TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANI KUITTI

# SIIRTYMINEN PERINTEISISTÄ OHJELMISTOTUOTANTOMENETELMISTÄ KETTERIIN MENETELMIIN

Kandidaatintyö

Tarkastaja: Pia Niemelä
Jätetty tarkastettavaksi
17.12.2017

# TIIVISTELMÄ

**Jani Kuitti**: Siirtyminen perinteisistä ohjelmistotuotantomenetelmistä ketteriin menetelmiin
Transition from traditional to agile software processes
Tampereen teknillinen yliopisto
Kandidaatintyö, 19 sivua
Joulukuu 2017
Tekniikan kandidaatin tutkinto-ohjelma
Pääaine: Ohjelmistotekniikka
Tarkastaja: Pia Niemelä

Avainsanat: ohjelmistotuotanto, ketterät menetelmät, vaihejako-menetelmät

Tässä työssä tutkitaan siirtymistä perinteisistä vaihejakoisista ohjelmistotuotantomenetelmistä ketteriin menetelmiin. Tutkimuksella pyritään selvittämään, millaisia muutoksia havaitaan menetelmää vaihdettaessa ja oliko menetelmän vaihtaminen lopulta kannattavaa. Tutkimus toteutettiin etsimällä tietokannoista aiheeseen liittyviä tutkimuksia ja analysoimalla niistä saatuja tuloksia. Tutkimuksista havaitut muutokset olivat pääosin hyviä ja keskittyivät tuotantotiimin sisäiseen ja asiakkaan väliseen kommunikointiin, ohjelmiston laatuun ja projektin näkyvyyteen. Työssä mukana olleiden tutkimusten tulosten perusteella voitiin päätellä, että siirtyminen käyttämään ketterää menetelmää on kannattavaa.

# SISÄLLYSLUETTELO

# 1. JOHDANTO

Nykypäivän ohjelmistotuotannossa on käytössä monia erilaisia menetelmiä, joissa toimintatavat voivat olla hyvin erilaisia. Uusimpina menetelminä on tullut mukaan ketterät menetelmät, joissa pyritään paremmin huomioimaan asiakas ja vastaamaan paremmin projektiin kohdistuviin muutoksiin. Onko siis syytä vaihtaa toimiva vanha menetelmä tällaiseen uuteen menetelmään ja millaisia hyötyjä sekä riskejä se tuo mukanaan? Tässä työssä tutkimusongelmana on siirtyminen perinteisistä vaihejakoisista ohjelmistotuotantomenetelmistä ketteriin menetelmiin. Työssä pyritään vastaamaan kysymyksiin: Millaisia muutoksia tapahtuu eri osa-alueilla ja oliko siirtyminen ketterään menetelmään kannattavaa.

Tekstin toisessa luvussa käydään lyhyesti läpi jokainen ohjelmistotuotantomenetelmä, joka on mukana tässä työssä. Kolmannessa luvussa käydään läpi tämän työn tekemiseen vaadittujen tutkimusten hakuprosessi. Siinä kerrotaan, mistä tietokannoista tutkimuksia haettiin, mitä hakulausekkeita käytettiin ja miten tulokset lopulta rajattiin. Neljännessä luvussa käydään läpi itse tutkimukset. Jokainen tutkimus käydään läpi ja niistä kerrotaan saadut tulokset. Viidennessä luvussa tehdään yhteenveto kaikkien edellisen luvun tutkimusten tuloksista ja analysoidaan niitä. Kuudennessa luvussa käydään läpi koko työn yhteenveto ja mietitään mahdollisuuksia aiheen tutkimiselle jatkossa.

# 2. OHJELMISTOTUOTANTOMENETELMÄT

Tässä luvussa esitellään lyhyesti kaikki tutkimuksissa mukana olleet ohjelmistotuotantomenetelmät. Tässä työssä käsiteltävät menetelmät voidaan jakaa kahteen eri ryhmään: perinteiset vaihejako-menetelmät sekä ketterät menetelmät.

## 2.1 Vaihejako-menetelmät

Ohjelmistotuotannossa vaihejako-menetelmät koostuvat tarkkaan määritellyistä vaiheista. Näihin voi kuulua esimerkiksi: asiakasvaatimusten analysointi, ohjelman rakenteen suunnittelu, ohjelmakoodin kirjoitus ja testaaminen. Vaihejako-menetelmissä eri vaiheiden suoritusjärjestys on tarkkaan määritelty ja ennen kuin voidaan siirtyä seuraavaan vaiheeseen, täytyy nykyisen vaiheen olla valmis. Näissä menetelmissä myöskin työn dokumentaatio jokaisessa välivaiheessa on erittäin tärkeää.[1]

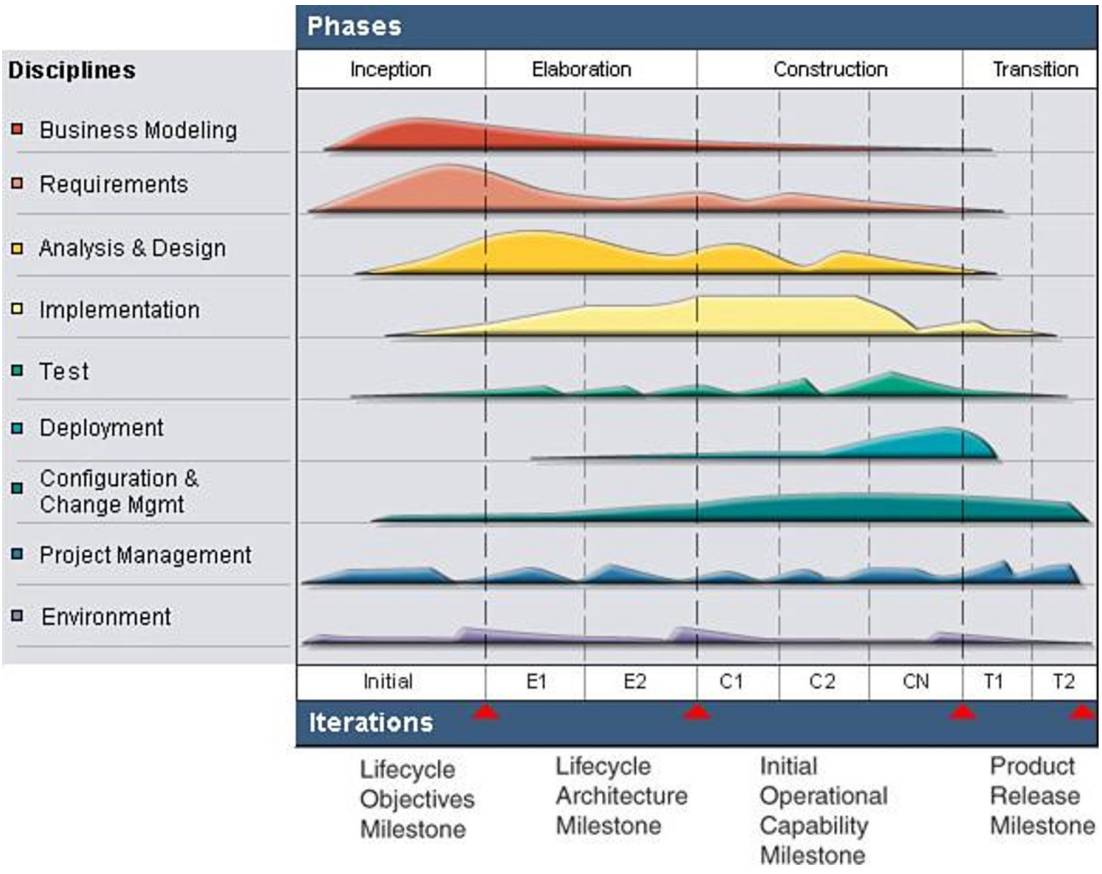
### 2.1.1 Rational Unified Process -menetelmä

Rational Unified Process (RUP) -menetelmä on liiketoimintalähtöinen ja siinä pyritään aikaisessa vaiheessa huomioimaan kaikki projektin mahdolliset riskitekijät ja ennaltaehkäisemään niitä. RUP:in pääpiirteisiin kuuluu muun muassa sopeuttaa prosessi kuhunkin projektiin, tehdä yhteistyötä eri tiimien välillä, sekä keskittyä pysähtymättä kehitteillä olevan ohjelmiston laatuun.[2]

RUP koostuu neljästä vaiheesta: alkuvaihe, suunnitteluvaihe, rakennusvaihe ja siirtymävaihe. Ensimmäisessä vaiheessa eli alkuvaiheessa määritellään projektin kustannukset sekä aikataulu. Samalla määritellään kriittiset käyttötapaukset sekä käydään läpi mahdolliset riskitekijät. Toisessa vaiheessa eli suunnitteluvaiheessa määritellään projektin rakenne, vaatimukset ja tehdään molemmista suunnitelmat koko projektin ajalle. Lisäksi tässä vaiheessa pyritään ottamaan huomioon kaikki projektin rakenteeseen liittyvät riskit.[2]

Kolmannessa vaiheessa eli rakennusvaiheessa luodaan ohjelmakoodia edellisten vaiheiden suunnitelmien perusteella. Koodista pyritään tekemään riittävän hyvää mahdollisimman nopeasti. Tässä vaiheessa myöskin suoritetaan ohjelmistolle vaadittavat testit ja julkaistaan eri kehitysversioita ohjelmasta. Neljännessä vaiheessa eli siirtymävaiheessa varmistetaan, että ohjelmasta tuli asiakkaan toiveiden mukainen ja asiakas koulutetaan ohjelman käyttöön. Lisäksi, jos ohjelmassa havaitaan jotain virheitä, ne korjataan tässä vaiheessa.[2]

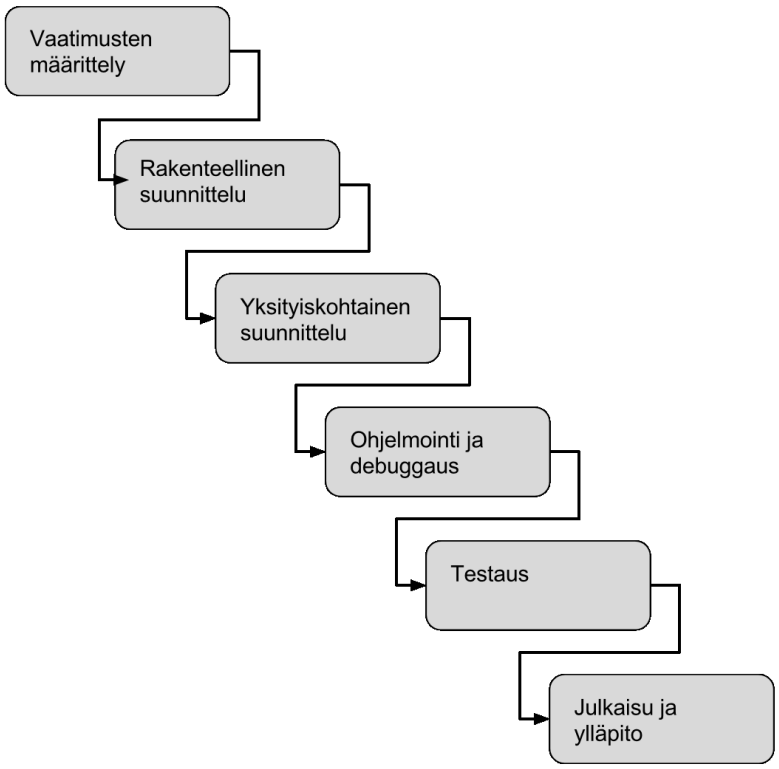
Kuvassa 1 esitetään edellisessä kappaleessa mainitut työvaiheet. Tämän lisäksi kuvasta havaitaan, mihin vaiheisiin erilaiset työt kuuluvat missäkin määrin.[2]




Kuva 1: RUP:in eri osa-alueiden työmäärät eri vaiheissa[2].

### 2.1.2 Vesiputous-menetelmä

Ohjelmistotuotannon vesiputous-menetelmä kehitettiin vuonna 1970, ja se oli ensimmäinen kehitetty vaihejako-menetelmä. Kuvassa 2 esitellään vesiputous-menetelmään kuuluvat vaiheet ja niiden suoritusjärjestys.[1]




Kuva 2: Vesiputous-menetelmän vaiheet[1].

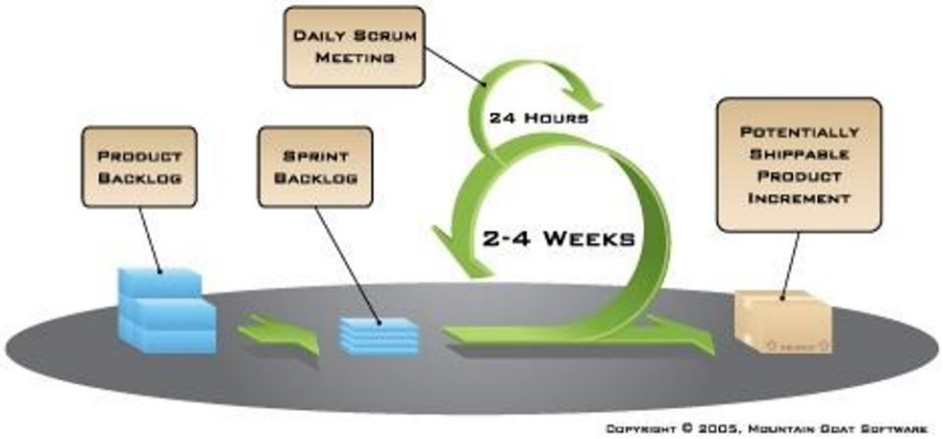
Jotta nykyisestä vaiheesta voitaisiin siirtyä seuraavaan vaiheeseen, täytyy nykyisen vaiheen olla täysin valmiina, ja kuten kuvasta 2 havaitaan, edelliseen vaiheeseen ei voida palata takaisin sen valmistuttua. Tämä menetelmä toimii hyvin, jos projektiin ei tule mitään muutoksia koko sen tuotantoaikana. Tämä on kuitenkin hyvin harvinaista, joten tämän menetelmän heikkona puolena on projektiin kohdistuvat muutokset. Jos esimerkiksi ohjelmointivaiheessa tulee jokin muutos asiakasvaatimuksiin, sitä voi olla hyvin vaikea lähteä toteuttamaan. Tällöin ohjelman rakenne on jo suunniteltu edellisten vaatimusten mukaan suunnitteluvaiheissa eikä sitä voida enää muokata.[1]

## 2.2 Ketterät menetelmät

Ohjelmistotuotannon ketterien menetelmien arvot ja periaatteet kehitettiin vuonna 2001 Yhdysvalloissa. Tällöin joukko ohjelmistotuotannon ammattilaisia hahmotteli yhdessä Agile Manifeston. Tämä sisälsi neljä arvoa ja 12 periaatetta, joita ohjelmistotuotannossa tulisi käyttää.[3] Ketterien menetelmien avulla voidaan helposti vastata muuttuviin asiakasvaatimuksiin ja ohjelma saadaan nopeammin asiakkaalle. Kyky vastata muutoksiin perustuu siihen, että jokainen työvaihe toistetaan monta kertaa ohjelman tuotannon aikana. Tässä projektitiimit ovat yleensä huomattavasti pienempiä kuin vastaavat tiimit vaihejako-menetelmissä.[1]

### 2.2.1 Scrum-menetelmä

Scrum-menetelmä koostuu sprinteistä. Sprintillä tarkoitetaan tavallisesti yhdestä neljään viikkoa kestävää jaksoa, jolloin ohjelmaan toteutetaan sprintin alussa valitut ominaisuudet. Jokaisen sprintin lopussa ohjelmasta on valmiina julkaisukelpoinen versio, joka voidaan esitellä asiakkaalle. Scrumissa on käytössä kaksi backlogia eli listaa erilaisista käyttäjätarinoista ja tehtävistä, joita tuotettavaan ohjelmaan halutaan sisällyttää. Näistä ensimmäinen on product backlog, joka sisältää kaikki projektin tehtävät, joita ei vielä ole ohjelmaan toteutettu. Toisena backlogina on sprint backlog. Tämä sisältää kaikki tehtävät, jotka on valittu toteutettavaksi nykyiseen sprinttiin.[1] Kuvassa 3 on esitetty molemmat backlogit ja yhden sprintin aikataulu[4].




Kuva 3: Scrumin prosessi[4].

Scrumiin kuuluu hyvin tärkeänä osana Scrum-tiimin päivittäiset tapaamiset, kuten kuvasta 3 voidaan havaita. Nämä tapaamiset ovat yleensä kestoltaan noin 15-30 minuuttia ja niihin osallistuu koko Scrum-tiimi. Tapaamisissa käydään läpi, miten projekti onnistuu, onko havaittu ongelmia ja jos on, niin keskustellaan, miten ne voitaisiin korjata.[1]

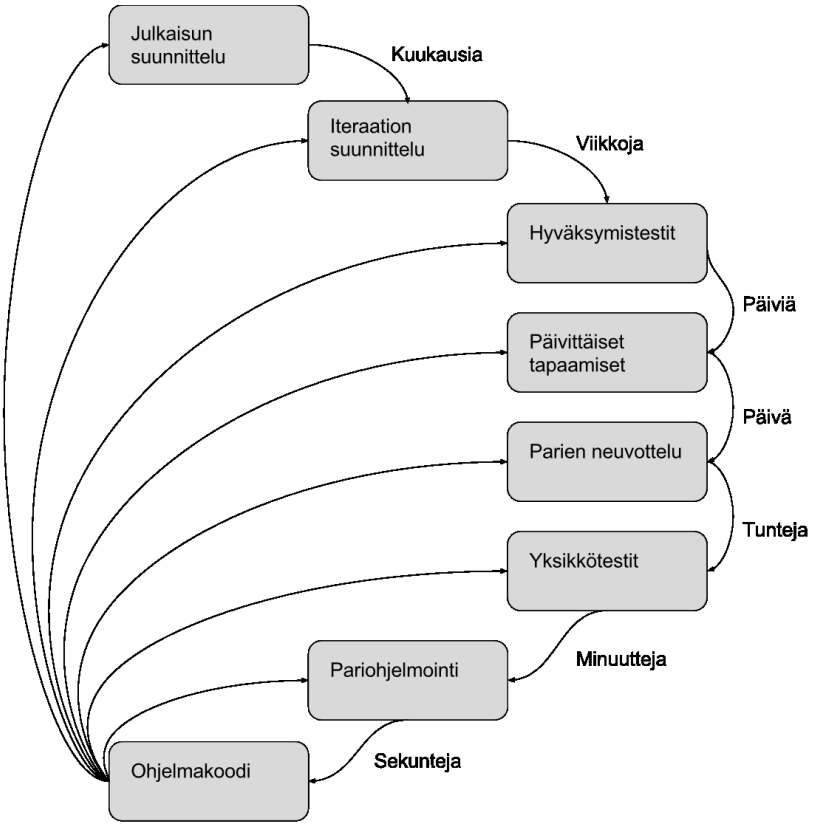
Scrum-tiimi koostuu ScrumMasterista, tuotteen omistajasta ja tuotantotiimistä. Tiimin koko on yleensä korkeintaan kymmenen henkilöä. ScrumMasterin tehtäviin kuuluu järjestää päivittäiset tapaamiset tiimin kesken, suojata tiimiä ulkopuolisilta vaikutteilta, sekä huolehtia projektin backlogeista. Tuotteen omistajan tehtäviin kuuluu huolehtia asiakkaan kanssa projektin tavoitteista ja kerätä ne product backlogiin. Lisäksi tuotteen omistajan tehtäviin kuuluu suunnitella sprint- ja julkaisutavoitteet muiden tiimiläisten kanssa. Tuotantotiimin tehtäviin kuuluu luoda product backlogin sisällöstä tehtäviä, joita he voivat toteuttaa projektiin. Tuotantotiimin tehtäviin kuuluu myös itse huolehtia siitä, että projekti etenee suunnitellun mukaisesti.[3]

### 2.2.2 Extreme Programming -menetelmä

Extreme Programming (XP) kehitettiin vuonna 1995. Tässä menetelmässä on erittäin tärkeässä osassa asiakkaan läsnäolo ohjelmiston kehityksessä. Tällöin mukana on asiakkaan edustaja, ja hän tekee esimerkiksi hyväksymistestit aina ennen ohjelman julkaisua. Yksikkötestit ovat myöskin hyvin tärkeässä osassa XP:ssä. Ohjelmakoodin osion kirjoitus aloitetaan aina luomalla sille osiolle omat yksikkötestit. Osiota voidaan pitää valmiina heti kun kaikki yksikkötestit menevät onnistuneesti läpi.[1]

XP:ssä on myöskin käytössä pariohjelmonti, jossa saman tietokoneen ääressä työskentelee kaksi henkilöä. Tällöin toinen heistä ohjelmoi ja toinen tarkkailee vieressä. Tarkkailijan tehtäviin kuuluu etsiä virheitä koodista, kommentoida ja esittää kysymyksiä tuotetusta ohjelmakoodista. Tätä tehdään esimerkiksi puoli tuntia kerrallaan ja sen jälkeen henkilöt vaihtavat osia. Kun parin tuotos on valmis, se liitetään heti osaksi ohjelmaa. Lisäksi tehdään vaadittavat testit, jotta havaitaan, ettei lisätty koodi riko mitään ohjelmassa.[1]

Kuvassa 4 käydään läpi XP:n suunnittelun ja palautteen kulkua. Kuvassa nähdään, kuinka pitkiksi ajoiksi kerrallaan tehdään suunnitelmia missäkin vaiheissa ja kuinka nopeasti mistäkin vaiheesta saadaan palautetta.[5]




Kuva 4: Extreme Programmingin suunnittelun ja palautteen kulku[5].

# 3. TUTKIMUSTEN HAKUPROSESSI

Tämän työn puitteissa ei ollut mahdollisuutta suorittaa omaa tutkimusta aihepiiriin liittyen, joten erilaisia muutoksia analysoidaan toisten tekemien tutkimusten tuloksista. Seuraavissa alaluvuissa käydään läpi tämän työn tiedonhakuprosessi.

## 3.1 Tietokannat

Tiedonhaussa käytettiin Tampereen teknillisen yliopiston (TTY) kirjaston Andor-tietokantaa sekä Institute of Electrical and Electronics Engineersin IEEE Xplore -tietokantaa. Andor sisältää kaikki TTY:n kirjaston saatavilla olevat kirjat, e-kirjat, artikkelit ja lehdet[6]. IEEE Xplore sisältää miljoonia kirjallisia julkaisuja tietotekniikan, sähkötekniikan ja elektroniikan aloilta[7].

## 3.2 Hakuavaimet

Hakulausekkeina käytettiin hakuavaimen ”software development” ja eri menetelmien nimistä koostuvaa yhdistelmää. Tästä esimerkkinä: ”software development” AND Scrum. Tällaisilla hakulauseilla löydettiin käytetyistä tietokannoista kaikki tässä työssä mukana olevat tutkimukset. Jokaisella hakulausekkeella tuli suuri määrä hakutuloksia, mutta tutkimukset löytyivät kuitenkin hakutulosten alkupäästä. Hakulausekkeisiin yritettiin myöskin lisätä joitain tutkimukseen viittaavia avainsanoja, kuten ”transition” tai ”study”. Tämä ei kuitenkaan lisännyt aiheeseen sopivien hakutulosten määrää, joten ne jätettiin hakulausekkeista pois kokonaan.

## 3.3 Hakutulosten rajaaminen

Hakutuloksia lähdettiin rajaamaan hakutilanteessa tekstin otsikon ja tiivistelmän avulla. Jos näiden perusteella teksti vaikutti sopivan tähän työhön, luettiin teksti kokonaisuudessaan läpi. Tekstistä etsittiin sekä kvalitatiivista että kvantitatiivista tutkimustietoa, jota voisi käyttää tämän työn tekemiseen. Tämän jälkeen teksti valittiin mukaan tähän työhön, jos se sisälsi tähän sopivaa tutkimusmateriaalia. Tästä aiheesta tehtyjä tutkimuksia ei löytynyt haussa käytetyistä tietokannoista kovinkaan paljoa, joten rajauksia ei tehty sen mukaan, mistä ohjelmistotuotantomenetelmästä siirryttiin mihinkin.

# 4. VALITTUJEN TUTKIMUSTEN REFEROINTI

Tässä luvussa käydään läpi tiedonhaussa valitut tutkimukset. Tutkimuksissa 1 ja 3 kerrotaan ohjelmistoyrityksen siirtymisestä käyttämään ketterää ohjelmistotuotantomenetelmää perinteisen vaihejako menetelmän sijaan, sekä kerrotaan millaisia hyviä ja huonoja muutoksia se aiheutti. Tutkimuksessa 2 keskitytään vertailemaan erilaisten ohjelmistotuotantomenetelmien vaikutusta erilaisten ohjelmistoprojektien onnistumiseen. Tutkimuksessa 4 tutkitaan, miten ketterien menetelmien tapoja voitaisiin hyödyntää ilmailuun liittyvässä turvallisuuskriittisessä ohjelmoinnissa ja tutkimuksessa 5 selvitettiin, miten hyvin ohjelmistotuotantomenetelmää vaihtaneet yritykset ovat ottaneet uuden menetelmän tavat käyttöön.

## 4.1 Tutkimus 1

Tutkimuksessa 1 käsitellään, millaisia vaikutuksia löydetään, kun yksi ohjelmistoalan yritys siirtyy Rational Unified Process (RUP) -vaihejako-menetelmästä ketterään Scrum-menetelmään. Tutkimukseen osallistui kaksi Alankomaalaista yritystä. Yritysten nimien käytölle tekstissä ei oltu annettu lupaa, joten tuottajayritystä kutsutaan yritykseksi A ja asiakasyritystä yritykseksi B. Yritys A on ohjelmistoalan konsultaatioyritys, jolla on myös ulkomaalainen toimisto Intiassa. Yritys B on rahoitusalan yritys. [8]

Tutkimuksessa oli mukana kaksi yrityksen A ohjelmistoprojektia. Tutkimuksessa kerättiin vain kvalitatiivista dataa suorittamalla haastatteluita. Haastattelut kestivät noin tunnin kerrallaan ja niihin osallistui koko Scrum-tiimi. Haastatteluissa kysymykset olivat valmiiksi laadittuja, mutta myös keskusteluille oli varattu aikaa. [8]

Yrityksellä A oli alun perin käytössä tuotannossa RUP-menetelmä ja tutkimuksen alkaessa, he vaihtoivat käyttöön Scrum-menetelmän. Yrityksen A tuotanto tapahtui pääasiassa Intiassa sijaitsevalla toimistolla. Ohjelman arkkitehtuuri ja asiakasvaatimukset hallinnoitiin Alankomaiden toimistolla. [8]

Tutkimustulokset jaettiin viiteen eri osa-alueeseen. Ensimmäisenä osa-alueena oli asiakasvaatimusten luonti ja asiakkaan mukaan ottaminen ohjelmiston kehityksessä. Aiemmin kaikki asiakasvaatimukset hoidettiin yrityksen A Alankomaiden toimiston ja yrityksen B välillä. Tällöin asiakasvaatimukset menivät sellaisinaan Intian toimistolle, eivätkä sen työntekijät päässeet vaikuttamaan niihin. Scrumin myötä Intian toimiston työntekijät pystyivät olemaan suoraan yhteydessä yritykseen B, joten he pääsivät paremmin vaikuttamaan asiakasvaatimuksiin ja samalla väärinymmärrysten määrä väheni. Nyt myöskin yritys B pystyi olemaan mukana ohjelmiston tuottamisessa ja vaikuttamaan tuotantoon suoraa. [8]

Toisena osa-alueena oli kommunikaatio. Tällöin Scrumin myötä yrityksen A työntekijöiden tapaamiskerrat lisääntyivät ja töiden ajoittaminen Alankomaiden ja Intian toimistojen välillä helpottui. Molemmat yritykset siirtyivät käyttämään videokonferenssi-tapaamisia tavallisten sähköpostin ja pikaviestintäohjelmien sijaan. Tämä lisäsi luottamusta yrityksen A eri toimistojen välillä ja lisäksi se tutustutti Intian toimiston työntekijät yrityksen B työntekijöiden kanssa. [8]

Kolmantena osa-alueena oli hinnoittelu ja kulut. Ketterän mallin myötä yrityksen B vaatimat muutokset olivat helpompia toteuttaa ja tästä ei aiheutunut myöskään ylimääräisiä kuluja. Yrityksen A Alankomaiden toimiston kulut myös pienenivät, sillä Scrumin myötä toimistojen välinen kommunikaatio parani, joka vähensi töiden määrää Alankomaiden toimistolla. [8]

Neljäntenä osa-alueena oli tiimityöskentely. Scrumin myötä yrityksen A työntekijät pystyivät osallistumaan useisiin eri projekteihin samanaikaisesti. Lisäksi Intian toimiston työntekijöillä oli vapaus tehdä haluamiaan töitä, sillä heidän ei enää tarvinnut odottaa tarkkoja määräyksiä Alankomaiden toimistolta. [8]

Viidentenä ja viimeisenä osa-alueena oli toimitusaika ja tuotantotahti. Tuotantomenetelmän vaihtuessa ohjelman toimitusaika lyheni kuudesta kuukaudesta kahteen kuukauteen. Tämän seurauksena huonona puolena havaittiin, ettei enää jäänyt tarpeeksi aikaa työn kunnolliseen dokumentointiin. Samalla tuotantotahdin kasvaessa yrityksen B oli myöskin vaikea pysyä tahdissa mukana. Yrityksellä B oli oma laadunvalvontaosasto, jonka tuli aina tarkistaa ohjelman uusi versio sen saapuessa. Tähän saattoi kuitenkin kulua jopa 10 päivää, kun tulokset olisi pitänyt saada jo saman päivän aikana. [8]

## 4.2 Tutkimus 2

Estler et al. [9] kertovat tutkimuksessaan, millaisia eroja havaitaan eri ohjelmistoprojektien onnistumisissa, kun verrataan erilaisia ohjelmistotuotantomenetelmiä maailmanlaajuisessa ohjelmistotuotannossa. Tutkimukseen osallistuvissa projekteissa käytettiin seuraavia menetelmiä: RUP, vesiputous, Scrum ja XP.[9]

Tutkimukseen osallistui 31 eri yritystä Euroopasta, Amerikasta ja Aasiasta. Tutkimukseen kerättiin tietoa 66:sta eri projektista ja ulkoistuksen taso vaihteli jokaisessa projektissa suuresti. Tutkimus jaettiin kahteen eri vaiheeseen: kyselyihin ja haastatteluihin. Kyselyissä kerättiin kvantitatiivista tietoa ja haastatteluissa kvalitatiivista tietoa.[9]

Ensimmäisessä vaiheessa kyselyihin osallistui 48 projektia ja jokainen kysely kohdistui yhteen projektiin. Projekteissa 19:sta käytettiin vaihejako malleja ja 29:ssä ketteriä malleja. Kyselyissä tutkittiin eroja eri projektien välillä seuraavilla osa-alueilla: projektin onnistuminen, projektin tärkeys, tiimin motivaatio, rahalliset säästöt, vaadittu reaaliaikaisen kommunikaation määrä ja vaadittu ei-reaaliaikaisen kommunikaation määrä. Tulosten

analysointiin käytettiin Mann–Whitney–Wilcoxon U -testiä merkitsevyystasolla 0,05. Tällä analysoitiin kyselyiden tuloksia jokaisella osa-alueella ja päästiin lopulta tulokseen, ettei eri ohjelmistotuotantomenetelmillä ja työn ulkoistamisen tasolla ollut huomattavaa vaikutusta projektin onnistumisen kannalta.[9]

Toisessa vaiheessa haastatteluihin osallistui 13 Sveitsiläistä ohjelmistoyritystä. Haastattelut koostuivat samankaltaisista kysymyksistä kuin kyselyt tutkimuksen ensimmäisessä vaiheessa. Haastatteluiden tulokset jaettiin hieman erilaisiin osa-alueisiin kuin kyselyissä. Ensimmäisenä osa-alueena oli projektin onnistuminen. Tässä 11 projektia 19:sta onnistuivat tutkimuksen mukaan täysin. Projektin onnistumista edesauttoivat eniten pätevät työntekijät ja onnistunut tiimityöskentely, kun taas onnistumista vaikeuttivat eniten epäpätevät työntekijät, kulttuurilliset erot ja kommunikaatiovaikeudet. Toisena osa-alueena oli projektin kulut. Haastatteluissa ei havaittu muutoksia kuluihin riippumatta käytetystä ohjelmistotuotantomallista tai ulkoistamisen tasosta. Kolmantena osa-alueena oli projektin laatu. Laatu pysyi samana riippumatta käytetystä ohjelmistotuotantomallista tai ulkoistamisen tasosta.[9]

Haastattelujen neljäntenä osa-alueena oli henkilökohtaiset taidot. Tutkimuksen mukaan työntekijöiden henkilökohtaiset taidot olivat erittäin tärkeä osa projektin onnistumista. Lisäksi havaittiin, että työntekijöiden henkilökohtaiset taidot huononevat, mitä kauemmaksi projekti ulkoistetaan yrityksen kotimaasta. Viidentenä osa-alueena oli kommunikointi ryhmäläisten välillä. Tämän havaittiin myöskin olevan tärkeä osa projektin onnistumista ja 13 projektia 18:sta vaati viikoittaisia tapaamisia ryhmän jäsenten välillä. Kommunikointiin käytettiin eniten pikaviestejä ja vähiten puheluja ja paikanpäällisiä tapaamisia. Kuudentena osa-alueena oli projektitiimin koko. Tutkimuksessa havaittiin, että ketteriä menetelmiä noudattavissa projekteissa tiimin koko oli 30 henkilöä tai vähemmän, kun taas vaihejako menetelmiä noudattavissa projekteissa tiimin koko oli jopa 120 henkilöä. Viimeisenä osa-alueena oli projektin kriittiset ongelmat. Ketterissä menetelmissä havaittiin huomattavasti vähemmän kommunikaatio-ongelmia kuin perinteisissä menetelmissä. Lisäksi ongelmat projektin hallinnoinnissa olivat yleisempiä perinteisissä menetelmissä kuin ketterissä menetelmissä.[9]

## 4.3 Tutkimus 3

Tässä tutkimuksessa käsitellään siirtymää vesiputous-menetelmästä Scrum-menetelmään. Tutkimuksen kohteena on yksi maailman suurimmista vakuutus- ja rahoitusalan yrityksistä, Yhdysvalloissa perustettu Nationwide Insurance. Vuonna 2007 yrityksessä havaittiin, että silloin käytössä olleen vesiputous-menetelmän avulla ei kyetty vastaamaan projektin sidosryhmien asettamiin aikataulutavoitteisiin. Lisäksi haluttiin lyhentää ohjelmiston toimitusaikaa ja parantaa sen laatua. Siirtyminen ketterään Scrum-menetelmään aloitettiin vuoden 2008 alkupuolella. Tällöin yritys palkkasi LitheSpeed-nimisen yrityksen valmentamaan heitä siirtymässä.[10]

Siirtymässä yritys otti Scrumin käyttöön vaiheittain. Ensin työntekijät tutustutettiin ketterään kehitykseen luomalla parempia tiloja yhteistyöhön, sekä parantamalla kommunikaatiota eri tiimien välillä. Lopulta otettiin käyttöön kaikki Scrum sisältämät menetelmä, mutta muutamiin osa-alueisiin tehtiin kuitenkin pieniä muutoksia, jotta menetelmä olisi mahdollisimman hyödyllinen yritykselle. Ensimmäinen muutos tehtiin tuotteen omistajan roolin. Yhden tuotteen omistajan sijaan käytettiin tuotetiimiä, jossa oli mukana henkilö jokaisesta yrityksen segmentistä. Toinen muutos tehtiin product backlogiin. Tässä kaikki product backlogissa olevat käyttäjätarinat koottiin toimiston seinälle näkyviin, joten ne olivat helposti kaikkien työntekijöiden nähtävissä. Kolmas muutos liittyi backlogien ja julkaisuaikataulujen suunnitteluun, jossa molemmat suunniteltiin kerralla kuukausiksi eteenpäin.[10]

Ensimmäiset hyödylliset muutokset havaittiin lähes välittömästi Scrumiin siirtymisen jälkeen. Aiemmin yksilöinä toimineet työntekijät alkoivat työskennellä tiimeissä ja eri alojen työntekijät alkoivat auttaa toisiaan. Myöskin kaikki työntekijät tiimeissä olivat aikaisempaan verrattuna enemmän mukana ohjelmiston suunnittelussa. Lisäksi ohjelmakoodin testauksessa testien läpäisyprosentti ensimmäisellä yrittämällä kasvoi 60–70 %:sta 90 %:iin.[10]

Noin kuuden kuukauden kuluttua alettiin kuitenkin havaita erilaisia ongelmia uudessa menetelmässä. Sprinttien suunnittelukeskustelut alkoivat muuttua vaikeammiksi ja alkuperäiset julkaisuaikataulut aliarvioitiin pahasti. Tuotetiimillä tuli usein ongelmia käyttäjätarinoiden luomisessa ja niille oli vaikeaa saada hyväksyntää sidosryhmiltä. Tällöin keskeneräiset käyttäjätarinat ja viime hetken muutokset niihin turhauttivat kehittäjiä. Lisäksi käyttäjäkokemukseen liittyviä tarinoita kirjoitettiin liikaa, sillä jokaisesta yksityiskohdasta tehtiin aina oma käyttäjätarinansa.[10]

Näitä havaittuja ongelmia pyrittiin korjaamaan neljällä parannuksella. Ensimmäisenä projektin suunnittelu jaettiin kahteen vaiheeseen, jolloin ensimmäisessä vaiheessa pyrittiin määrittämään yleisesti projektin tavoitteita ja suurimpia haasteita sidosryhmien kanssa ja toisessa vaiheessa tehtiin korkean tason suunnitelmat ja luotiin backlogiin tehtäviä. Toisena korjauksena projekteista tehtiin visuaalisempia. Tähän käyttöön varattiin yksi yrityksen kokoushuoneista ja sen seinälle muodostettiin kuva projekteista eri värisistä muistilappujen avulla. Näin oli helppoa havaita projektien suurimmat tavoitteet ja niiden riippuvuudet toisista projekteista. Lisäksi projektitiimien päivittäisiä tapaamisia alettiin järjestää tämän seinän luona.[10]

Kolmantena parannuksena muutettiin käyttäjätarinoiden kehitysprosessia. Käyttäjätarinoita alettiin kehittää iteratiivisesti ja alettiin käyttää järjestelmää, jossa käyttäjätarinaa kehitettiin viidessä eri tasossa. Käyttäjätarina siirrettiin järjestelmässä seuraavalle tasolle heti, kun se toteutti kaikki nykyisen tason vaatimukset. Käyttäjätarinan päästessä viidennelle tasolle, se on toteutettu ohjelmaan ja on valmiina julkaistavaksi. Neljäntenä tehtiin parannus käyttöliittymän käyttäjätarinoihin. Tässä siirryttiin tekemään testejä erilaisten

käyttöliittymäprototyyppien avulla. Testeistä saatiin tietoa siitä, millainen käyttöliittymäelementti toimii hyvin ja millainen ei. Näin pystyttiin vähentämään käyttöliittymäelementteihin liittyvien yksityiskohtaisten käyttäjätarinoiden määrää.[10]

## 4.4 Tutkimus 4

Tutkimuksessa 4 pyritään selvittämään, voisiko turvallisuuskriittisessä, ilmailuun liittyvässä ohjelmistotuotannossa siirtyä käyttämään ketteriä menetelmiä edellisen vesiputousmenetelmän sijaan kokonaan tai jollain osa-alueilla. Ilmailuun liittyvässä ohjelmistotuotannossa on käytössä standardi DO-178B. Tämä standardi ei määrittele tarkkaan, mitä menetelmää tulisi projektissa käyttää, vaan mitkä tavoitteet tulisi täyttyä projektissa.[11]

Tutkimuksessa käytiin aluksi läpi, mitkä ketterät tavat voitaisiin ottaa suoraan käyttöön sellaisinaan. Näitä tapoja oli muun muassa ohjelman yksinkertainen suunnittelu, toimiva ohjelma onnistumisen mittarina, ohjelmakoodin yhteinen omistus koko tiimin kesken ja ohjelmointinormien noudattaminen. Toisena käytiin läpi tapoja, jotka voitaisiin ottaa käyttöön lähes sellaisinaan, mutta vaativat pieniä muutoksia. Näihin kuului esimerkiksi tiimien itsejohtaminen, jossa standardi vaatii, että korkeamman kriittisyyden projektissa täytyy olla itsenäinen johtaja ja testaaja projektilla. Lisäksi tähän kuului myös tuotantotiimin ja asiakkaan yhteistyö päivittäin, pariohjelmointi, juuri valmistuneen ohjelmakoodin liittäminen heti osaksi ohjelmaa ja testaaminen, sekä muutosten helpompi hyväksyminen projektiin.[11]

Itse tutkimuksessa päätettiin keskittyä seuraaviin aihealueisiin: testivetoinen kehitys, pariohjelmointi, ohjelmakoodin jatkuva integraatio, kiinteän mittaiset iteraatiojaksot ja asiakasvetoinen suunnittelu. Testivetoisessa kehityksessä otettiin kehittäjiä mukaan vaatimusten suunnitteluun, jotta testien tekemisestä tulisi helpompaa testausvaiheessa ja jottei vaatimuksia tarvitsisi muuttaa jälkikäteen testien vuoksi. Alkuperäisessä vesiputousmallissa suunnitteluvaiheessa ei käytetty yhtään aikaa vaatimusten suunnitteluun testaamisen kannalta, joten testejä oli hyvin vaikea toteuttaa, kun niiden aika tuli. Pariohjelmoinnissa laitettiin aluksi kaksi kokenutta kehittäjää pariksi ratkaisemaan jotain erittäin korkean riskin tehtävää. Tässä arvioitiin, että kahdella kehittäjällä saadaan tehtävä toteutettua paremmin, vähemmillä virheillä ja pienempään aikaan verrattuna yhteen kokeeseen kehittäjään. Lisäksi kokeiltiin yhden aloittelevan kehittäjän ja yhden kokeneen kehittäjän paria. Tässä aloitteleva kehittäjä sai kokemusta tällaisen ohjelmiston kehityksestä ja kokeneen kehittäjän oli helppo vastata aloittelevan kehittäjän kysymyksiin.[11]

Ohjelmakoodin jatkuvassa integraatiossa oli alun perin ongelma se, että eri ominaisuudet laitettiin yhteen ensimmäisen kerran vasta myöhäisessä vaiheessa tuotantoa, joten muutoksia oli tällöin vaikeampi toteuttaa. Nyt projektille luotiin automaattinen testausympäristö, joka testasi aina ohjelmaan lisättävän ominaisuuden. Tutkimuksessa havaittiin, että tämä ominaisuus oli hyvin hyödyllinen, sillä se mahdollisti eri osien yhteensopimattomuuden huomaamisen varhaisessa vaiheessa ja täten helpotti tämän korjaamista. Kiinteä

mittaisissa iteraatiojaksoissa havaittiin, että nämä auttoivat projektin suunnittelussa ja aikatauluttamisessa. Aiemmin tässä oli ollut käytössä vesiputous-menetelmälle tyypillinen tapa suunnitella projektin alussa koko projektin sisältö ja aikataulutus. Asiakasvetoisessa suunnittelussa asiakas otettiin suunnitteluun mukaan jokaisessa projektin vaiheessa, kun aiemmin asiakas oli ollut mukana suuresti vain alussa ylemmän tason suunnitelmaa tehtäessä.[11]

## 4.5 Tutkimus 5

Tutkimuksessa 5 pyritään selvittämään, miten erilaiset yrityksen ovat ottaneet käyttöön jonkin ketterän menetelmän. Tutkimuksessa ei määritelty, mitkä ohjelmistotuotantomenetelmät yrityksillä oli alun perin käytössä. Tutkimuksella pyrittiin vastaamaan millaisia menetelmiä tai niiden yhdistelmiä on tällä hetkellä käytössä yrityksellä, millaisia strategioita käytettiin ketterien menetelmien käyttöönotossa ja mitkä ovat suurimmat ja pienimmät hyödyt sekä haitat siirryttäessä ketterään menetelmään. Tutkimukseen osallistui alun perin 63 vastaajaa, mutta vain 34 vastaajaa suoritti tutkimuksen kaikkine vaiheineen loppuun asti.[12]

Tutkimus koostui kyselylomakkeesta, joka sisälsi neljä eri vaihetta. Ensimmäisessä vaiheessa kysymykset kohdistuivat vastaajan taustatietoihin. Tässä kysyttiin esimerkiksi vastaajan työvuosien määrää, työnkuvaa, yrityksen ja tuotantotiimin kokoa, sekä kokemusta erilaisista menetelmistä ja näiden asettaminen paremmuusjärjestykseen. Toisessa vaiheessa keskityttiin siihen, millaisia eri tapoja vastaajalla oli käytössä. Nämä tavat voivat olla sekä perinteisistä, että ketteristä menetelmistä. Kolmannessa vaiheessa vastaajaa pyydettiin asettamaan tärkeysjärjestykseen menetelmien erilaiset ulkoiset hyödyt, sisäiset hyödyt ja rajoitukset. Neljännessä vaiheessa vastaajan sai halutessaan lisätä omat yhteystietonsa kyselyn mukaan ja kommentoida kyselyä.[12]

Tutkimustuloksina ensimmäiseen vaiheeseen saatiin, että suurin osa kyselyyn vastanneista oli ohjelmoijia. Kyselyyn osallistui myös prosessi asiantuntijoita ja projektin johtajia. Yritysten koko vaihteli 50:stä työntekijästä 4500:een työntekijään ja tuotantotiimin koko oli yleisesti alle 11 henkilöä. Toisen vaiheen tuloksissa vastaajat jaettiin ryhmiin vastausten perusteella. Näitä ryhmiä oli esimerkiksi täysin ketteriä tapoja noudattavat vastaajat ja joitain perinteisiä tapoja ja ketteriä tapoja yhdistävät vastaajat. Tähän suositumpia vastauksia oli muun muassa tapaamiset kasvokkain, ohjelman jatkuva integraatio ja iteraation suunnittelutapaamiset. Kolmannen vaiheen tulokset olivat hyvin vaihtelevia vastaajien kesken. Vastaajien arvostelemat ulkoiset ja sisäiset hyödyt eri menetelmissä olivat tulosten mukaan hyvin samanarvoisia, eikä mikään niistä korostunut erityisesti. Rajoituksissa kuitenkin korostui erityisesti taitavien ammattilaisten tarve, joka näkyi monessa eri vastaajaryhmässä.[12]

Tutkimuksen lopuksi tutkijaryhmä teki vielä seuraavat havainnot. Yrityksen suurimmat näkyvät ulkoset hyödyt olivat arvo, laatu ja suhde asiakkaaseen. Suurimmat sisäiset hyödyt ketterän menetelmän käyttöönotosta olivat henkilöstön sosiaalisten taitojen parantuminen, luottamus, palaute, sekä tietämys ja oppiminen. Ketterien menetelmien käyttäminen vaatii tiimiin osaavia työntekijöitä ja johtajia. Siirtymistavalla perinteisestä ketterään menetelmään on väliä, sillä jotkut tavat johtavat huonompiin tuloksiin kuin toiset.[12]

# 5. TUTKIMUSTULOKSET JA ANALYSOINTI

Tässä luvussa käsitellään edellisessä luvussa läpi käytyjen tutkimusten tulokset. Tulokset jaettiin kolmeen ryhmään, joista ensimmäinen on kommunikaatio ja asiakkaan huomioiminen. Tämä sisältää projektitiimin sisäisen kommunikoinnin, asiakkaan kanssa kommunikoinnin ja projektissa asiakkaan toiveiden huomioimisen. Toinen ryhmänä on ohjelmiston laatu ja projektin näkyvyys. Tämä taas sisältää ohjelmiston laatuun vaikuttavia tekijöitä, kuten testauksen ja dokumentaation ja tekijöitä liittyen projektin näkyvyyteen tuotantotiimin ja asiakkaan näkökulmasta.

## 5.1 Kommunikaatio ja asiakkaan huomioiminen

Tutkimuksista voitiin havaita, että ketterään menetelmään siirtymisen jälkeen tuotantotiimin sisäinen kommunikaatio ja asiakkaan kanssa kommunikointi paranivat huomattavasti. Tuotantotiimin sisäisen kommunikaation parantumiseen vaikutti ryhmäkoon pienentyminen jopa noin 120:stä henkilöstä noin 30:een henkilöön. Lisäksi kommunikaatiota paransi ketteriin menetelmiin tärkeänä osana kuuluvat päivittäiset tapaamiset, joissa keskusteltiin projektin etenemisestä ja mahdollisista ongelmista tuotannossa. Tämän lisäksi tutkimuksessa 3 havaittiin, että ketterän menetelmän käyttöönotto innosti työntekijöitä ryhmäytymään keskenään ja jopa eri alojen työntekijät tekivät töitä ryhmässä.

Asiakkaan ja tuotantotiimin välinen kommunikaatio parani, sillä toisin kuin perinteisissä menetelmissä, joissa asiakas oli yleensä mukana vain aivan alussa tuotantoprosessia korkean tason suunnittelussa, nyt asiakas oli mukana ohjelmiston tuotannossa jokaisessa vaiheessa ja pystyi vaikuttamaan siihen. Lisäksi ohjelmiston toimitusaika lyheni esimerkiksi kuudesta kuukaudesta kolmeen viikkoon, jolloin asiakkaalla oli mahdollisuus nähdä, mihin suuntaan ohjelma on menossa ja antaa palautetta siitä. Tässä ketterä kehitys mahdollisti myöskin muutosten tekemisen helposti ohjelmaan, eli jos asiakas ei ollut tyytyväinen johonkin ominaisuuteen ohjelmassa, se voitiin helposti ottaa uudelleen kehitettäväksi ja muokata asiakkaan toiveiden mukaiseksi. Huonona puolena kuitenkin havaittiin tutkimuksessa 3, että viime hetkellä tehdyt muutokset aiheuttivat työntekijöissä turhautuneisuutta.

## 5.2 Ohjelmiston laatu ja projektin näkyvyys

Tutkimusten mukaan tuotettavan ohjelmiston laatu parantui ketterää menetelmää käytettäessä. Esimerkiksi tutkimuksen 3 mukaan ohjelma läpäisi sille laaditut testit ensimmäisellä yrityksellä 90 %:n varmuudella, kun aiemmin vastaava varmuus oli 60–70 %:a. Lisäksi tutkimuksessa 4 pariohjelmoinnin käyttöönottaminen vähensi ohjelmaan syntyviä virheitä ja ohjelmakoodia pystyttiin tuottamaan nopeammin entiseen verrattuna. Myöskin

testivetoinen kehitys paransi ohjelmiston laatua, sillä sen avulla voitiin heti testata eri koodiosioiden yhteensopivuudet toistensa kanssa.

Huonona puolena tutkimuksessa 1 havaittiin, ettei projektin dokumentaation tekemiseen jäänyt enää tarpeeksi aikaa nopeutuneen toimitusajan johdosta. Vaihejako-menetelmiin kuuluu erittäin tärkeänä osana työn dokumentointi ja tämä jää pienenpään arvoon ketterissä menetelmissä, joissa keskitytään enemmän itse ohjelmakoodin kirjoittamiseen, kuin sen dokumentoimiseen. Ohjelmiston laadun kannalta työn dokumentointi on tärkeää, sillä koodista on hyvä tehdä helposti ymmärrettävää ja ylläpidettävää.

Otettaessa käyttöön ketterä menetelmä projektin näkyvyys asiakkaalle parani huomattavasti. Tutkimuksessa 3 projektista tehtiin näkyvämpi työntekijöille luomalla projektista seinälle kuva, josta työntekijöiden oli helppo seurata esimerkiksi ohjelman eri osien riippuvuuksia toisistaan. Asiakkaalle projektin näkyvyys myöskin parani, sillä ohjelman kasvanut toimitustahti takasi asiakkaalle uuden version ohjelmasta jopa alle kuukauden välein, kun taas vaihejako menetelmällä olisi ohjelmaa pitänyt odottaa jopa kuusi kuukautta. Tutkimuksessa 4 havaittiin, että ketterään menetelmään käytettäessä asiakas oli enemmän mukana projektin suunnittelussa projektin jokaisessa vaiheessa. Aiemmin asiakas oli ollut mukana suunnittelussa vain projektin alkupuolella.

# 6. YHTEENVETO

Tässä työssä käsiteltiin siirtymistä perinteisistä vaihejakoisista ohjelmistotuotantomenetelmistä ketteriin menetelmiin ja pyrittiin löytämään hyviä ja huonoja muutoksia siirtymässä. Lopuksi pyrittiin vastaamaan kysymykseen: Oliko siirtyminen uuteen menetelmään kannattavaa? Työssä esiteltiin aluksi lyhyesti tutkimuksista löytyvät ohjelmistotuotantomenetelmät ja kerrottiin millaiset tavat missäkin on käytössä. Esiteltyihin menetelmiin kuuluivat seuraavat menetelmät: Rational Unified Process, vesiputous, Scrum ja Extreme Programming. Tämän jälkeen esiteltiin tietokannat, josta etsittiin aiheeseen liittyviä tutkimuksia, sekä kuvailtiin haussa käytettyjä hakulausekkeita ja kerrottiin haulla löytyneiden dokumenttien rajauksesta.

Seuraavaksi siirryttiin esittelemään valitut kuusi tutkimusta. Jokaisesta tutkimuksesta kerrottiin pääpiirteittäin tutkimuksen aloitusasetelma, tutkimuksen kulku ja tutkimuksen lopputulokset. Tämän jälkeen siirryttiin analysoimaan tutkimuksista saatuja tuloksia. Siirtymässä ketteriin menetelmiin havaittiin hyvinä puolina muun muassa parantunut vuorovaikutus tuotantotiimin eri jäsenten välillä, asiakkaan läsnäolon tärkeys projektin jokaisessa vaiheessa kasvoi ja muutokset ohjelmaan olivat helpompia toteuttaa. Siirtymässä löydettiin myös huonoja puolia. Näitä olivat esimerkiksi työn puutteellinen dokumentointi ja työntekijöiden turhautuminen viime hetken muutoksiin.

Tähän työhön valittujen tutkimusten tulosten perusteella voidaan päätellä, että siirtyminen ketterään menetelmään vaihejakoisesta menetelmästä on kannattavaa. Siirtymisessä havaittiin sekä hyviä, että huonoja muutoksia, mutta hyviä muutoksia saatiin kuitenkin reilusti enemmän kuin huonoja muutoksia ja huonot muutokset saatiin kuitenkin pääasiassa lopulta korjattua. Tulevaisuudessa aiheen tutkimista voisi jatkaa, mutta tällöin aiheesta tulisi tehdä oma tutkimus yhden tai usean yrityksen kanssa.

# LÄHTEET

[1] J. Dooley, Software Development and Professional Practice, 1st ed. Springer Verlag, Berkeley, CA, 2011, .

[2] A.K. Shuja, J. Krebs, I. Books24x7, IBM® Rational Unified Process® Reference and Certification Guide: Solution Designer, 1st ed. IBM Press, Upper Saddle River,, N.J, 2008, .

[3] A.T. Pham, P. Pham, I. Books24x7, Scrum in Action : Agile Software Project Management and Development, 1st ed. Cengage Learning, Boston, 2011, .

[4] Mountain Goat Software Scrum process, https://www.mountaingoatsoftware.com/agile/scrum/resources/overview.

[5] J.D. Wells XP flow Chart, http://www.extremeprogramming.org/map/loops.html.

[6] TTY kirjasto Andor, http://www.tut.fi/fi/kirjasto/ajankohtaista/andor-kaikkien-tiedonjanoisten-sankari-x158988c3.

[7] IEEE IEEE Xplore, http://ieeexplore.ieee.org/Xplorehelp/#/overview-of-ieee-xplore/about-ieee-xplore.

[8] R. Noordeloos, C. Manteli, H. V. Vliet, From RUP to Scrum in Global Software Development: A Case Study, 2012 IEEE Seventh International Conference on Global Software Engineering, pp. 31–40.

[9] H. C. Estler, M. Nordio, C. A. Furia, B. Meyer, J. Schneider, Agile vs. Structured Distributed Software Development: A Case Study, 2012 IEEE Seventh International Conference on Global Software Engineering, pp. 11–20.

[10] K. G. Fisher, A. Bankston, From Cradle to Sprint: Creating a Full-Lifecycle Request Pipeline at Nationwide Insurance, 2009 Agile Conference, pp. 223–228.

[11] S.H. VanderLeest, A. Buter, Escape the waterfall: Agile for aerospace, 2009 IEEE/AIAA 28th Digital Avionics Systems Conference, pp. 16.

[12] A. Solinski, K. Petersen, Blekinge Tekniska Högskola, Fakulteten för dataveten-skaper, Institutionen för programvaruteknik, Prioritizing agile benefits and limitations in relation to practice usage, Software Quality Journal, Vol. 24, Iss. 2, 2016, pp. 447.