä on tarve ja halu muokata hyödyntämiään käytänteitä vastaamaan enemmän yrityksen omia tarpeita. Kaikkia käytänteitä oli muokattu useimmissa yrityksissä jollakin tavalla, mutta usein tehdyt muutokset eivät olleet kovin suuria. Yrityksissä käytänteisiin tehdyt muutokset on esitetty tiivistetysti liitteessä 4. Tarkimmin määritelmän mukaan hyödynnettiin sprinttejä, sprintin katselmointitapaamista ja koodin yhteistä omistajuutta. Päivittäinen Scrum-tapaaminen järjestettiin yhtä haastateltavaa lukuun ottamatta kaikilla muilla määritelmän mukaisesti. Jatkuvaan integraatioon ei ole tehty sen suurempaa muutosta kuin että sitä ei välttämättä toteutettu projekteissa useita kertoja päivän aikana.

Tehdyt muutokset liittyivät usein joko käytänteeseen käytettävään aikaan tai projektiin liittyvään asiakkaaseen. Muokatuimpia käytänteitä olivat pariohjelmointi, testivetoinen kehitys ja jatkuva asiakkaan läsnäolo. Pariohjelmointia puhtaasti määritelmän mukaan harjoittavia yrityksiä haastatteluihin ei osallistunut yhtään. Pariohjelmointia muokattiin kaikissa tapauksissa rajoittamalla sen käyttöä vain jo aiemmin mainittuihin erityistilanteisiin. Testivetoisessa kehityksessä ja jatkuvassa asiakkaan läsnäolossa suurin ero määritelmään nähden oli se, että asiakasta hyödynnettiin vähemmän kuin mitä määritelmän mukaan pitäisi tai voisi. Asiakkaan jatkuvaa fyysistä läsnäoloa ei koettu haastatteluissa tarpeelliseksi, vaan jopa haitalliseksi asiaksi, mutta kaikki haastateltavat olivat yhtä mieltä siitä, että asiakkaan oli hyvä olla tiimin tavoitettavissa vähintään kerran päivässä sähköpostitse tai puhelimitse. Asiakas oli myös harvemmin mukana testien kirjoittamisprosessissa, mikä oli useimmiten yrityksissä tehty muutos testivetoisen kehityksen määritelmään nähden.

Aiemmissa tutkimuksissa ei useinkaan ole otettu kantaa käytänteiden muokkaustapoihin, erityisesti jos kyse oli Scrumin käytänteestä. Pariohjelmoinnin osalta erilaisia toteutustapoja taas tuli esille aiemmissakin tutkimuksissa ja niissä myös pohdittiin tilanteita, joihin pariohjelmointi sopisi parhaiten. Näissäkin tutkimuksissa esille nostettiin pariohjelmoinnin toimivan parhaiten uuden työntekijän perehdytyksessä ja vain vaativien ohjelmiston osien kehittämisessä.

6.4 Scrumin ja XP:n käytänteiden väliset suhteet

Joidenkin käytänteiden osalta voitiin havaita, että ne täydensivät ja tukivat toisiaan. Esimerkkinä tästä voidaan nostaa esimerkiksi sprintti, jonka keston määritteleminen voi vaikuttaa muihinkin Scrumiin liittyvien käytänteiden onnistumiseen. Sprintti tuntui toimivan kaikissa haastatelluissa yrityksissä parhaiten kahden viikon sykleissä.

Sprintti on yksi Scrumin käytänteiden tukirangoista, jonka yhteydessä muut käytänteet toteutetaan, joten sprintin toimivuus tai toimimattomuus voi näkyä myös muissa tutkituissa käytänteissä. Erityisesti sprintin ajanjakson valinta vaikuttaa siihen, millä tasolla esimerkiksi sprintin suunnittelukokouksessa voidaan suunnitella tulevaa sprinttiä ja kuinka paljon tuotteen tehtävälistalta voidaan poimia tehtäviä sprintin tehtävälistalle. Haastatteluissa tällaisia ongelmia ei juurikaan tuotu esille, koska lähes kaikki olivat todenneet projekteissaan kahden viikon mittaisen sprintin olevan sopivan pituinen ajanjakso kehitystyölle. Kahden viikon jaksoissa sprinttiin liittyviin kokouksiin (kuten sprintin suunnittelukokous, sprintin katselmointitapaaminen ja sprintin arviointitapaaminen) tuleva käsiteltävien asioiden määrä oli vielä hyvin hahmotettavissa ja käsiteltävissä järkevässä ajassa.

Sprintin suunnittelukokouksissa ja sprintin katselmointitapaamisissa yhteistä oli, että molemmat tuovat asiakkaan tiimin kanssa samaan pöytään keskustele-

maan asioista, mikä koettiin haastatteluissa hyvänä asiana. Sprintin katselmointitapaaminen auttoi haastattelujen mukaan myös asiakkaita parantamaan suoriutumistaan sprintti kerrallaan, koska katselmointitapaamisessa he oppivat ymmärtämään, mitä asioita vaatimusmäärittelyssä ja asiakkaan palautteessa kaivattiin tiimin näkökulmasta katsottuna. Molemmissa käytänteissä ongelmallista oli asiakkaan saaminen fyysisesti paikan päälle, joten muutamassa haastattelussa sanottiin, että nämä kaksi tapaamista on ajan säästämiseksi niputettu samalle päivälle, jolloin ensin käydään läpi kuluneen sprintin aikana toteutetut toiminnallisuudet, jonka jälkeen siirrytään suunnittelemaan uutta sprinttiä.

Sprintin suunnittelukokouksessa käsiteltävien asioiden määrä ja laajuus taas vaikuttaa siihen, miten hyvin sprintin tehtävälistaa voidaan hyödyntää ja miten tarkalla tasolla tehtäviä voidaan käydä tai yleensäkään pystytään käymään läpi suunnittelukokouksen aikana. Sprintin tehtävälistan tehtävät tulevat tuotteen tehtävälistalta, jonka toimivuus tai toimimattomuus voi luonnollisesti kertautua myös sprintin tehtävälistan käsittelyyn. Myös hyödyt ovat samankaltaiset: molemmat käytänteet auttavat haastattelujen mukaan tiimiä kommunikoimaan keskenään ja asiakkaan kanssa paremmin sekä auttavat kehitettävän ohjelmiston hahmottamisessa.

XP:n käytänteissä vastaavaa riippuvuussuhdetta tai yhteneväisyyttä eri käytänteiden välillä ei tullut niinkään selkeästi esille, vaikka aiemmissa tutkimuksissa tällaisia riippuvuussuhteita onkin etsitty ja havaittu (Lui & Chan 2005). Tällaista kerrannaisvaikutusta ei voinut tunnistaa tehdyn tutkimuksen perusteella pariohjelmoinnin tai jatkuvan asiakkaan läsnäolon osalta, mutta loput kolme XP:n käytännettä näyttäisivät tukevan toinen toistaan. Testivetoisen kehityksen koettiin olevan luonnollinen pari jatkuvalle integraatiolle, koska usein integraation yhteydessä ajettiin ohjelmistoon myös etukäteen kirjoitetut

integraatiotestit. Myös koodin yhteinen omistajuus ja jatkuva integraatio muodostivat luonnollisen parin kehitystyössä.

6.5 Tutkimuksen reliabiliteetti ja validiteetti

Kvalitatiivisen tutkimuksen luotettavuutta tulee arvioida reliabiliteetin ja validiteetin suhteen. Tutkimus toteutettiin haastattelututkimuksena, jolloin tutkimuksen reliabiliteetin ja validiteetin arvioiminen on haasteellista. *Reliabiliteetti* tarkoittaa tutkimuksen toistettavuutta (Hirsjärvi, Remes & Sajavaara 2000). Tämän tutkimuksen toistettavuutta auttaa se, että tutkimusprosessi on kuvattu tarkasti alusta loppuun: tutkimuksen suunnittelu, valmistelu, tiedonkeruu ja analysointi sekä tutkimuksen raportointi. Myös alustava kysely ja haastattelurunko on esitetty (liitteet 1 ja 2). Haastattelurungon edelleen käyttämistä edesauttaa se, että se sisältää strukturoituja osia. Yksi näistä on Fichmanin ja Kemererin (1997) menetelmän käyttöönoton ja käytön tasoluokittelu. Toistettavuus ei kuitenkaan tarkoita sitä, että samaa haastattelurunkoa käytettäessä saataisiin samanlaisia tuloksia. Vastaukset riippuvat tutkimukseen osallistuvista yrityksistä ja haastateltaviksi valituista henkilöistä. Tässäkin tutkimuksessa samasta yrityksestä valitut haastateltavat antoivat jonkin verran erilaisia vastauksia.

Validiteettia voidaan tarkastella ulkoisen validiteetin ja sisäisen validiteetin näkökulmasta. *Ulkoisella validiteetilla* tarkoitetaan tulosten yleistettävyyttä (Metsämuuronen 2006). Koska kyseessä oli haastattelututkimus, jonka tulokset olivat yksittäisten haastateltavien kokemuksia ja havaintoja, eivät tutkimuksessa saadut empiiriset tulokset sinällään ole tiukasti ottaen yleistettävissä edes koskemaan haastateltavan omaa yritystä. Haastattelututkimukseen valittujen yritysten ja haastateltavien vähäinen määrä ei myöskään tue pidemmälle vietyjen yleistysten tekemistä.

Sisäisellä validiteetilla tarkoitetaan sitä, että tutkitaanko sitä mitä on tarkoitus tutkia (Metsämuuronen 2006). Onko teoria valittu oikein? Katetaanko ne alueet, jotka on tarkoitus kattaa? Ovatko käsitteet oikeita? Tämän tutkimuksen perustana ei ole ollut mitään tiettyä teoriaa. Tutkimus perustuu kuitenkin ketterää kehittämistä ja ketteriä käytäntöjä koskevaan kirjallisuuteen. Tutkimuksella tähdättiin keskeisen sisältöalueen kattamiseen. Tätä varten empiirinen osa jaettiin kahteen osaan. Ensimmäisessä kyselyn kohteena olivat kaikki Scrumin ja XP:n käytänteet ja Leanin periaatteet. Toisessa vaiheessa kyselyn kohteena olivat ne käytänteet, joista haastateltavilla oli käytännön kokemusta.

Haastateltaville annetuissa esitiedoissa ja haastatteluissa käytetyt termit ja määritelmät perustuivat alkuperäislähteisiin. Määritelmät pyrittiin pitämään lyhyinä ja selkeinä. Haastattelurunkoa testattiin koehaastattelulla, jonka perusteella runkoa hieman muokattiin ja joitakin kysymyksiä selkeytettiin. Koehaastateltavalla oli käytännön kokemusta ketterien menetelmien hyödyntämisestä ohjelmistoalalla, eli hän täytti haastateltavilta vaaditut kriteerit. Kaikki haastateltavat olivat työskennelleet ohjelmistoalan töissä vähintään neljän vuoden ajan ja ketteristä menetelmistä heillä oli kokemusta vähintään kolmen vuoden ajalta. Kaikki haastateltavat olivat tämän lisäksi toimineet eri projekteissa joko Scrum-mestarin tai projektipäällikön rooleissa, mikä antaa lisää painoarvoa heidän kokemuksilleen ja mielipiteilleen. Mahdollisia väärinymmärryksiä pyrittiin välttämään esittämällä tarkentavia kysymyksiä jo haastattelun aikana ja tämän lisäksi lähettämällä haastatteluista kerätty aineisto tarkistettavaksi tarkentavien kysymysten kera jokaiselle haastateltavalle. Tällöin haastateltavilla oli vielä mahdollisuus tarkistaa omat vastauksensa kuhunkin kysymykseen ja kommentoida niitä sekä tarkentaa niitä vastauksia, jotka jäivät haastattelijalle epäselviksi.

Haastattelurunkoa hyödynnettiin mukaillen kunkin haastateltavan omaa kokemusperää, jotta saadut tulokset olisivat todella luotettavia ja autenttisia. Haastateltavilta kysyttiin ennen varsinaista haastattelutilannetta, mistä Scrumin, XP:n ja Leanin käytänteistä heillä oli kokemusperäistä tietoa ja näiden vastausten perusteella voitiin haastattelurungosta käydä läpi vain tarvittavat kohdat. Tällöin saadut vastaukset ovat luotettavampia. Haastattelutilanteessa esitettiin kustakin käytänteestä aluksi sen lyhyt määritelmä siten kuin se on alkuperäislähteessä ilmaistu. Käytänteiden määritelmiä ei selitetty tämän enempää haastattelutilanteessa, ellei haastateltava pyytänyt itse lisäselvennystä määritelmään. Tällä pyrittiin vähentämään haastattelijan vaikutusta käytänteen määritelmään.

Tiedonkeruumuotona strukturoitu haastattelu oli soveltuva tähän tutkimukseen, koska sen avulla oli mahdollista saada esille myös sellaisia näkökulmia, joita haastattelija ei osannut ajatella etukäteen (Hirsjärvi, Remes & Sajavaara 2000). Avoin haastattelutilanne antoi mahdollisuuden esittää myös tarkentavia lisäkysymyksiä, jolloin haastattelijan väärien tulkintojen mahdollisuus pienenee. Tällaisen haastattelututkimuksen validiteettia pienentää kuitenkin puolistrukturoidun haastattelun luonne. Tutkimustavasta johtuen on mahdollista, että haastattelija vaikutti haastateltavan vastauksiin välillisesti esimerkiksi ohjaamalla keskustelua johonkin suuntaan lisäkysymyksillä tai tarkentavilla kysymyksillä. Suurin osa haastatteluista toteutettiin yrityksen omissa tiloissa, joissa muut työntekijät eivät päässeet kuulemaan haastattelutilannetta, jolloin haastateltavat saivat puhua vapaammin. Toisaalta kaikkia haastateltavia ei pyydetty suoraan henkilökohtaisesti mukaan tutkimukseen, vaan yrityksen ylempi edustaja valitsi haastatteluun sopivat henkilöt alaisistaan. Tällöin on mahdollista, että haastateltavat antoivat haastattelussa vain sellaisia lausuntoja, jotka ovat sopivia myös heidän esimiestensä nähtäväksi. Tämä vähentää tutkimuksen validiteettia.

7 YHTEENVETO

Tämän tutkimuksen tavoitteeksi oli asetettu vastata seuraaviin kysymyksiin: Mitä ketteriä käytänteitä ja millä tavoin käytetään suomalaisissa ohjelmistoalan yrityksissä? Millaisia kokemuksia niistä on saatu ja miten kokemukset vertautuvat aiempien tutkimusten tuloksiin?

Tehty tutkimus koostuu kahdesta osasta, käsitteellis-teoreettisesta ja empiirisestä osasta. Ensimmäisessä osassa esiteltiin ketterää lähestymistapaa ja kuvattiin Scrumin ja XP:n taustaa, vaiheistusta ja rooleja. Scrumin ja XP:n lisäksi käsiteltiin lyhyesti myös Lean-ohjelmistokehitystä ja Kanbania. Tämän lisäksi tarkasteltiin myös ketterän lähestymistavan käyttöönottoon liittyviä malleja. Tutkielman empiiristä osaa pohjustettiin tarkastelemalla aikaisemmin toteutettuja empiirisiä tutkimuksia ja aihealueen tutkimuksen tilannetta yleisesti. Tutkielman empiirinen osuus toteutettiin strukturoituna haastatteluna neljässä suomalaisessa ohjelmistoalan yrityksessä. Haastatteluihin osallistui yhteensä kuusi haastateltavaa, joilta kysyttiin kokemuksia Scrumin seitsemästä käytänteestä ja XP:hen liittyvästä viidestä käytänteestä.

Tutkielman kirjallisuusselvityksessä havaittiin, ettei aikaisempia vastaavia empiirisiä tutkimuksia ole juurikaan tehty, kun etsitään tutkimustuloksia erityisesti yksittäisten käytänteiden hyödyntämisestä. Sellaisia tutkimuksia kuitenkin löytyi, joissa yksittäisiäkin käytänteitä käsitellään jollakin tasolla, mutta joissa tutkimuksen pääpaino ei ole ollut juuri käytänteiden arvioinnissa. Joistakin yksittäisistä käytänteistä oli aikaisemmin tehty runsaasti tutkimusta. Tällaisia olivat pariohjelmointi ja testivetoinen kehitys. Scrumin osalta aiemmista empiirisistä tutkimuksista oli haastavaa etsiä tutkimustuloksia yksittäisten käytänteiden vaikutuksesta ohjelmistokehitykseen, koska Scrum on useimmiten käytössä kokonaisuutena. Esille nousi kuitenkin selvästi se, että laadukkaalle

empiiriselle tutkimukselle on todella tarvetta, olipa kyse sitten ketterän lähestymistavan mukaisten menetelmien tai niihin kuuluvien yksittäisten käytänteiden tutkimuksesta.

Haastattelututkimuksen tulokset olivat samansuuntaisia aikaisemmin tehtyjen tutkimusten tulosten kanssa. Selvästi poikkeavia tuloksia aiempiin tutkimuksiin nähden ei haastatteluissa noussut esille. Lähes kaikkia käsiteltyjä käytänteitä oli muokattu jollakin tavalla, jotta ne palvelisivat paremmin yrityksen omia prosesseja ja toimintatapoja. Eniten muokattuja käytänteitä olivat pariohjelmointi, jatkuva asiakkaan läsnäolo sekä tuotteen ja sprintin tehtävälistat. Esimerkiksi asiakkaan läsnäolon määrä projektissa nousi useammassakin käytänteen muokkaamisessa jollakin tavalla esille. Asiakas ei ollut useimmiten niin tiivisti mukana kehitysprosessissa kuin mitä käytänteiden määritelmän mukaisesti oli tarkoitus. Aikaisemmin paljon tutkittua pariohjelmointia hyödynnettiin vain joissakin erityistilanteissa, eivätkä haastateltavat voineet kuvitella pariohjelmointia käytettävän jatkuvasti kaiken koodin kehitystyössä. Näitä havaintoja tukevat myös aikaisemmat tutkimukset. Tuotteen ja sprintin tehtävälistoille tehdyt muutokset koskivat lähinnä tehtävien aika-arvioiden tekemistä ja tehtävien jakoa nimetyille vastuuhenkilöille. Kaikki käytänteet koettiin keskimäärin hyödyllisiksi sillä tavoin sovellettuna kuin yritykset niitä olivat muokanneet. Kolme suosituinta käytännettä olivat koodin yhteinen omistajuus, sprintin tehtävälista ja jatkuva integraatio, vaikkakin erot muiden käytänteiden hyödyllisyysasteeseen eivät olleet suuria.

Tämän tutkimuksen tulosten perusteella voidaan todeta, että Scrumilla on edelleen vahva jalansija ketterässä kehityksessä, vaikka Lean ja erityisesti Kanban ovat suosiotaan kasvattavia menetelmiä. XP:tä toteutetaan harvoin kokonaisena menetelmänä, jossa kaikki siihen kuuluvat käytänteet olisivat hyödynnettyinä yhtäaikaisesti. Ohjelmistokehityksessä XP on kuitenkin läsnä yksittäisten

käytänteiden muodossa, jotka voivat olla hyvinkin suosittuja ja pidettyjä kehitystyön työkaluja.

Vastaavanlaista tutkimusta, jossa käytänteitä käsiteltäisiin irrallaan siitä menetelmästä, johon ne kuuluvat, ei ole aikaisemmin tehty yli menetelmärajojen suomalaisissa ohjelmistoalan yrityksissä. Tutkimus tarjoaa siis yksityiskohtaisempaa tietoa siitä, missä tilanteissa käytänteitä yrityksissä hyödynnetään, miten niitä on muokattu ja miten hyödyllisiksi ne on koettu. Tutkimus kokoaa myös yhteen tietoa aikaisemmista tehdyistä tutkimuksista käytännekohtaisesti.

Käytännön työssä tutkielman tuloksista on apua pohdittaessa tiettyjen Scrumin tai XP:n käytänteiden käyttöönottoa tai käyttötapaa. Tutkielma tarjoaa Scrumista ja XP:stä yleiskuvan menetelmänä ja kuvaa niiden käytänteet ja niitä koskevat empiiriset tulokset tiiviissä paketissa. Tutkielma antaa myös lähtökohdan yksittäisten käytänteiden tarkemmalle tutkimukselle tai mallin, jonka avulla empiiristä tutkimusta voisi laajentaa koskemaan myös tämän tutkimuksen ulkopuolelle jääneitä ketteriä käytänteitä ja menetelmiä.

Tämän tutkimuksen rajoitteena on sen pieni otanta. Näin pienellä haastateltavien määrällä tuloksia on tarkasteltava lähinnä vain yksittäisten haastateltavien arvioina ja mielipiteinä, joista ei voida vetää tarkkoja johtopäätöksiä tai yleistyksiä. Jotta saataisiin yleistettävämpiä tuloksia, tulisi tutkimuksen haastattelurungon pohjalta toteuttaa laajempi haastattelu- tai kyselytutkimus.

Tutkimus antaa pohjan myös syventää tutkimusta rajatumpiin aiheisiin. Käytänteistä kysyttäviin kysymyksiin olisi voinut lisätä kysymyksen, miten haastateltavat kokevat käytänteiden hyödyllisyyden siten kuin se on määritelmässä ilmoitettu sen lisäksi, että he arvioivat sen hyödyllisyyttä muokattuna.

Haastatteluissa voisi myös kysyä, onko haastateltavilla kokemusta aiemmista epäonnistuneista käytänteiden muokkausyrityksistä ja niiden epäonnistumisten syistä. Ketterien käytänteiden tarkastelussa olisi hyvä ottaa huomioon myös tiimin tai yrityksen kypsyystaso, eli missä vaiheessa tiimin kehitystä mikäkin käytänne kannattaisi ottaa mukaan kehitysprosessiin.

LÄHTEET

Abrahamsson P., Salo O. & Ronkainen J. 2002. Agile software development methods : review and analysis. VTT Publications.

Abrahamsson P., Warsta J., Siponen M.T. & Ronkainen J. 2003. New Directions on Agile Methods: A Comparative Analysis. Teoksessa A. Jacobs & F. Titsworth (toim.) Proceedings of the 25th International Conference on Software Engineering Portland, Oregon, USA, May 3-10. Los Alamitos, CA, USA: IEEE Computer Society, 244-254. Saatavilla myös pdf-muodossa <http://doi.ieeecomputersociety.org/10.1109/ICSE.2003.1201204>.

Abrahamsson P., Hanhineva A. & Jäälinoja J. 2005. Improving Business Agility Through Technical Solutions: A Case Study on Test-Driven Development in Mobile Software Development. Teoksessa R. Baskerville, L. Mathiassen, J. Pries-Heje & J. DeGross (toim.) Business Agility and Information Technology Diffusion IFIP TC8 WG 8.6 International Working Conference May 8-11, Atlanta, Georgia, USA, IFIP International Federation for Information Processing. Boston: Springer, 227-243.

Abrahamsson P., Conboy K. & Wang X. 2009. 'Lots done, more to do': the current state of agile systems development research. European Journal of Information Systems 18(4), 281-284. Saatavilla myös pdf-muodossa <http://dx.doi.org/10.1057/ejis.2009.27>.

Agile Alliance 2001. Agile Manifesto [online]. Agile Alliance [viitattu 23.11.2009]. Saatavilla www-osoitteessa <http://agilemanifesto.org/>.

AgileCollab 2011. The Nokia Test [online]. NetSolutions [viitattu 3.8.2011]. Saatavilla www-osoitteessa <http://www.agilecollab.com/the-nokia-test>.

Agile Journal 2010. The Marriage of Lean, Scrum and Extreme Programming (XP): How to Align Agile Across an Organization [online]. CMC Media [viitattu 10.7.2011]. Saatavilla www-osoitteessa <http://www.agilejournal.com/articles/columns/column-articles/2878-the-marriage-of-lean-scrum-and-extreme-programming-xp-how-to-align-agile-across-an-organization>.

Ambler S.W. 2002. Agile Modelling : Effective Practices for eXtreme Programming and the Unified Process. New York, USA: John Wiley & Sons, Inc.

Anderson D. 2010. Kanban: successful evolutionary change for your technology business. Sequim, Washington: Blue Hole Press.

Arisholm E., Gallis H., Dybå T. & Sjøberg D. 2007. Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering 33(2), 65-86. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4052584>.

Auvinen J., Back R., Heidenberg J., Hirkman P. & Milovanov L. 2005. Improving the Engineering Process Area at Ericsson with Agile Practices. A Case Study. Turku Centre for Computer Science, TUCS Technical Report 716.

Back R., Milovanov L., Porres I. & Preoteasa V. 2002. An Experiment on Extreme Programming and Stepwise Feature Introduction. Turku Centre for Computer Science, TUCS Technical Report 451.

Bahli B. & Zeid E.S.A. 2005. The role of knowledge creation in adopting extreme programming model: an empirical study. Teoksessa M. Salem & M. El-Hadidi (toim.) ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society, 2005 Cairo, Egypt, December 5-6. IEEE, 75-87.

Baskerville R., Ramesh B., Levine L., Pries-Heje J. & Slaughter S. 2003. Is "Internet-speed" software development different? IEEE Software 20(6), 70-77.

Beck K. 1999a. Embracing change with extreme programming. IEEE Computer 32(10), 70-77. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=796139&isnumber=17277?tag=1>.

Beck K. 1999b. Extreme Programming Explained: Embrace Change. Canada: Addison Wesley Professional, 224. Saatavilla myös pdf-muodossa <http://www.inf.unioeste.br/~victor/processoIII/Metodologias/Extreme%20Programming%20Explained%20-%20Kent%20Beck.pdf>.

Begel A. & Nagappan N. 2007. Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study. Teoksessa L. O'Conner (toim.) Empirical Software Engineering and Measurement, 2007 Madrid, Spain, September 20-21. Los Alamitos, CA: IEEE Computer Society, 255-264. Saatavilla myös pdf-muodossa

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4343753&isnumber=4343709>.

Benefield R. 2010. Seven dimensions of agile maturity in the global enterprise: a case study. Teoksessa R.H. Sprague, Jr. (toim.) Proceedings of 43rd Hawaii International Conference on System Sciences Koloa, Kauai, Hawaii, January 5-8. Los Alamitos, CA: IEEE Computer Society, 1-7.

Berczuk S. 2007. Back to Basics: The Role of Agile Principles in Success with an Distributed Scrum Team. Teoksessa J. Eckstein, F. Maurer, R. Davies, G. Melnik & G. Pollice (toim.) AGILE 2007 Washington, DC, August 13-17. Los Alamitos, CA: IEEE Computer Society, 382-388. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4293565>.

Boehm B. 1988. A spiral model of software development and enhancement. Computer 21(5), 61-72.

Boehm B. & Turner R. 2003. Observations on balancing discipline and agility. Teoksessa Anonymous (toim.) Proceedings of the Agile Development Conference, 2003.Salt Lake City, Utah, USA, 25-28 June 2003. Los Alamitos, CA: IEEE Computer Society, 32-39.

Boehm B. & Turner R. 2004. Balancing Agility and Discipline: Evaluating and Integrating Agile and Plan-Driven Methods. Teoksessa F. Titsworth (toim.) Proceedings of the 26th International Conference on Software Engineering Edinburgh, UK, May 23-28. Washington, DC, USA: IEEE Computer Society, 718-719.

Canfora G., Cimitile A. & Visaggio C.A. 2004. Working in pairs as a means for design knowledge building: an empirical study. Teoksessa D. Azada (toim.) 12th IEEE International Workshop on Program Comprehension Bari, Italy, June 24-26. Los Alamitos, CA, USA: IEEE Computer Society, 62-68. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=01311048>.

Ceschi M., Sillitti A., Succi G. & De Panfilis S. 2005. Project management in plan-based and agile companies. IEEE Software 22(3), 21-27.

Chong J. 2005. Social behaviors on XP and non-XP teams: a comparative study. Teoksessa S. Kawada (toim.) Proceedings of the Agile Development Conference, 2005 Denver, Colorado, July 24-29. Los Alamitos, CA: IEEE Computer Society, 39-48. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1609803>.

Cockburn A. 2002. Agile Software Development. Boston: Addison Wesley.

Cockburn A. & Williams L. 2001. The Costs and Benefits of Pair Programming. Teoksessa G. Succi & M. Marchesi (toim.) Extreme Programming Examined, The XP Series. Boston, USA: Addison-Wesley, 223-247. Saatavilla myös pdf-muodossa <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.9064&rep=rep1&type=pdf>.

Conboy K. 2009. Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. Information Systems Research 20(3), 329-354.

Crisp AB 2010. Scrum Checklist [online]. Crisp AB [viitattu 17.7.2011]. Saatavilla www-osoitteessa <http://www.crisp.se/scrum/checklist>.

Crisp AB 2011. Kanban [online]. Crisp AB [viitattu 16.5.2011]. Saatavilla www-osoitteessa <http://www.crisp.se/kanban>.

Dalcher D., Benediktsson O. & Thorbergsson H. 2005. Development life cycle management: a multiproject experiment. Teoksessa J. Rozenblit, T. O'Neill & J. Peng (toim.) Proceedings of the 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005. Greenbelt, Maryland, April 4-7. Los Alamitos, CA: IEEE Computer Society, 289-296.

Dingsøyr T., Dybå T. & Abrahamsson P. 2008. A Preliminary Roadmap for Empirical Research on Agile Software Development. Teoksessa G. Melnik, P. Kruchten & M. Poppendieck (toim.) Proceedings of the Agile 2008 Conference Toronto, Ontario, Canada, August 4-8. Los Alamitos, CA, USA: IEEE Computer Society, 83-94. Saatavilla myös pdf-muodossa <http://doi.ieeecomputersociety.org/10.1109/Agile.2008.50>.

Dybå T., Arisholm E., Sjøberg D., Hannay J. & Shull F. 2007. Are Two Heads Better than One? On the Effectiveness of Pair Programming. IEEE Software 24(6), 12-15. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4375233>.

Dybå T. & Dingsøyr T. 2008. Empirical studies of agile software development: A systematic review. Information and Software Technology 50(9-10), 833-859. Saatavilla myös pdf-muodossa <http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V0B-4RRFN8D-1&_user=1234512&_rdoc=1&_fmt=&_orig=search&_sort=d&_docanchor=&

view=c&_searchStrId=1079820497&_rerunOrigin=scholar.google&_acct=C000052082&_version=1&_urlVersion=0&_userid=1234512&md5=bd8b822dcd8795a165e994c74431ba14>.

Erdogmus H., Morisio M. & Torchiano M. 2005. On the effectiveness of the test-first approach to programming. IEEE Transactions on Software Engineering 31(3), 226-237. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1423994&tag=1>.

Ferreira C. & Cohen J. 2008. Agile systems development and stakeholder satisfaction: a South African empirical study. Teoksessa R. Botha, C. Cilliers (toim.) Proceedings of the 2008 annual research conference of the South African Institute of Computer Scientists and Information Technologists on IT research in developing countries: riding the wave of technology Wilderness, South Africa, 6 - 8 October. New York, NY, USA: ACM, 48-55. Saatavilla myös pdf-muodossa <http://doi.acm.org/10.1145/1456659.1456666>.

Fichman R. & Kemerer C. 1997. The Assimilation of Software Process Innovations: An Organizational Learning Perspective. Management Science 43(10), 1345-1363. Saatavilla myös pdf-muodossa <http://www.jstor.org/stable/2634410>.

Fitzgerald B., Hartnett G. & Conboy K. 2006. Customising agile methods to software practices at Intel Shannon. European Journal of Information Systems 15(2), 200-213. Saatavilla myös pdf-muodossa <http://teaching.fec.anu.edu.au/infs8005/R110.pdf>.

Gittins R. & Hope S. 2001. A study of Human Solutions in eXtreme Programming. Teoksessa E. Kadoda (toim.) Proceedings of 13th Workshop of the Psychology of Programming Interest Group Bournemouth, UK, April 18- 20. www.ppig.org, 41-51. Saatavilla myös pdf-muodossa <http://www.ppig.org/papers/13th-gittins.pdf>.

Gittins R., Hope S. & Williams I. 2002. Qualitative Studies of XP in a Medium Sized Business. 2nd International conference on eXtreme Programming and Flexible Processes in Software Engineering, Villasimius, Sardinia, Italy. Saatavilla www-muodossa <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.9132&rep=rep1&type=pdf>.

Grossman F., Bergin J., Leip D., Merritt S. & Gotel O. 2004. One XP experience: introducing agile (XP) software development into a culture that is willing

but not ready. Teoksessa H. Lutfiyya, J. Singer & D. Stewart (toim.) Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research Markham, Ontario, Canada, October 5-7. IBM Press, 242-254. Saatavilla myös pdf-muodossa <http://delivery.acm.org/10.1145/1040000/1034933/p242-grossman.pdf?key1=1034933&key2=7524795621&coll=GUIDE&dl=GUIDE&CFID=77669921&CFTOKEN=33581700>.

Highsmith J. 2000. Adaptive software development: a collaborative approach to managing complex systems. New York, USA: Dorset House Publishing Co.

Hirsjärvi S., Remes P. & Sajavaara P. 2000. Tutki ja kirjoita. 6. uud. laitos, Helsinki: Kirjayhtymä Oy.

Hulkko H. & Abrahamsson P. 2005. A multiple case study on the impact of pair programming on product quality. ICSE '05: Proceedings of the 27th international conference on Software engineering St. Louis, MO, USA, May 15-21. New York, NY, USA: ACM, 495-504. Saatavilla myös pdf-muodossa <http://delivery.acm.org/10.1145/1070000/1062545/p495-hulkko.pdf?key1=1062545&key2=0530995621&coll=GUIDE&dl=GUIDE&CFID=77694115&CFTOKEN=69087043>.

Ikonen M. 2010. Leadership in Kanban Software Development Projects: A Quasi-controlled Experiment. Teoksessa P. Abrahamsson & N. Oza (toim.) Lean Enterprise Software and Systems. First International Conference, LESS 2010, Helsinki, Finland, October 17-20. Berlin Heidelberg: Springer, 85-98.

Ilieva S., Ivanov P. & Stefanova E. 2004. Analyses of an agile methodology implementation. Teoksessa R. Steinmetz & A. Mauthe (toim.) Proceedings of the 30th Euromicro Conference Rennes, France, August 31 - September 3. Los Alamitos, CA, USA: IEEE Computer Society, 326-333. Saatavilla myös pdf-muodossa <http://research.uni-sofia.bg/bitstream/10506/11/1/SPPI04_31_ilievas.pdf>.

InfoQ 2011. Scrum checklist 2011 [online]. InfoQ [viitattu 17.7.2011]. Saatavilla www-osoitteessa <http://www.infoq.com/resource/minibooks/scrum-checklists/en/pdf/Scrum%20CheckList%202011.pdf>.

Kettunen P. & Laanti M. 2005. How to steer an embedded software project: tactics for selecting the software process model. Information and Software Technology 47(9), 587-608.

Kniberg H. & Skarin M. 2010. Kanban and Scrum - making the most of both. http://www.infoq.com/minibooks/kanban-scrum-minibook: C4Media, 122-1. Saatavilla myös pdf-muodossa <http://www.infoq.com/minibooks/kanban-scrum-minibook>.

Komi-Sirviö S. & Tihinen M. 2005. Lessons learned by participants of distributed software development. Knowledge and Process Management 12(2), 108-122.

Korhonen K. 2009. Migrating Defect Management from Waterfall to Agile Software Development in a Large-Scale Multi-site Organization: A Case Study. Teoksessa P. Abrahamsson, M. Marchesi, F. Maurer, W. Aalst, J. Mylopoulos, M. Rosemann, M.J. Shaw & C. Szyperski (toim.) Agile Processes in Software Engineering and Extreme Programming. Proceedings of 10th International Conference, XP 2009, Pula, Sardinia, Italy, May 25-29. Lecture Notes in Business Information Processing. Berlin, Heidelberg: Springer, 73-82.

Korkala M., Abrahamsson P. & Kyllönen P. 2006. A Case Study on the Impact of Customer Communication on Defects in Agile Software Development. Teoksessa F. Maurer & G. Melnik (toim.) Proceedings of AGILE 2006 Conference Minneapolis, Minnesota, July 23-28. IEEE Computer Society, 76-88.

Koskela J. & Abrahamsson P. 2004. On-Site Customer in an XP Project: Empirical Results from a Case Study. Teoksessa T. Dingsøyr (toim.) Software Process Improvement, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1-11. Saatavilla myös pdf-muodossa <http://dx.doi.org/10.1007/978-3-540-30181-3_1>.

Kroll P. & Kruchten P. 2003. The Rational Unified Process made easy: A practitioner's guide to the RUP. Addison Wesley Professional.

Kruchten P. 1996. A Rational Development Proces. Crosstalk 9(7), 11-16.

Kähkönen T. 2004. Agile methods for large organizations - building communities of practice. Agile Development Conference, 2004 Salt Lake City, Utah, June 22-26. Los Alamitos, CA: IEEE Computer Society, 2-10.

Ladas C. 2009. Scrumban - Essays on Kanban Systems for Lean Software Development. USA: Modus Cooperandi Press.

Larman C. & Basili V. 2003. Iterative and incremental developments, a brief history. Computer 36(6), 47-56. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1204375>

Layman L., Williams L. & Cunningham L. 2004. Exploring extreme programming in context: an industrial case study. Agile Development Conference, 2004 Salt Lake City, Utah, June 22-26. Los Alamitos, CA: IEEE Computer Society, 32-41. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1359793&isnumber=29819>.

Leffingwell D. 2007a. The Essence of XP. Teoksessa D. Leffingwell (toim.) Scaling Software Agility: Best Practices for Large Enterprises, The Agile Software Development Series. USA: Addison Wesley Professional, 29-40.

Leffingwell D. 2007b. Lean Software, DSDM, and FDD. Teoksessa D. Leffingwell (toim.) Scaling Software Agility: Best Practices for Large Enterprises, The Agile Software Development Series. USA: Addison Wesley Professional, 63-73.

Li J., Moe N.B. & Dybå T. 2010. Transition from a plan-driven process to Scrum: a longitudinal case study on software quality. Teoksessa G. Succi, M-Morisio & N. Nagappan (toim.) Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, Bolzano-Bozen, Italy, September 16-17. New York, NY, USA: ACM, 13:1-13:10. Saatavilla myös pdf-muodossa <http://doi.acm.org/10.1145/1852786.1852804>.

Lindström L. & Jeffries R. 2004. Extreme programming and agile software development methodologies. Information Systems Management 21(3), 41-52. Saatavilla myös pdf-muodossa <http://www.oobeyagroup.com/images/ExtremeProgrammingAgileSoftwareDevelopmentLindstromJeffries.pdf>.

Lui K. & Chan K. 2005. A Road Map for Implementing eXtreme Programming. Teoksessa M. Li, B. Boehm & L. Osterweil (toim.) Unifying the Software Process Spectrum, Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 474-481.

MacKenzie A. & Monk S. 2004. From Cards to Code: How Extreme Programming Re-Embodies Programming as a Collective Practice. Computer Supported Cooperative Work 13(1), 91-117.

Mann C. & Maurer F. 2005. A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction. Teoksessa S. Kawada (toim.) ADC '05: Proceedings of the Agile Development Conference Denver, Colorado, USA, July 24-29. Washington, DC, USA: IEEE Computer Society, 70-79. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1609806>.

Mannaro K., Melis M. & Marchesi M. 2004. Empirical Analysis on the Satisfaction of IT Employees Comparing XP Practices with Other Software Development Methodologies. Teoksessa G. Goos, J. Hartmanis & J. van Leeuwen (toim.) Extreme Programming and Agile Processes in Software Engineering, Garmisch-Partenkirchen, Germany, June 6-10. Berlin, Heidelberg: Springer, 166-174. Saatavilla myös pdf-muodossa <http://www.springerlink.com/content/6ku47eumv9hmkcxw/fulltext.pdf>.

Marcal A., De Freitas B., Soares F., Marciel T. & Belchior A. 2008. Blending Scrum practices and CMMI project management process areas. Innovations in Systems and Software Engineering 4(1), 17-29. Saatavilla myös pdf-muodossa <http://www.springerlink.com/content/u34hng336827w284>.

Marchenko A. & Abrahamsson P. 2008. Scrum in a Multiproject Environment: An Ethnographically-Inspired Case Study on the Adoption Challenges. Teoksessa G. Melnik, P. Kruchten & M. Poppendieck (toim.) AGILE '08. Conference Toronto, ON, August 4-8. Los Alamitos, CA: IEEE Computer Society, 15-26. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4599449>.

Martin A., Biddle R. & Noble J. 2004. The XP customer role in practice: Three studies. Agile Development Conference 2004 Salt Lake City, USA, June 22-26. Los Alamitos, CA: IEEE Computer Society, 42-54. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1359794&isnumber=29819>.

Melnik G. & Maurer F. 2004. Introducing agile methods: three years of experience. Teoksessa R. Steinmetz & A. Mauthe (toim.) Proceeding of the 30th Euromicro Conference, 2004.Rennes, France, 31 August - 3 September. Los Alamitos, CA: IEEE Computer Society, 334-341. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1333388>.

Merisalo-Rantanen H., Tuunanen T. & Rossi M. 2005. Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases. Journal of Database Management 16(4), 326-333. Saatavilla myös pdf-muodossa

<http://213.250.107.20/ttt/Downloads/Old%20Wine%20in%20A%20New%20Bottle.pdf>.

Metsämuuronen J. 2006. Laadullisen tutkimuksen käsikirja. Jyväskylä: Gummerus Kirjapaino.

Mirakhorli M., Khanipour Rad A., Shams F., Pazoki M. & Mirakhorli A. 2008. RDP technique: a practice to customize xp. Teoksessa P. Kruchten & S. Adolph (toim.) APOS '08: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral Leipzig, Germany, May 10-18. New York, NY, USA: ACM, 23-32. Saatavilla myös pdf-muodossa <http://portal.acm.org/citation.cfm?doid=1370143.1370149>.

Moe N.B., Dingsøyr T. & Dybå T. 2010. A teamwork model for understanding an agile team: A case study of a Scrum project. Information and Software Technology 52(5), 480-491. Saatavilla myös pdf-muodossa <http://dx.doi.org/10.1016/j.infsof.2009.11.004>.

Montesi M. & Lago P. 2007. Software engineering article types: An analysis of the literature. Journal of Systems and Software 81(10), 1694-1714. Saatavilla myös pdf-muodossa <http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V0N-4R8H1K1-2-4&_cdi=5651&_user=1234512&_pii=S0164121207003184&_orig=search&_coverDate=10%2F31%2F2008&_sk=999189989&view=c&wchp=dGLzVzz-zSkWA&md5=330d77d353aeff7477f0b8aa5aa28404&ie=/sdarticle.pdf>.

Nonaka I. & Takeuchi H. 1986. The New New Product Development Game. Harward Business Review January-February 1986, 137-146. Saatavilla myös pdf-muodossa <http://www.iei.liu.se/fek/frist/723g18/articles_and_papers/1.107457/TakeuchiNonaka1986HBR.pdf>.

Nosek J.T. 1998. The case for collaborative programming. Communications of the ACM 41(3), 105-108. Saatavilla myös pdf-muodossa <http://doi.acm.org/10.1145/272287.272333>.

O'Reilly T. 1999. Lessons from open-source software development. Communications of the ACM 42(4), 32-37. Saatavilla myös pdf-muodossa <http://doi.acm.org/10.1145/299157.299164>.

Paasivaara M., Durasiewicz S. & Lassenius C. 2008. Using scrum in a globally distributed project: a case study. Software Process: Improvement and Practice 13(6), 527-544. Saatavilla myös pdf-muodossa

<http://www3.interscience.wiley.com/cgi-bin/fulltext/121526963/PDFSTART>.

Packlick J. 2007. The Agile Maturity Map A Goal Oriented Approach to Agile Improvement. Teoksessa J. Eckstein, F. Maurer, R. Davies, G. Melnik & G. Pollice (toim.) AGILE 2007 Washington, DC, August 13-17. Los Alamitos, CA: IEEE Computer Society, 266-271.

Palmer S.R. & Felsing J.M. 2002. A Practical Guide to Feature-Driven Development. USA: Prentice Hall.

Patel C. & Ramachandran M. 2009. Agile Maturity Model (AMM): A Software Process Improvement framwork for Agile Software Development Practices. International Journal of Software Engineering 2(1), 3-28. Saatavilla myös pdf-muodossa <http://www.doaj.org/doaj?func=abstract&id=564138>.

Paulk M. 2001. Extreme programming from a CMM perspective. IEEE Software 18(6), 19-26.

Pikkarainen M. & Mäntyniemi A. 2006. An Approach for Using CMMI in Agile Software Development Assessments: Experiences from Three Case Studies. SPICE 2006 Conference Luxemburg, May 4-5.

Poppendieck M. & Poppendieck T. 2003. Lean Software Development: An Agile Toolkit for Software Development Managers. USA: Addison-Wesley Professional.

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

Robinson H. & Sharp H. 2005. Organisational culture and XP: three case studies. Agile Conference, 2005. Proceedings Denver, Colorado, July 24-29. Los Alamitos, CA: IEEE Computer Society, 49-58. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1609804>.

Rohunen A., Rodriguez P., Kuvaja P., Krzanik L. & Markkula J. 2010. Approaches to Agile Adoption in Large Settings: A Comparison of the Results from a Literature Analysis and an Industrial Inventory. Teoksessa M. Ali Babar, M. Vierimaa & M. Oivo (toim.) Product-Focused Software Process Improvement. Proceedings of 11th International Conference,

PROFES 2010, Limerick, Ireland, June 21-23. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 77-91.

Royce W. 1970. Managing the development of large software systems. IEEE Wescon Proceedings Los Angeles, USA, August 25-28. IEEE Wescon, A/1-1-A/1-9. Saatavilla myös pdf-muodossa <http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf>.

Rumpe B. & Schröder A. 2002. Quantitative Survey on Extreme Programming Projects. Proceedings of the 3rd international conference on extreme programming and flexible processes in software engineering (XP 2002) Alghero, Italy, May 26-30. Springer, 95-100. Saatavilla myös pdf-muodossa <https://www.agilealliance.org/system/article/file/1043/file.pdf>.

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. Software, IET 2(1), 58-64. Saatavilla myös pdf-muodossa <http://dx.doi.org/10.1049/iet-sen:20070038>.

Schwaber K. 1995. SCRUM Development Process. Teoksessa D. Patel, C. Casanave, G. Hollowell, J. Miller & J. Sutherland (toim.) Business Object Design and Implementation: OOPSLA'95 Workshop Proceedings Austin, Texas, USA, October 15-19. Springer, 1-23. Saatavilla myös pdf-muodossa <http://jeffsutherland.com/oopsla/schwapub.pdf>.

Schwaber K. 2004. Agile Project Management with Scrum. Redmond Washington: Microsoft Press.

Schwaber K. & Beedle M. 2002. Agile software development with Scrum. Upper Saddle River, NJ: Prentice Hall.

Schwaber K. & Sutherland J. 2010. Scrum. Scrum.org, 1-21. http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf [viitattu 20.05.2011]

Sidky A., Arthur J. & Bohner S. 2007. A disciplined approach to adopting agile practices: the agile adoption framework. London : Springer, 203-216.

Siniaalto M. & Abrahamsson P. 2007. A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage. Teoksessa L. O'Conner (toim.) ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement Madrid,

Spain, September 20-21. Washington, DC, USA: IEEE Computer Society, 275-284. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4343755>.

Sjøberg D., Dybå T. & Jørgensen M. 2007. The Future of Empirical Methods in Software Engineering Research. Teoksessa L. Briand & A. Wolf (toim.) International Conference on Software Engineering: 2007 Future of Software Engineering Minneapolis, May 23-25. Washington, DC, USA: IEEE Computer Society, 358-378. Saatavilla myös pdf-muodossa <http://dx.doi.org/10.1109/FOSE.2007.30>.

Stapleton J. 1997. Dynamic systems development method - The method in practice. Addison Wesley.

Sutherland J. & Schwaber K. 2007. The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. http://scrumtraininginstitute.com/: Scrum Training Institute.

Sutherland J., Viktorov A., Blount J. & Puntikov N. 2007. Distributed scrum: agile project management with outsourced development teams. Teoksessa R.H. Sprague, Jr. (toim.) 40th Annual Hawaii International Conference on System Sciences (HICSS'07) Big Island, Hawaii, January 3-6. IEEE Computer Society, 274a. Saatavilla myös pdf-muodossa <http://www.computer.org/portal/web/csdl/doi/10.1109/HICSS.2007.180>.

Svensson H. & Höst M. 2005a. Introducing an Agile Process in a Software Maintenance and Evolution Organization. Ninth European Conference on Software Maintenance and Reengineering Manchester, March 21-23. Los Alamitos, CA: IEEE Computer Society, 256-264. Saatavilla myös pdf-muodossa <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1402140>.

Svensson H. & Höst M. 2005b. Views from an Organization on How Agile Development Affects Its Collaboration with a Software Development Team. Teoksessa F. Bomarius & S. Komi-Sirviö (toim.) Product Focused Software Process Improvement, Lecture Notes in Computer Science. Berlin: Springer Verlag, 21-52.

Tessem B. 2003. Experiences in Learning XP Practices: A Qualitative Study. Teoksessa M. Marchesi & G. Succi (toim.) Extreme Programming and Agile Processes in Software Engineering, Lecture notes in Computer Science. Berlin, Heidelberg: Springer, 131-137. Saatavilla myös pdf-muodossa <http://www.springerlink.com/content/6g7l2tddp16677ng/fulltext.pdf>.

Tichy W. 2011. Empirical software research: an interview with Dag Sjøberg, University of Oslo, Norway. Ubiquity 2011(June), 2:1-2:14. Saatavilla myös pdf-muodossa <http://doi.acm.org/10.1145/1998372.1998374>.

Tingling P. & Saeed A. 2007. Extreme Programming in Action: A Longitudinal Case Study. Teoksessa J. Jacko (toim.) Human-Computer Interaction. Interaction Design and Usability, Lecture Notes in Computer Science. Heidelberg: Springer Berlin, 242-251. Saatavilla myös pdf-muodossa <http://www.springerlink.com/content/v0285t72x0274406/>.

van der Storm T. 2008. Backtracking Incremental Continuous Integration. Teoksessa K. Kontogiannis, C. Tjortjis & A. Winter (toim.) Proceedings of 12th European Conference on Software Maintenance and Reengineering Athens, Greece, April 1-4. IEEE Computer Society, 233-242. Saatavilla myös pdf-muodossa <http://www.ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4493318>.

Vanderburg G. 2005. A simple model of agile software processes - or - Extreme programming annealed. ACM SIGPLAN NOTICES 40(10), 539-545.

VersionOne 2006. Survey: "The State of Agile Development" [online]. VersionOne [viitattu 20.05.2011]. Saatavilla www-osoitteessa <http://www.versionone.com/pdf/StateofAgileDevelopmentSurvey.pdf>.

VersionOne 2007. 2nd Annual Survey "The State of Agile Development" [online]. VersionOne [viitattu 20.05.2011]. Saatavilla www-osoitteessa <http://www.versionone.com/pdf/StateOfAgileDevelopmet2_FullDataReport.pdf>.

VersionOne 2008. 3rd Annual Survey "The State of Agile Development" [online]. VersionOne [viitattu 20.05.2011]. Saatavilla www-osoitteessa <http://www.versionone.com/pdf/3rdAnnualStateOfAgile_FullDataReport.pdf>.

VersionOne 2009. 4th Annual Survey "The State of Agile Development" [online]. VersionOne [viitattu 20.05.2011]. Saatavilla www-osoitteessa <http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf>.

VersionOne 2010. 5th Annual Survey "The State of Agile Survey" [online]. VersionOne [viitattu 20.05.2011]. Saatavilla www-osoitteessa <http://www.versionone.com/state_of_agile_development_survey/10/>.

Wellington C.A., Briggs T. & Girard C.D. 2005. Comparison of student experiences with plan-driven and agile methodologies. Proceedings 35th Annual Conference Frontiers in Education, 2005. FIE '05.Indianapolis, IN, October 19-22. Champaign, IL: IEEE Computer Society, T3G-18.

West D. & Grant T. 2010. Agile development mainstream adoption has changed agility [online], [viitattu 14.5.2011]. Saatavilla pdf-muodossa <http://www.arcserve.com/~/media/Files/IndustryAnalystReports/agile_development_mainstream_232616.pdf>.

LIITE 1 ALUSTAVA KYSELY

Pyytäisin aluksi laittamaan seuraavaan listaan kunkin käytänteen jälkeen merkinnän

- K jos käytänne on yrityksessä käytössä vakituisesti,
- O jos käytänne on joskus ollut esim. pilottiprojektissa mukana, mutta ei enää vakituisessa käytössä jostakin syystä,
- E jos käytänteestä ei ole minkäänlaisia käyttökokemuksia yrityksessä,
- X jos käytänne ei ole ollenkaan tunnettu.

Näiden merkintöjen lisäksi voi vapaasti kommentoida käyttöastetta tai muuta vastaavaa. Tarvittaessa annan lisätietoja kunkin käytänteen tarkoituksesta ja käyttötavasta, jos nimien perustella ne eivät ole aivan selviä.

Extreme Programming -nimiseen ketterään menetelmään liittyvät käytänteet:

- Pariohjelmointi (pair programming)
- Koodin yhteinen omistajuus (collective code ownership)
- 40-tunnin työviikko
- Testivetoinen kehitys (test-driven development, TDD)
- Jatkuva integraatio (continuous integration)
- Refaktorointi
- Suunnittelupeli (planning game)
- Pienet julkistukset (small releases)
- Metaforat
- Yksinkertainen suunnittelu (simple design)
- Jatkuva asiakkaan läsnäolo (on-site customer)
- Avoin työtila (open workspace)
- Vain säännöt (just rules)

Scrum -nimiseen ketterään menetelmään liittyvät käytänteet:

- Tuotteiden tehtävälista (product backlog)
- Työmäärän arviointi (effort estimation)
- Sprintti
- Sprintin suunnittelukokous (Sprint planning meeting)
- Sprintin tehtävälista (Sprint backlog)
- Päivittäinen Scrum-tapaaminen (daily Scrum meeting)
- Sprintin katselmointitapaaminen (Sprint review meeting)
- Sprintin arviointitapaaminen (Sprint retrospective meeting)

Lean software development -nimiseen ketterään menetelmään liittyvät käytänteet:

- vältä jätettä (avoid waste)
- paranna oppimista (amplify learning)
- päätä niin myöhään kuin mahdollista (decide as late as possible)
- toimita niin nopeasti kuin mahdollista (deliver as fast as possible)
- anna valtaa tiimille (empower the team)
- tee eheää työtä (build integrity in)
- näe kokonaisuus (see the whole)

LIITE 2 HAASTATTELURUNKO

Yleiset organisaatioon ja haastateltavan taustaan liittyvät kysymykset

- *Kerrotaan hieman taustaa tutkimuksesta, johon haastattelu liittyy.*
- *Millä perusteella mukanaolevat käytänteet on valittu?*
- *Saako haastattelun nauhoittaa?*

Haastattelujen sisällön luottamuksellisuus:

  - *Haastattelun nauhoite ja sen auki kirjoitettu osuus on vain tekijän ja ohjaajan tiedossa.*
  - *Nauhoite tuhotaan ja haastattelun sisältö muokataan sellaiseksi, että se erittelee vain yrityksen (kirjaimilla A-D) ja haastateltavan henkilön (numerolla 1 tai 2).*
    - *Haastattelusta tehtävät muistiinpanot lähetetään muutaman työpäivän kuluessa haastateltavalle hyväksyttäväksi / kommentoitavaksi.*
  - *Tutkielma julkaistaan vapaasti kaikkien luettavaksi, niin että yritystä tai haastateltavaa ei voi tutkielmasta kukaan ulkopuolinen päätellä.*

- Haastateltavan tiedot
  - Nimi
  - Kokemus ohjelmistotuotannon töistä (vuosina)?
  - Kuinka monessa ketterässä projektissa olet työskennellyt?
  - Kokemus ketteristä menetelmistä?
    - aloittelija (alle vuosi)
    - kokenut (1-3 vuotta)
    - ekspertti (yli 3 vuotta)
  - Rooli ketterää menetelmää hyödyntävässä tiimissä:
    - projektipäällikkö
    - tiimin jäsen
    - Scrum-mestari
- Tiimin tiedot (lähinnä siltä osin, mistä kokemukset ovat peräisin)
  - Minkä kokoisia tiimit ovat olleet keskimäärin?
  - Tiimin jäsenten keskimääräinen kokemustaso ohjelmistokehityksessä
    - (aloittelijat, kokeneet, ekspertit)
  - Projektin kehityskohde?
  - Onko hajautettu projekti?
  - Millainen asiakas projektissa on mukana?
- Yritys
  - Koko:
    - Pieni
    - Keskisuuri (250)
    - Suuri
  - Kuinka suuri osa projekteista ketteriä? (%)
- *Käydään läpi kunkin käytänteen kohdalta sen omaksumisen taso yrityksessä listan avulla*

Kaikkia käytänteitä koskevat yleiset kysymykset

Esitellään ensin käytänteen määritelmä lyhyesti

- Onko käytännettä muokattu projektikohtaisesti? Miten?
- Kerro käytänteen hyvät puolet
- Kerro käytänteen huonot puolet / ongelmat
- Onko käytänteestä mielestäsi hyötyä vai haittaa?
  - Hyöty asteikolla -3…0…3 (-3 haittaa paljon, 0 ei merkitystä, 3 suuri hyöty)

Scrumiin liittyvät erityiset kysymykset

Tuotteen tehtävälista / Product Backlog -3 -2 -1 0 1 2 3

- Kuka tehtävät asettaa?
- Kuinka tehtävät priorisoidaan?
- Kuinka usein tehtäviä päivitetään tuotteen tehtävälistaan?

Sprintin tehtävälista / Sprint Backlog -3 -2 -1 0 1 2 3

- Kuka valitsee tehtävät tuotteen tehtävälistasta sprintin tehtävälistaan?
- Muokataanko tehtäviä, vai tulevatko ne sellaisenaan tuotteen tehtävälistasta?
- Miten sprinttiin osataan valita sopivan kokoisia tehtäviä, eli miten tehtävät arvotetaan ajallisesti?
    - Planning poker / tuntiarviot / tehtävien pilkkominen…
- Kuinka usein tehtäviä päivitetään sprintin tehtävälistassa?
    - Kuka tehtäviä päivittää?

Sprintti / Sprint -3 -2 -1 0 1 2 3

- Kuinka kauan yksi sprintti yleensä kestää?

Sprintin suunnittelukokous / Sprint Planning Meeting -3 -2 -1 0 1 2 3

- Ketkä suunnittelukokoukseen osallistuvat?
- Kokouksen kulku?
    - Kuka kokousta johtaa?
    - Tehdäänkö kirjattuja päätöksiä?
    - Mitä dokumentteja kokouksessa syntyy?
- Onko kokouksille määrätty maksimipituutta?

Päivittäinen Scrum-tapaaminen / Daily Scrum -3 -2 -1 0 1 2 3

- Ketkä tapaamiseen yleensä osallistuvat?
- Käytetäänkö kolmea kysymystä?
    - (Mitä olet tehnyt, mitä teet seuraavaksi ja mitä vaikeuksia tai esteitä on ollut työssäsi?)
    - Käsitelläänkö ilmenneitä ongelmia myöhemmin jossakin jatkopalaverissa?
- Venyvätkö tapaamiset?

Sprintin katselmointitapaaminen / Sprint Review meeting -3 -2 -1 0 1 2 3

- Ketkä tapaamiseen osallistuvat?
- Mikä rooli kullakin henkilöllä on tapaamisessa?
- Miten kehitettävää tuotetta käydään läpi katselmoinnissa?
    - Millä tasolla, kuinka tarkasti yksityiskohtia käydään läpi?
- Asiakkaan mielipide katselmointitapaamisesta?

Sprintin arviointitapaaminen / Sprint Retrospective meeting -3 -2 -1 0 1 2 3

- Järjestetäänkö virallista arviointitapaamista, vai käydäänkö näitä asioita läpi epävirallisesti jossakin muualla (esim. kahvihuoneessa)?
- Ketkä arviointitapaamiseen osallistuvat?
- Toteutuuko jokaisen Sprintin jälkeen?
- Kirjataanko muutosehdotuksia?
    - Minne?
    - Kenelle ne näkyvät?
- Kuinka usein muutosehdotukset otetaan käytäntöön?

XP:hen liittyvät erityiset kysymykset

Pariohjelmointi / Pair programming -3 -2 -1 0 1 2 3

- Miten pariohjelmointi toteutetaan? (yhdellä koneella/kahdella koneella vierekkäin tmv.)
- Kuinka usein pariohjelmointia hyödynnetään?
- Millaisia ongelmia ratkotaan pariohjelmoinnin avulla?
- Miten parit yleensä muodostetaan?
    - Miten parin kokemustausta vaikuttaa pariohjelmoinnin tekemiseen?

Koodin yhteinen omistajuus /Collective code ownership -3 -2 -1 0 1 2 3

- Minkä tyyppiset koodit ovat yhteisomistuksessa?
- Missä määrin rinnalla käytetään vastuuhenkilöä tietylle koodille?
- Onko vaikuttanut positiivisesti koodin luettavuuteen?

Testivetoinen kehitys / Test-driven development -3 -2 -1 0 1 2 3

- Testivetoisesta kehityksestä sanotaan olevan hyötyä siinä, että
    - se auttaa hahmottamaan seuraavaksi kirjoitettavaa koodia
    - sen avulla löydetään virheet helpommin
    - miten näet, että hyöty näiden välillä jakaantuu? (%)
- Miten testivetoista kehitystä toteutetaan projekteissa?
- Kuka kirjoittaa testit?

Jatkuva integraatio / Continuous integration -3 -2 -1 0 1 2 3

- Kuinka usein integrointia tehdään?

Jatkuva asiakkaan läsnäolo / On-site customer -3 -2 -1 0 1 2 3

- Kuka hoitaa asiakkaan roolin projekteissa? (erityisesti jos varsinaista ulkopuolista asiakasta ei ole)
- Onko asiakas läsnä fyysisesti?
    - Jos ei, mikä on pääasiallinen yhteydenpitotapa/-väline?
- Kuinka usein asiakkaan kanssa ollaan tekemisissä?
    - Viikkotasolla / päivätasolla / tunteina?
- Kuka on yleensä asiakkaaseen yhteydessä?
- Onko yhteydenpito mielestäsi riittävää?

Haastattelun jälkeiset asiat

Kiitos haastatteluun osallistumisesta!

Tiivistelmä haastattelussa käydyistä olennaisista asioista lähetetään haastateltavalle muutaman päivän kuluessa.

- *Missä muodossa?*

Syksyllä gradu loppuun ja kun se on mennyt tarkastuksesta läpi, se lähetetään kaikille haastatteluun osallistuneille pdf-muodossa.

Käsiteltävät käytänteet

Scrum

- Tuotteen tehtävälista / Product Backlog
- Sprintin tehtävälista / Sprint Backlog
- Sprintti / Sprint
- Sprintin suunnittelukokous / Sprint Planning Meeting
- Päivittäinen Scrum-tapaaminen / Daily Scrum
- Sprintin katselmointitapaaminen / Sprint Review meeting
- Sprintin arviointitapaaminen / Sprint Retrospective

XP

- Pariohjelmointi / Pair programming
- Koodin yhteinen omistajuus /Collective code ownership
- Testivetoinen kehitys / Test-driven development
- Jatkuva integraatio / Continuous integration
- Jatkuva asiakkaan läsnäolo / On-site customer

Käytänteen omaksumisen taso yrityksessä

Taso	Tasolle siirtymisen kriteeri
1. Tietoisuus (Awareness)	Avainpäättäjät ovat tietoisia käytänteestä X
2. Kiinnostus (Interest)	Organisaatio on sitoutunut hankkimaan lisää tietoa käytänteestä X
3. Arviointi / kokeilu (Evaluation / trial)	Organisaatio on aloittanut käytänteen arvioinnin tai kokeilun
4. Sitoutuminen (Commitment)	Organisaatio on sitoutunut käyttämään käytännettä X yhdessä tai useammassa projektissa
5. Rajoitettu käyttö (Limited deployment)	Organisaatio on ottanut ohjelmaansa käytänteen X säännöllisen, mutta silti rajoitetun käytön
6. Yleinen käyttö (General deployment)	Organisaatio on saavuttanut tason, jossa käytännettä X käytetään uudessa kehityksessä merkittävässä osassa.

Kategoria	Kriteeri
7. Hylkääminen (Rejection)	Organisaatio on arvioinut ja hylännyt käytänteen X nykyisessä tilanteessa
8. Epäjatkuvuus (Discontinuance)	Organisaatio oli sitoutunut käyttämään käytännettä X, mutta ei käytä sitä tällä hetkellä, eikä usko käyttävänsä sitä tulevaisuudessakaan

Hyödyllisyysasteikko

-3	-2	-1	0	1	2	3
Käytänteestä on selvästi haittaa projektille	Käytänne haittaa projektia	Käytänne vaikeuttaa hieman projektia	Käytänteestä ei ole ollut hyötyä eikä haittaa projektille	Käytänteestä hieman hyötyä projektille	Käytänne on hyödyllinen projektille	Käytänne erittäin hyödyllinen

Käytänteiden määritelmät lyhyesti

Scrum

- Tuotteen tehtävälista / Product Backlog
    - Priorisoitu lista projektin vaatimuksista sekä niiden aika-arvioista, jotka muutetaan valmiiksi tuotteen toiminnallisuudeksi.
    - Aika-arviot ovat päivän tasolla ja ne tarkentuvat sitä myötä, mitä korkeammalla ne ovat tuotteen tehtävälistan prioriteettiasteikolla.
    - Lista kehittyy muuttuvien liiketoiminta- ja teknologiamuutosten mukaan.
    - Listassa on mukana toiminnallisia ja ei-toiminnallisia vaatimuksia sekä ongelmia (issues)
- Sprintin tehtävälista / Sprint Backlog
    - Lista tehtävistä, jotka määrittävät tiimin työtehtävät seuraavan sprintin aikana.
    - Lista kehittyy sprintin aikana.
    - Jokaisen tehtävän (task) yhteydessä mainitaan, kuka on vastuussa tehtävän tekemisestä ja mikä on jäljellä olevan tehtävän aika-arvio.
    - Aika-arviota päivitetään joka päivä.
- Sprintti / Sprint
    - Rajoitettu 30 päivään, jonka aikana tiimi työstää valitsemistaan tuotteen tehtävälistan tehtävistä tuotteen toiminnallisuuksia
- Sprintin suunnittelukokous / Sprint Planning Meeting
    - Yhden päivän mittainen tapaaminen, joka kestää 8 tuntia jokaisen sprintin alussa
    - Tapaaminen on jaettu kahteen 4 tunnin mittaiseen osaan
    - Ensimmäisessä osassa tuotteen omistaja esittelee tärkeimmät tuotteen tehtävälistan tehtävät tiimille, jonka jälkeen he miettivät yhdessä, kuinka monta tehtävää seuraavan sprintin aikana pystytään toteuttamaan. Tiimi sitoutuu tapaamisen päätteeksi tekemään sovitut tehtävät.
    - Suunnittelukokouksen toisessa osassa tiimi suunnittelee, kuinka he toteuttavat sovitut tehtävänsä määrittelemällä ne tarkemmalla tasolla sprintin tehtävälistaan
- Päivittäinen Scrum-tapaaminen / Daily Scrum
    - Rajoitettu 15 minuuttiin
    - Lyhyt tiimin tapaaminen tilanteen selvittämiseksi, jossa tiimin jäsenet synkronoivat tehtävänsä ja edistymisensä sekä raportoivat mahdolliset esteensä Scrum-mestarille, jotta niihin voidaan puuttua.
- Sprintin katselmointitapaaminen / Sprint Review meeting
    - Rajoitettu 4 tuntiin
    - Jokaisen Sprintin lopussa tiimi esittelee tuotteen omistajalle sekä muille kiinnostuneille tahoille, mitä se on saanut aikaiseksi kuluneen sprintin aikana.
    - Vain valmiita toiminnallisuuksia voidaan demonstroida.
- Sprintin arviointitapaaminen / Sprint Retrospective
- Rajoitettu 3 tuntiin
    - Scrum-mestarin johtama tilaisuus, jossa tiimi keskustelee juuri päättyneestä sprintistä ja määrittelee, mitä voitaisiin muuttaa, jotta seuraava sprintti olisi miellyttävämpi tai tehokkaampi

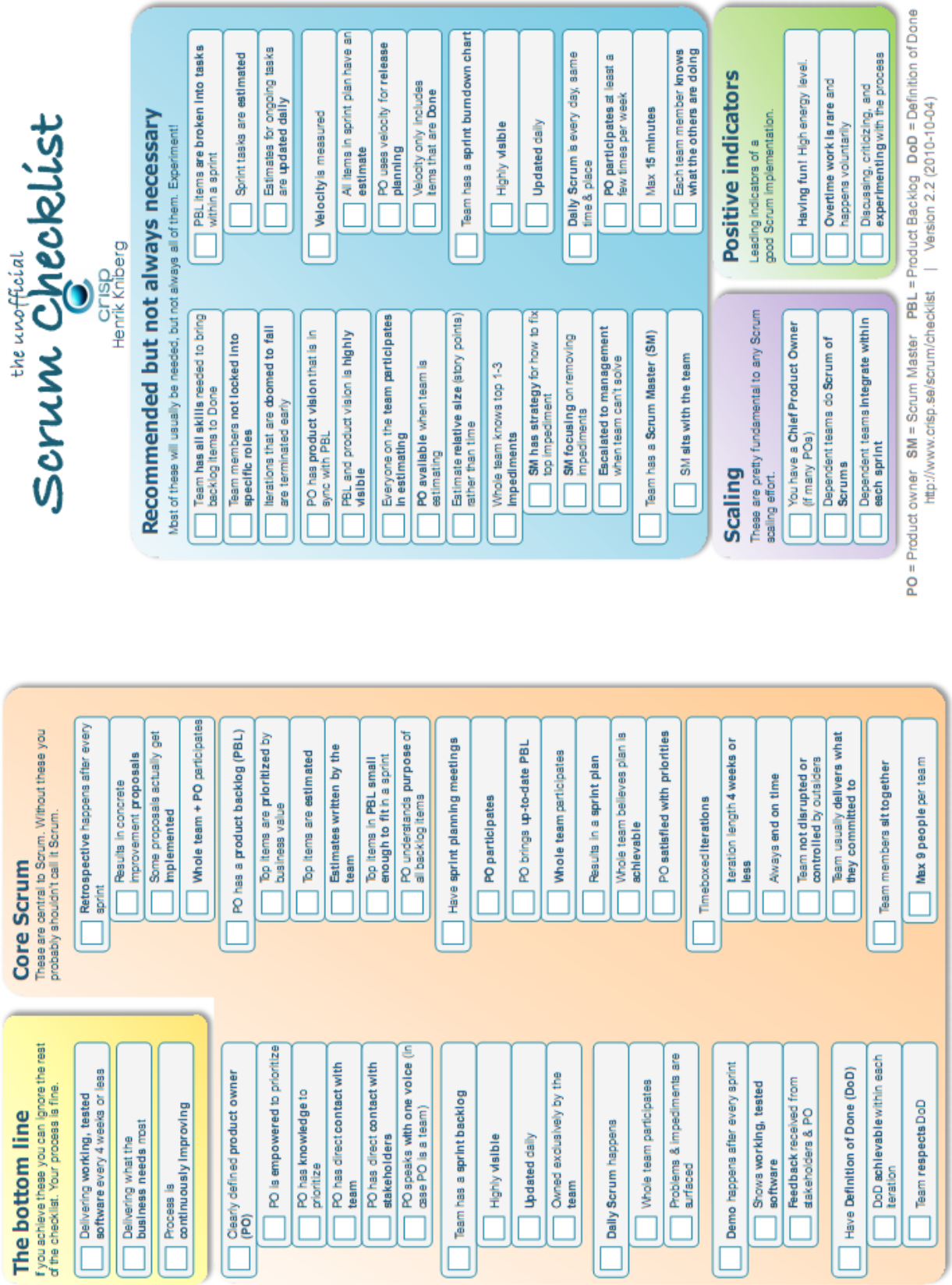
XP

- Pariohjelmointi / Pair programming
  - ”Kaikki tuotettu koodi on kirjoitettu kahden ohjelmoijan istuessa yhdellä koneella”
  - Koodin yhteinen omistajuus / Collective code ownership
  - ”Kuka vain voi muuttaa mitä tahansa koodin osaa järjestelmässä milloin vain”
- Testivetoinen kehitys / Test-driven development
  - Kehittäjät kirjoittavat koko ajan yksikkötestejä, jotka pitää läpäistä virheettömästi, jotta kehitystyö voi jatkua.
  - Asiakkaat kirjoittavat testejä, joidenka läpäisseitä toiminnallisuuksia voidaan pitää valmiina.
- Jatkuva integraatio / Continuous integration
  - ”Integroi ja rakenna järjestelmä monta kertaa päivässä, aina kun jokin tehtävä on saatu valmiiksi”
- Jatkuva asiakkaan läsnäolo / On-site customer
  - Ota tiimiin mukaan asiakas, joka voi vastata kysymyksiin milloin vain on tarvetta
  - Beck määrittelee asiakkaan sellaiseksi, joka todella käyttää kehitettävää tuotetta sen valmistumisen jälkeen.

Scrumin käytänteitä koskevat määritelmät: Schwaber, 2004, Agile Project Management with Scrum, Appendix B
XP:n käytänteitä koskevat määritelmät: Beck, 1999, Extreme Programming Explained, kappale 10

LIITE 3 SCRUMIN TARKISTUSLISTA

Lähde: Kniberg,H (2010)



LIITE 4 TIIVISTELMÄ TULOKSISTA

LIITE 4 Tiivistelmä siitä, kuinka käytänteitä oli muokattu haastatelluissa yrityksissä sekä käytänteiden hyvät ja huonot puolet aiempien empiiristen tutkimusten mukaan (merkitty numerolla 1) ja haastattelututkimuksen mukaan (merkitty numerolla 2).

Käytänne	Käytänteen muokkaus	Hyvät puolet	Huonot puolet
Sprintti	Sprintti oli mitoitettu useimmiten 2 viikon mittaiseksi.	+ Tuo läpinäkyvyyttä prosessiin (1) + Tekemisellä selkeä päämäärä, tahti ja suuntaviivat (1, 2) + Säännöllinen palaute (2)	- Sprintin seuraaminen vaikeaa eri teknologioita yhdistävissä projekteissa (1) - Asiakkaalle joskus vaikeaa lukita kehitystä seuraavaksi 2 viikoksi (2)
Tuotteen tehtävälista	Tehtävien ajallista arviota ei tehty aina, eikä se välttämättä ollut tunneissa mitattuna.	+ Hyvä kommunikoinnin väline tiimin sisäisesti (1, 2) + Tehtävien parempi hahmottaminen (1, 2) + Tuo läpinäkyvyyttä tekemiseen asiakkaan suuntaan (2)	- Varottava jakamasta isoja kokonaisuuksia vain yhdelle tiimin jäsenelle (1) - Tehtävien liiallinen pilkkominen voi hukata idean isommasta kokonaisuudesta ja rajoittaa käytettäviä teknologioita (2)
Sprintin suunnittelukokous	Tapaamista oli ajallisesti lyhennetty, eikä sitä aina erotettu kahteen erilliseen osaan.	+ Auttaa tehtävän ymmärtämisessä ja tavoitteen saavuttamisessa (1) + Antaa selkeän rajauksen kehitettävälle ohjelmistolle (1) + Auttaa asiakkaan kanssa työskentelyssä (1, 2) + Suunnitelmallisuus ehdoton edellytys koko työskentelylle (2)	- Suunnittelupalaverien ajallinen venyminen (1) (-) Helppo tehdä huonosti (2) (-) Jotta tuottaisi tulosta, on kaikkien osallistujien todella valmistauduttava siihen (2)
Sprintin tehtävälista	Tehtäville ei yleensä määritelty vastuuhenkilöä. Aika-arvioita ei päivitetty joka päivä.	+ Tapa analysoida tekemistä (2) + Selkeyttää tiimin tekemistä (2) + Vähentää tiimin stressiä ja tuo rauhaa työskentelyyn (2) + Tiimin tehtävät aina samassa paikassa (2)	(-) Jos tehtävälistan tilannetta kaunistellaan asiakasta varten, aiheuttaa tämä ongelmia (1) - Tehtävien liiallinen pilkkominen tappaa kehittäjien oman luovuuden (2)

(jatkuu…)

LIITE 4: Tiivistelmä siitä, kuinka käytänteitä oli muokattu haastatelluissa yrityksissä sekä käytänteiden hyvät ja huonot puolet aiempien empiiristen tutkimusten mukaan (merkitty numerolla 1) ja haastattelututkimuksen mukaan (merkitty numerolla 2) (jatkuu)

Käytänne	**Käytänteen muokkaus**	**Hyvät puolet**	**Huonot puolet**
Päivittäinen Scrum-tapaaminen	Useimmiten tapaaminen toteutettiin määritelmän mukaisesti.	+ Parantaa yhteenkuuluvuuden tunnetta (1) + Tietoisuus muiden edistymisestä (1, 2) + Parempi yleiskuva projektista (1, 2) + Kommunikointi parantui (1) +Virheet havaittiin ja korjattiin nopeammin (1) + Tuo läpinäkyvyyttä eri tiimien ja toimipaikkojen välille (1) + Paljasti ongelmat ajoissa (1, 2) + Auttaa tiimiä pääsemään yhteiseen päämäärään (2) + Toimii kannustimena tekemiselle (2)	- Tapaamisten venähtäminen (1) - Joissakin kulttuureissa ongelmista kertominen voi olla vaikeaa (1, 2) - Jos tiimi työskentelee muutenkin tiiviisti ja samassa tilassa, ei tapaamisella ole niin suurta merkitystä tekemiseen (2)
Sprintin katselmointitapaaminen	Ei merkittäviä eroja määritelmään.	+ Auttaa hahmottamaan projektia paremmin (1) +Auttaa hahmottamaan, toimitettiinko sitä, mitä luvattiin (1) + Luo asiakkaaseen luottamusta tiimin taitoihin (1) + Hyvä tapa levittää tietoa muille tiimin jäsenille (1) + Tiimi saa arvokasta palautetta työstään (2) + Selkeä ja konkreettinen takaraja projektille (2)	- Tapaamisen ajallinen venyminen (1) - Vie paljon aikaa (2) - Asiakkaan saaminen paikan päälle on joskus haastavaa (2)

(jatkuu…)

LIITE 4: Tiivistelmä siitä, kuinka käytänteitä oli muokattu haastatelluissa yrityksissä sekä käytänteiden hyvät ja huonot puolet aiempien empiiristen tutkimusten mukaan (merkitty numerolla 1) ja haastattelututkimuksen mukaan (merkitty numerolla 2) (jatkuu)

Käytänne	**Käytänteen muokkaus**	**Hyvät puolet**	**Huonot puolet**
Sprintin arviointitapaaminen	Tapaamisen kesto vaihteli paljon, mutta määritelmän rajoissa.	+ Ohjelmoijat saivat palautetta työstään (1) + Pakottaa tiimiä miettimään todella, että tehtiinkö sitä, mitä luvattiin (1) + Hyvä työkalu prosessin kehittämiseen (2) + Projektipäällikkö saa suoraa palautetta siitä, mikä toimii ja mikä ei (2)	- Jos asioihin ei reagoida tarpeeksi nopeasti tapaamisen jälkeen, tapaamisen merkitys vähenee (1, 2) - Tietoa vaikea levittää eri tiimien välillä (1)
Pariohjelmointi	Pariohjelmointia ei hyödynnetty missään yrityksessä puhtaasti, pariohjelmointi otettiin käyttöön vain tilanteen niin vaatiessa.	+ Kehittää koodin laatua (1) + Parantaa tiedonsiirtoa projektissa (1) + Vähentää koodin virheitä (1, 2) + Voi parantaa koodin suunnittelua (1, 2) + Toimii perehdyttämis-menetelmänä (2)	- Vie resursseja (1) - Ei sovi kaikille (1, 2) - Vaatii enemmän keskittymistä (1, 2)
Koodin yhteinen omistajuus	Ei juurikaan muutoksia määritelmään nähden	+ Koodin laatu paranee (1) + Tiimillä parempi käsitys kehitettävästä ohjelmistosta (1, 2) + Koodin luettavuus ja selkeys parantuu (2) + Koodi on luettavampaa paremman kommentoinnin ansiosta (2)	- Ei sovi välttämättä isoille tiimeille (1) - Ei sovi, jos tiimin jäsenten osaamisalueet ovat hyvin erilaisia (1) - Voi aiheuttaa ongelmia, jos muutoksia tehdään ilman, että tiedetään minne kaikkialle ne vaikuttavat (2)

(jatkuu…)

LIITE 4: Tiivistelmä siitä, kuinka käytänteitä oli muokattu haastatelluissa yrityksissä sekä käytänteiden hyvät ja huonot puolet aiempien empiiristen tutkimusten mukaan (merkitty numerolla 1) ja haastattelututkimuksen mukaan (merkitty numerolla 2) (jatkuu)

Käytänne	**Käytänteen muokkaus**	**Hyvät puolet**	**Huonot puolet**
Testivetoinen kehitys	Asiakas ei yleensä osallistunut testien kirjoittamiseen.	+ Koodi on tasaisempaa ja laadukkaampaa (1) + Vähentää virheitä (1) + Nopeuttaa testausprosessia (1) + Nostaa testikattavuutta ja lisää yksikkötestauksen määrää (1) + Helpottaa koodin ylläpitoa (2) + Auttaa koodin ymmärtämisessä (2) + Nopeuttaa virheiden löytämistä (2)	- Haasteellinen kokemattomille tiimeille, koska vaatii uuden ajattelutavan (1) - Koettiin takaperoisena käytänteenä (1) - Vaativaa ja siksi hidasta (2) - Vaikea perustella asiakkaalle, koska ei näennäisesti tuota asiakkaalle mitään (2)
Jatkuva integraatio	Integrointia ei välttämättä tehty aina montaa kertaa päivässä.	+ Julkaisut ovat asiakkaan nähtävillä nopeammalla aikataululla (1) + Virheet havaittiin aikaisemmin (1) + Ohjelmistoprojektien hallittavuus oli parempaa (1) + Lisää luottamusta tehtyyn ohjelmistoon (2) + Testit tulee ajettua tämän ansiosta vähintään kerran päivässä (2) + Pitää kehittäjät ajan tasalla siitä, missä mennään ja toimiiko kaikki odotusten mukaisesti (2)	- Vaikea toteuttaa täysin laajojen ohjelmistojen kanssa (2)

(jatkuu…)

LIITE 4: Tiivistelmä siitä, kuinka käytänteitä oli muokattu haastatelluissa yrityksissä sekä käytänteiden hyvät ja huonot puolet aiempien empiiristen tutkimusten mukaan (merkitty numerolla 1) ja haastattelututkimuksen mukaan (merkitty numerolla 2) (jatkuu)

Käytänne	**Käytänteen muokkaus**	**Hyvät puolet**	**Huonot puolet**
Jatkuva asiakkaan läsnäolo	Asiakas ei yleensä ollut fyysisesti läsnä samassa tilassa tiimin kanssa.	+ Asiakkaalta sai arvokasta palautetta päivittäin (1) + Säästää aikaa kun asiakkaan kommentteja ei tarvitse odotella (2) + Auttaa erityisesti jos asiakkaan toimialan tuntemus on tärkeää (2)	- Asiakkaan rooli on haastava (1) - Jos asiakas todella läsnä jatkuvasti, voi se olla tiimille ennemminkin taakka kuin apu (2) - Asiakas ei voi olla samassa tilassa tiimin kanssa, jos tiimillä on samanaikaisesti muita keskeneräisiä projekteja (2)