Matti Lehtinen

Lean-ohjelmistokehityksen käyttöönoton jälkeisiä ohjelmistokehitysnopeutta rajoittavia tekijöitä

Tietotekniikan
pro gradu -tutkielma
9. lokakuuta 2011

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Matti Lehtinen
Yhteystiedot: matti.t.t.lehtinen@jyu.fi
Työn nimi: Lean-ohjelmistokehityksen käyttöönoton jälkeisiä ohjelmistokehitysnopeutta rajoittavia tekijöitä
Title in English: Constraints of software development velocity after Lean software development implementation
Työ: Tietotekniikan pro gradu -tutkielma
Sivumäärä: 107
Tiivistelmä: Tutkielmassa tarkastellaan, mitä Lean ja Lean-ohjelmistokehitys ovat, mitä Lean-ohjelmistokehitys tarkoittaa tapausyrityksessä, eli Sysdrone Oy:ssä, sekä mitkä ovat tapausyrityksen merkittävimmät ohjelmistokehitysnopeutta rajoittavat tekijät puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Tutkimus toteutettiin tapaustutkimuksena, jossa tutkimusaineistoa kerättiin puolen vuoden aikajaksolta. Johtopäätöksinä merkittävimmiksi rajoittaviksi tekijöiksi tunnistettiin tuotoslista, uudet teknologiat, projektien riittämätön läpinäkyvyys, tiettyjen henkilöiden ylikuormitus, puutteet prosessien noudattamisessa, dokumentointi sekä asiakaskommunikaatio.
English abstract: The thesis examines what Lean and Lean software development are, what Lean software development means in the case company or Sysdrone Ltd and what are the most biggest constraints of software development velocity in the case company half a year after adapting Lean software development. The research was made as a case study where research material were collected within half a year time period. Conclusions were that most significant restrictions are product backlog, new technologies, lack of transparency on projects, overloading of specific persons, processes were not always followed, documentation and customer communication.
Avainsanat: Lean, ohjelmistokehitys, kehitysprosessi, Kanban, arvo, arvovirta, virtaus, imuohjattu järjestelmä, rajoite, hukka
Keywords: Lean, software development, development process, Kanban, value, value stream, flow, pull system, constraint, waste

Esipuhe

Törmäsin työni ohessa sattumalta Leaniin ja Lean-ohjelmistokehitykseen. Lean vaikutti mielenkiintoiselta lähestymistavalta ohjelmistojen tekemiseen, sillä sen periaatteet ovat peräisin autojen valmistuksesta, jolla on ohjelmistojen valmistusta pidempi historia. Kiinnostukseni Lean-ohjelmistokehitystä kohtaan heräsi, ja ammattini yrittäjänä sekä ohjelmistokehitysjohtajana Sysdrone Oy:ssä mahdollisti asioiden viemisen eteenpäin sekä lopulta tämän tutkielman tekemisen.

Ensimmäiseksi haluan kiittää yrittäjäkumppaneitani Jani Lehtistä ja Ilkka Laitista, jotka ovat mahdollistaneet Leanin kokeilun ja käyttöönoton Sysdronen ohjelmistokehitysprojekteissa. Haluan osoittaa kiitokset myös Jaakko Kaskelle ja Joni Purojärvelle. Kaski on ollut ansiokkaasti viemässä agile coach -roolissa Lean-periaatteita mukaan Sysdronen käytännön tekemiseen, ja tuonut siten arvokasta näkemystä tässä tutkielmassa esitettyihin asioihin. Purojärvi on ollut pitkään mukana suunnittelemassa ketteriä ohjelmistokehitysprosesseja Sysdronelle, ja tuonut kokemuksellaan oman näkökulmansa esiin useisiin Lean-ohjelmistokehitykseen liittyviin seikkoihin. Haluan kiittää asiantuntevia ohjaajiani Anneli Heimbürgeriä ja Jonne Itkosta, jotka ovat ohjaamisen lisäksi haastaneet näkemyksiäni ja tuoneet siten esille uusia ajatuksia. Kiitos siskolleni tutkielman oikolukemista. Iso kiitos kuuluu tottakai myös vanhemmilleni, jotka ovat tukeneet minua opinnoissani.

Jyväskylässä lokakuussa 2011

Matti Lehtinen

Sanasto

5 miksi-kysymystä (5 whys) Menetelmä, jolla etsitään ongelmien perimmäisiä syitä vastatoimenpiteiden asettamiseksi kysymällä viisi kertaa peräkkäin miksi.

A3-raportit Raportointitapa, jossa kaikki tieto sisällytetään yhdelle A3-arkille, oli kyse miten laajasta asiasta hyvänsä. Tällöin raportin laatija joutuu miettimään esitystapaa ja turha teksti tulee karsittua pois.

Andon Visuaalinen ohjauslaite, joka ilmoittaa työntekijöille vioista, toimintahäiriöistä tai muista ongelmista.

CMM-malli (Capability Maturity Model) Tuotekehityksen kypsyysmalli, joka sisältää viisi tasoa ja eri prosessialueita. Malli kuvaa, kuinka hyvin organisaatio hallitsee prosessejaan. Taso 1 on aloitustaso ja taso 5 on paras taso.

Genchi genbutsu Meneminen paikanpäälle itse katsomaan todellisen tilanteen ymmärtämiseksi.

Hansei Tarkoittaa karkeasti ottaen itsearviointia. On osa japanilaista kulttuuria ja liittyy yksilön vastuuseen.

Heijunka Tuotannon ja aikataulujen tasapainottamista sekä tuotantomäärien että tuotevalikoiman suhteen. Esimerkiksi tasapainotetussa tuotannossa tehdään joka päivä sama määrä ja valikoima tuotteita.

Hukka Hukkaa on kolmenlaista: ks. muda, mura ja muri.

Imuohjaus Kysyntäpohjainen järjestelmä, jossa työ aloitetaan vasta signaalista. Esimerkiksi tuotteen valmistus aloitetaan vasta asiakkaan tilauksesta. Imuohjaus on työntöohjauksen vastakohta.

Jidoka Nelivaiheinen laadunvalvontaprosessi: havaitse vika, pysähdy, suorita välitön korjaustoimenpide, tutki perimmäinen syy ja aseta vastatoimet.

JIT (Just In Time, Juuri Oikeaan Tarpeeseen) Toimitetaan vain ja ainoastaan tarvittavia artikkeleita niitä tarvitsevalle asiakkaalle vasta silloin, kun niitä tarvitaan, ja vain sen verran kuin niitä tarvitaan.

Kaizen Inkrementaalinen jatkuva parantaminen.

Kanban Signaali, jota käytetään ilmoittamaan artikkelien tarpeesta ja joka aiheuttaa täydennyksen. Voi olla esimerkiksi kortti, värillinen golf-pallo tai tyhjä kori. Ohjelmistokehityksessä voidaan tarkoittaa myös Kanban-prosessimallia, joka on ketterä ohjelmistokehitysmenetelmä.

Läpimenoaika (Lead time) Ohjelmistokehityksessä aika vaatimuspyynnön saamisesta asiakkaalta siihen, kun vaatimus on toimitettu ja käyttöönotettu asiakkaan tuotantoympäristöön.

Muda Lisäarvoa tuottamaton toiminta, jota on alun perin tunnistettu valmistuksessa seitsemää perustyyppiä: ylituotanto, odottelu, tarpeeton kuljetus, ylikäsittely, liiallinen varasto, tarpeeton liike ja viat.

Mura Epätasaisuudesta tai varianssista johtuva hukka.

Muri Ihmisten, laitteiden, järjestelmien tai muusta ylikuormituksesta johtuva hukka.

Poka-yoke Tuotannon virheitä estävä laite tai järjestelmä.

Tahtiaika (Cycle time) Ohjelmistokehityksessä aika vaatimuksen analysoinnista siihen, kun se on valmis käyttöönotettavaksi tuotantoympäristöön.

TOC-teoria (Theory of Constraints) Esteiden teoria, joka perustuu ajatukseen, että jokaisessa järjestelmässä on este tai rajoite. Jollei rajoitetta olisi, järjestelmän tuotos olisi joko ääretön tai nolla.

Tuotteen omistaja (Product owner) Scrum-prosessimallista lähtöisin oleva rooli, joka edustaa asiakasta, liiketoimintaa ja sidosryhmiä. Tuotteen omistajan tärkeimpiä tehtäviä on tuotoslistan ylläpito.

Tuotoslista (Product backlog) Lista, joka sisältää kaiken tuotteeseen suunnitellun toiminnallisuuden sekä ohjelmistovirheet, tekniset työt ja muut mahdolliset asiat, jotka tarvitsee tehdä tuotteen valmiiksi saattamiseksi.

Työntöohjaus Järjestelmä, jossa toimitus tai valmistus tapahtuu riippumatta senhetkisestä kysynnästä. Esimerkiksi tietty määrä hyödykkeitä ”työnnetään” jälleenmyyjälle tasaisin väliajoin riippumatta jälleenmyyjän senhetkisestä varastotilanteesta.

Sisältö

1 Johdanto

Tämä tutkimus käsittelee Lean-ohjelmistokehitystä ja tapausyrityksen ohjelmistokehityksen nopeutta rajoittavia tekijöitä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Tapausyrityksenä toimii Sysdrone Oy, joka on jyväskyläläinen ohjelmistotalo.

Lean on japanilaista alkuperää oleva yleismaailmallinen filosofia ja ajattelutapa, jota voidaan soveltaa lähestulkoon kaikilla aloilla [WJR07]. Se pohjautuu Toyotan tapaan *(engl. The Toyota Way)* suunnitella ja valmistaa autoja [Lik04].

Lean on tapa tehdä enemmän vähemmillä resursseilla. Lean-nimitys juontaa juurensa siihen, että Leanissa kaikkea (esimerkiksi resursseja, työntekijöitä, varastoa ja tarvittavaa aikaa) on vähemmän [WJ03]. Sen käytänteet kattavat koko yrityksen ja sitä voidaan soveltaa lähestulkoon kaikille liiketoiminta-alueille. Leania käytetään mm. autoteollisuudessa, Yhdysvaltojen asevoimissa, terveydenhuollossa, tuottoa tavoittelemattomissa organisaatioissa, jälleenmyynnissä, lentokoneteollisuudessa, pankin alalla, terveydenhuollossa ja ohjelmistoteollisuudessa [SW07, s. 11–12].

Lean-ohjelmistokehitys on Lean-ajattelun soveltamista ohjelmistojen kehittämiseen, mikä on varsin tuore sovellusalue Leanille. Ohjelmistoteollisuuden erityispiirteenä on, että toimintatapoja muutettiin useissa yrityksissä ketterien menetelmien myötä 1990-luvun loppupuolella ja 2000-luvun alussa. Samoihin aikoihin ilmestyivät James Womackin, Daniel Jonesin ja Daniel Roosin teos *The Machine That Changed World* (1990) [WJR07] sekä Womackin ja Jonesin teos *Lean Thinking* (1996) [WJ03]. Nämä kirjat toivat Toyotan ajattelumallin ja tuotantojärjestelmän yleiseen tietoisuuteen, mistä seurasi Leanin yleistyminen muillakin kuin autoteollisuuden alalla. Ohjelmistoala on kuitenkin vasta viimeaikoina alkanut kiinnostumaan Leanista. Lean-ohjelmistokehityksen tunnetuimpina vaikuttajina voidaan pitää Mary ja Tom Poppendieckiä, joiden kirjat *Lean Software Development - An Agile Toolkit* ja *Implementing Lean Software Development - From Concept To Cash* ovat vaikuttaneet suuresti siihen, mitä Lean-ohjelmistokehityksen ajatellaan olevan.

Koska Lean-ohjelmistokehitys on varsin uusi tapa kehittää ohjelmistoja, sen käyttöönottoon liittyy todennäköisesti myös haasteita. TOC-teoria on Eliyahu M. Goldrattin kehittämä esteiden teoria, jonka mukaan jokaisessa järjestelmässä on rajoite, mikä hidastaa järjestelmän nopeutta. Ilman rajoitteita järjestelmän nopeus olisi joko ääretön tai nolla. Tämä tutkimus keskittyy etsimään näitä rajoitteita tapausyrityksen

ohjelmistotuotannosta Lean-käyttöönoton jälkeen. Aineistoa kerättiin vuoden 2011 tammikuusta saman vuoden kesäkuun loppuun saakka. Itse rajoitteet eivät välttämättä liity suoraan Lean-ohjelmistokehitykseen, mutta Lean-ajattelu itsessään antaa hyvät mahdollisuudet löytää näitä rajoitteita ja siten poistaa niitä.

1.1 Tutkimuksen tavoite

Tutkimuksen tavoitteena on selvittää suurimmat ohjelmistokehityksen nopeutta hidastavat pullonkaulat Sysdrone Oy:ssä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Toisin sanoen tarkoituksena on löytää ne tekijät, jotka rajoittavat sillä hetkellä ohjelmistojen tekemisen nopeutta. Nopeudella tarkoitetaan, kuinka nopeasti asiakasvaatimus pystytään toteuttamaan sen pyynnöstä siihen, että se on asiakkaan käytettävissä. Keskeisin tutkimuskysymys tulee suoraan tavoitteesta:

- *Mitkä ovat merkittävimmät ohjelmistokehityksen nopeutta rajoittavat tekijät tutkittavassa yrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen?*

Apukysymyksiä ovat:

- *Mitä Lean-ohjelmistokehitys on?*
- *Mitä Lean-ohjelmistokehitys tarkoittaa tutkittavassa yrityksessä?*

Ensimmäisen apukysymyksen tarkoituksena on auttaa ymmärtämään, mitä Lean-ohjelmistokehityksellä tarkoitetaan. Jälkimmäinen kysymys ohjaa selvittämään, mitä Sysdronessa Lean-ohjelmistokehityksen ajatellaan olevan. Tarkoituksena on tunnistaa teorian ja käytännön toimintatapojen välinen ero. Lean-ohjelmistokehityksen ymmärtäminen tutkittavassa yrityksessä on tutkimuksen kannalta oleellista, jotta tutkimusaineiston luotettava analysointi on mahdollista.

Ohjelmistokehityksen nopeutta rajoittavien tekijöiden selvittäminen on tärkeää, sillä tutkimustulosten perusteella tapausyrityksessä tiedostetaan ne asiat, joiden poistamiseksi täytyy tehdä työtä, mikäli käyttöönotettua Lean-ajattelun mukaista ohjelmistokehitystä halutaan nopeuttaa. Nopeampi ohjelmistokehitys mahdollistaa mm. ohjelmistojen tuotantokustannuksien laskemisen ja parantaa siten asiakastyytyväisyyttä. Tällöin asiakas saa haluamansa ohjelmiston edullisemmin tai vaihtoehtoisesti samalla budjetilla enemmän ominaisuuksia tilaamaansa ohjelmistoon. Asiakas saa myös ohjelmistonsa nopeammin markkinoille, mikä voi olla ratkaiseva kilpailuetu haasteellisessa markkinatilanteessa.

1.2 Tutkimusmenetelmät

Tutkimuksen tutkimusmenetelmänä on tapaustutkimus. Tapaustutkimuksessa tarkoituksena on tutkia syvällisesti vain yhtä tai muutamaa kohdetta tai rajattua kokonaisuutta käyttämällä monipuolisia ja eri menetelmillä hankittuja tietoja [SKP06]. Tapaustutkimuksessa pyritään ymmärtämään yksittäisiä tapauksia niiden erityisessä kontekstissaan. Täten tapaustutkimus soveltuu hyvin yksittäisen yrityksen sisällä tapahtuvaan tutkimukseen.

Tässä tutkimuksessa on kyseessä niin sanottu yhden tapauksen tutkimus, jossa tutkimustapauksena toimii Jyväskylässä sijaitseva ohjelmistoyritys Sysdrone Oy. Koska kyseessä on tapaustutkimus, tavoitteena ei ole löytää yleistyksiä, vaan tutkimustulokset ovat sovellettavissa ainoastaan tapausyrityksen sisällä.

Aineistona toimii tapausyrityksen ohjelmistokehitysprosessin kehityspalavereissa syntyneet muistiot sekä asiakasvaatimuskohtaiset mittaustiedot kehitysprosessin eri työvaiheisiin käytetystä ajasta. Tutkimuksessa hyödynnetään siis olemassa olevaa sekä laadullista että määrällistä aineistoa, mikä on tavallista tapaustutkimuksessa [JJ04, s. 75]. Laadullisesta aineistosta kerätään esilletuodut parannusehdotukset ja ne taulukoidaan. Taulukointia varten ehdotukset kategorisoidaan ennalta määritettyihin kategorioihin. Määrällisestä aineistosta muodostetaan aikasarja-analyysi, jonka avulla voidaan tehdä johtopäätöksiä kehitysprosessissa sillä hetkellä olevista pullonkauloista.

Analysoiden molempia aineistoja ja niistä saatuja tuloksia tehdään päätelmiä tapausyrityksen Lean-ohjelmistokehityksen pullonkauloista. Lopuksi tutkimustuloksista vedetään yhteenveto ja esitetään ehdotuksia esille tulleiden pullonkaulojen poistamiseksi.

1.3 Rajaukset

Leanista on kaksi näkökulmaa: japanilainen ja länsimaalainen. Teoriaosassa ohjelmistokehityksen osalta keskitytään pääosin länsimaalaiseen näkökulmaan. Japanilaisesta näkökulmasta poimitaan vain tiettyjä asioita. Näin voidaan keskittyä kuvaamaan sitä, miten Lean Suomessa useimmiten käsitetään. Tutkielma antaa lukijalle perustiedot Leanista, mutta aihealueen laajuuden vuoksi tarkempien yksityiskohtien kuvaaminen jätetään tutkielman ulkopuolelle.

Tutkielmassa oletetaan, että lukijalla on perustietämys ohjelmistokehityksestä ja ketteristä ohjelmistokehitysmenetelmistä. Näiden kuvaaminen on rajattu ulos teoriaosuudesta, jolloin voidaan keskittyä kuvaamaan Lean-ohjelmistokehitystä tar-

kemmin.

Empiirisessä osassa keskitytään tutkimaan tapausyritystä. Muut organisaatiot rajataan tutkimuksen ulkopuolelle. Tutkimuksessa ei pyritä löytämään yleismaailmallisia haasteita Lean-ohjelmistokehityksen käyttöönotossa vaan tutkitaan ainoastaan yhtä tapausta. Tutkimuksessa ei myöskään oteta kantaa tapaan, jolla Lean-ohjelmistokehitystä toteutetaan tapausyrityksessä muuten kuin vertaamalla sitä lyhyesti kirjallisuudessa esitettyihin näkemyksiin Lean-ohjelmistokehityksestä. Myös sen tutkiminen, onko Lean-ohjelmistokehityksen käyttöönotto nopeuttanut ohjelmistokehitystä tapausyrityksessä, rajataan tutkimukseen kuulumattomaksi.

1.4 Tutkielman rakenne

Toisessa luvussa käsitellään Leanin historiaa. Tähän liittyy eri tuotantoparadigmojen ymmärtäminen sekä Toyotan kasvutarina. Kolmannessa luvussa siirrytään käsittelemään itse Leania ja sen perusperiaatteita sekä tutustutaan kahteen eri näkökulmaan Leanista ja vertaillaan näiden suhdetta toisiinsa. Lean-ajattelun periaatteisiin tutustumisen jälkeen neljännessä luvussa katsotaan, kuinka niitä voidaan soveltaa ohjelmistokehitykseen. Viidennessä luvussa esitellään tapausyritys Sysdrone Oy sekä tarkastellaan Lean-ohjelmistokehitystä kyseisessä yrityksessä. Kuudennessa luvussa käydään läpi käytetyt tutkimusaineistot ja -menetelmät. Seitsemännessä luvussa tarkastellaan ja analysoidaan tutkimustuloksia. Kahdeksannessa luvussa tehdään johtopäätöksiä tutkimustulosten pohjalta, pohditaan tutkimuksen luotettavuutta sekä esitetään johtopäätösten perusteella suosituksia ja mietitään jatkotutkimustarpeita. Viimeisessä eli yhdeksännessä luvussa vedetään tutkimus yhteen.

2 Leanin historia

Lean-ajattelu pohjautuu Lean-tuotantoon, joka tulee suurelta osin japanilaisesta autoteollisuudesta. Teollisuus on historiansa aikana käynyt läpi kolme tuotantoparadigmaa: käsityöläisyys, massatuotanto ja Lean-tuotanto. Näistä kaksi jälkimmäistä on syntynyt autoteollisuudessa, mutta niiden, erityisesti massatuotannon, vaikutukset ulottuvat jopa siihen, kuinka ihmiset päivittäin ajattelevat arkisia asioita. Leanissa on siis kyse erittäin laajasta asiasta. [MS05]

Koska Lean on myös filosofia ja ajattelutapa, on sen ymmärtämiseksi tärkeää tuntea sen historiaa. Leanin historia pohjautuu edellä mainittuihin tuotantoparadigmoihin.

2.1 Käsityöläisyys

Käsityöläisyyttä voidaan pitää ensimmäisenä tuotantojärjestelmänä. Se oli ainoa tuotantojärjestelmä ennen teollista vallankumousta ja pohjautuu lähes kokonaan yksilön kokemukseen ja osaamiseen. Normaalisti tuotteet valmistettiin yksi kerrallaan ammattikuntaan kuuluvien käsityöläisten toimesta, joilla oli usein oppipoika apunaan. [MS05, s. 9–12]

Käsityöläisyydessä standardeja ja prosesseja on vain vähän tai ei ollenkaan, minkä takia se on helpoin tuotantojärjestelmä ymmärtää. Pääsääntöisesti kaikki uudet toimialat ovat lähteneet liikkeelle käsityöläisyydestä. Taitavat työntekijät toteuttavat suunnitelmat parhaaksi katsomallaan tavalla. Yksi työntekijä saattaa valmistaa tuotteen alusta loppuun täysin itsenäisesti. Koska toimialasta opittu tieto on enimmäkseen työntekijöiden päässä, on kokemuksella merkittävä rooli käsityöläisyydessä. Lopputuloksena kaksi käsityöllä valmistettua tuotetta eivät koskaan ole samanlaisia keskenään. Käsityö on kuitenkin erittäin asiakasystävällinen ja joustava malli, koska asiakas voi esittää omat toiveensa juuri hänen tuotteeseensa. Samalla käsityö on kuitenkin erittäin kallista. Työntekijöiden kannalta käsityö on palkitsevaa, koska he pääsevät toteuttamaan itseään ja heillä on työn osalta varsin vapaat kädet. Jotkut liittävät korkean laadun käsityöhön. Tämä on kuitenkin täysin riippuvaista työn tekijästä. [MS05, s. 9–12]

Joitain tuotteita tehdään nykyisinkin käsityönä. Se sopii parhaiten tuotteille, jotka ovat pieniä, yksinkertaisia ja joissa jokainen tuote eroaa toisistaan [MS05, s. 11].

Päteekö tämä ohjelmistoille? Sen sijaan, että ohjelmistot muuttuisivat pienemmiksi ja yksinkertaisimmiksi, koot ovat kasvamaan päin [MS05, s. 11]. Liiketoimintaohjelmistot sisältävät enemmän ja enemmän toimintoja, jotka aiemmin oli tapana hoitaa erillisillä apuohjelmilla. Teollisuusohjelmistotkin muuttuvat yleispätevimmiksi ajan myötä. Ohjelmistot ovat myös monimutkaisia. Näillä perusteilla ohjelmistoja ei kannattaisi tehdä käsityönä. Ohjelmistot ovat kuitenkin yksilöitä, mikä tarkoittaa, että ala on lähempänä käsityötä kuin moni muu ala.

Joissain kriittisissä järjestelmissä jopa ihmishenkiä voi olla vaarassa, mikäli ohjelmisto toimii väärin. Tämä sotii ohjelmistojen käsityönä tekemistä vastaan, sillä käsityönä tehdessä ohjelmistojen laatu on täysin yksilöiden osaamisen varassa. Käsityö ei täten ole ainakaan laajassa mittakaavassa erityisen hyvä paradigma ohjelmistokehitykseen.

2.2 Amerikkalainen massatuotantomalli

Yhdysvalloilla oli 1790-luvun loppupuolella ongelma: heillä ei ollut riittävästi aseita puolustaakseen itseään. Thomas Jefferson oli tuolloin diplomaattina Pariisissa ja paikalla heinäkuussa 1785, kun aseseppä Honoré Blanc oli kutsunut korkea-arvoisia sotilaita ja diplomaatteja paikalle katsomaan demonstraatiota. Blanc oli ottanut 50 kivääristä pois mekanismit, joita hän kutsui lukoiksi, ja laittanut ne laatikoihin. Vierailijat saivat poimia laatikoista lukot sattumanvaraisesti ja asettaa ne kivääreihin huomatakseen, että mikä tahansa lukko sopi mihin tahansa kivääriin. Jefferson tajusi tuolloin, että jos keskenään vaihdettavia osia pystytään tuottamaan helposti, lähes kouluttamatonta työvoimaa käyttäen voidaan koota halvalla suuri määrä aseita. Tämä oli alkusykäys massatuotantomallille. [PP08, s. 1]

Varsinaisen massatuotantoparadigman kehitti Henry Ford 1910-luvulla ja mullisti samalla maailmaa. Hän sai muun muassa pienennettyä auton kokoamiseen tarvittavan ajan 12 tunnista 90 minuuttiin, korotettua työntekijöiden päiväpalkat 2,40 dollarista viiteen dollariin ja pienennettyä yhdeksän tunnit työpäivät kahdeksan tunnin mittaisiksi [PP08, s. 2–3]. Massatuotannolle viisi ominaista luonteenpiirrettä ovat toistettavuus, suuret infrastruktuurit, tehokkuus, suuret organisaatiot ja teknologiakeskeisyys [MS05, s. 12].

Henry Ford pilkkoi tuotantolinjan mahdollisimman pieniin osiin. Hän etsi parhaan tavan tehdä jokainen yksittäinen osatehtävä. Standardoiduilla ja toistuvilla työtehtävillä mahdollistettiin tuotantolinjan työntekijän kouluttaminen tehtäväänsä 10 minuutissa. Ei tarvittu korkeasti koulutettua työvoimaa kokoamaan autoja. Keskenään vaihdettavat työntekijät olivat massatuotannon mahdollistamiselle yhtä

tärkeä tekijä kuin keskenään vaihdettavat osat, jotka keksittiin vuosisata aikaisemmin. [PP08, s. 2–3]

Massatuotantomallin haittapuolena on sen joustamattomuus. Jokainen liukuhihnalla valmistettu tuote on keskenään samanlainen riippumatta asiakkaiden erilaisista tarpeista. Fordkaan ei pystynyt juuri varioimaan tuotettuja autoja. Järjestelmä oli suunniteltu tuottamaan valtava määrä autoja, mutta vain muutamaa eri mallia. Liukuhihnalla tuotetut tietyn automallin autot olivat keskenään identtisiä. Ääriesimerkkinä kaikki ensimmäiset T-Fordit olivat tästä johtuen mustia. [Lik04, s. 20–21]

Ohjelmistokehityksessä jokaista kehitettyä ohjelmistoa voidaan pitää yksilönä. Joistain ohjelmistoista valmistetaan lähes joka vuosi uusi versio, mutta silti uudet versiot ovat yksilöitä, jotka on usein tehty inkrementaalisesti vanhan version päälle. Näin ollen massatuotannon periaatteet eivät ainakaan sellaisenaan sovellu ohjelmistokehitykseen. Tätä käsitystä vahvistaa myös se, että ohjelmistoja kehitetään, mutta ei tuoteta, kuten esimerkiksi autoja. Autoihin liittyy sekä uuden automallin kehittäminen että sen tuottaminen. Ohjelmistokehityksessä monistusvaiheen, johon massatuotanto on kehitetty, voi mieltää ilmaiseksi. Kehityksen ja tuotannon eroista kerrotaan lisää luvussa 4.

Nykypäivän ohjelmistokehityksestä on tunnistettavissa massatuotannon luonteenpiirteitä, joita ovat siis toistettavuus, suuret infrastruktuurit, tehokkuus, suuret organisaatiot ja teknologiakeskeisyys.

Massatuotannon tehokkuusajattelusta on esimerkiksi peräisin ajattelumalli, että komponenteista kasataan valmis tuote, joka testataan ja korjataan tarvittaessa. Iso osa ohjelmistoista valmistetaan tämän kaltaisella mallilla, jossa testaus on prosessin loppupäässä (tunnetaan ohjelmistokehityksessä vesiputousmallina). Ei ole kuitenkaan syytä olettaa, että kyseinen malli olisi luonteenomaisin tai tehokkain tapa valmistaa ohjelmistoja [MS05, s. 12–19]. Ohjelmistoja ei myöskään pääsääntöisesti voida koota valmiista komponenteista, vaikka tähän on pyritty.

Myös useat muut perinteiset ajattelumallit, joita ohjelmistokehityksessä yleisesti noudatetaan, ovat peräisin massatuotannosta. Mikäli lukija haluaa perehtyä näihin tarkemmin, löytyy lisätietoa esimerkiksi Peter Middletonin ja James Suttonin kirjasta *Lean Software Strategies* [MS05].

2.3 Lean-tuotanto

Toyoda Automatic Loom Works perustettiin vuonna 1926 valmistamaan ja markkinoimaan Sakichi Toyodan (kuva 2.1) kehittämiä automaattisia kangaspuita (kuva 2.2) [Toya, s. 16–17]. Sakichi Toyoda palkkasi amerikkalaisen insinöörin nimel-

Kuva 2.1: Sakichi Toyoda [Wik]

tä Charles A. Francis tuomaan amerikkalaisen tuotantomallin yhtiöönsä. Francis muun muassa kehitti määritykset ja standardoi osat. Automaattiset kangaspuut olivat kuitenkin erittäin monimutkaiset valmistaa, ja niinpä Toyoda joutui amerikkalaisen mallin vastaisesti käyttämään kaikista taitavimpia insinöörejä, jotka olivat saaneet koulutuksen Japanin yliopistoissa. Toyodan työntekijät eivät olleet keskenään vaihdettavia, mikä oli Henry Fordin massatuotantoparadigman yksi kulmakivi. Uuden tuotantojärjestelmän voidaan katsoa saaneen alkunsa. Tämä Toyodan uusi malli oli sekoitus sekä käsityöläisyyttä että massatuotantoa. [PP08, s. 3–4]

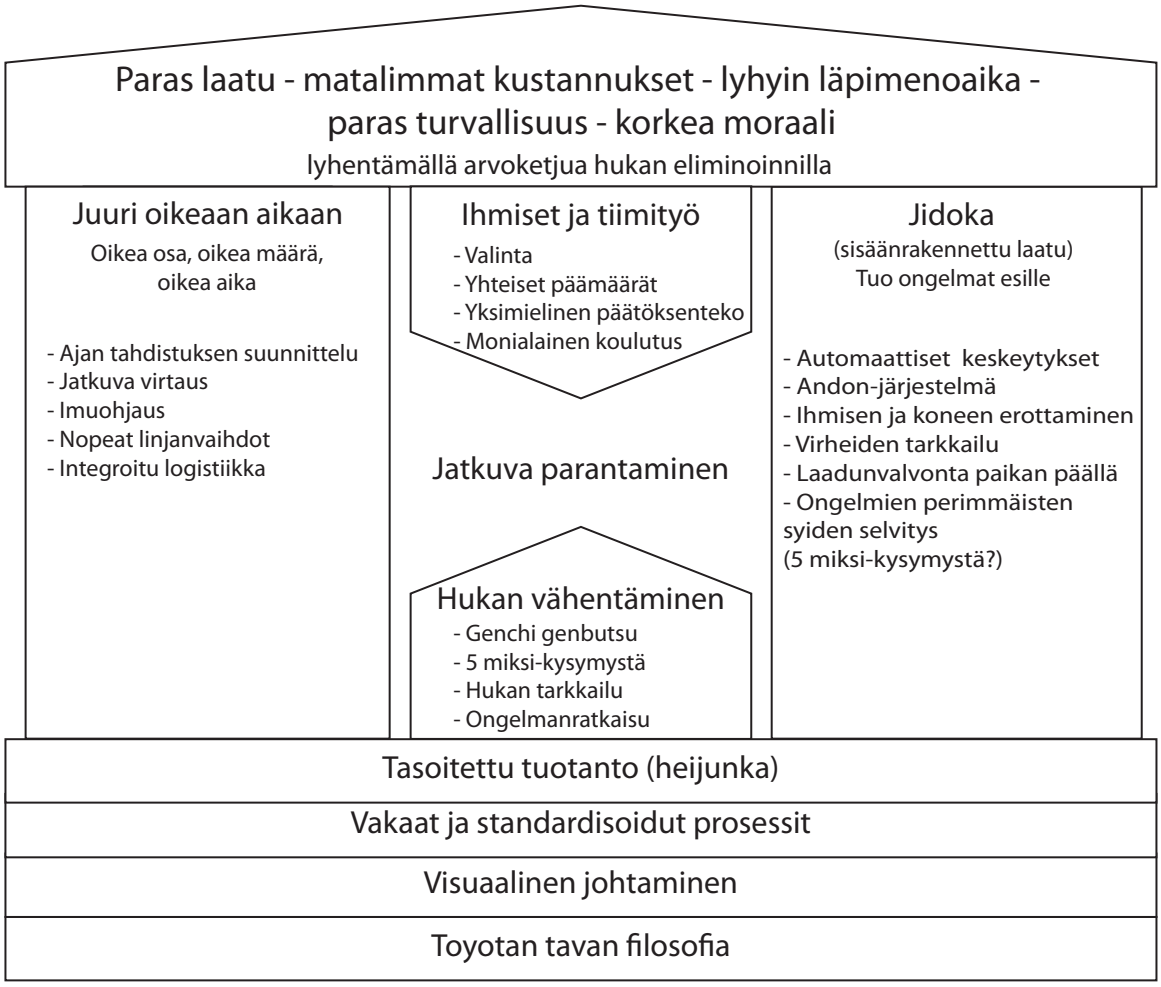
Kuva 2.2: Toyodan automaattiset kangaspuut (tyyppi G)

Eräs käänteen tekevistä hetkistä oli helmikuussa 1927. Toyoda Automatic Loom Works piti tuolloin tilaisuuden tekstiili-insinööreille esitelläkseen yhtiön uusia au-

tomaattisia kutomakoneita. Ensin vierailijat näkivät, kuinka kutomakoneet valmistettiin taitavien ammattilaisten toimesta tarkoilla työkaluilla. Tämän jälkeen heidät vietiin tilaan, jossa oli käytössä 520 kappaletta Toyoda-kutomakoneita, ja niiden käyttämiseen tarvittiin ainoastaan 20 kutojaa. Tämä oli tuohon aikaan suuri saavutus. [PP08, s. 4]

Sakichi Toyodan poika Kiichiro lähetettiin isänsä toimesta vuonna 1929 Englantiin neuvottelemaan suosituimman automaattisen kutomakoneen patenttioikeuksien myymisestä Platt Brothers-yhtiölle. Kiichiro neuvotteli hinnaksi 100 000 Englannin puntaa. Näitä voittoja hyväksikäyttäen hän aloitti autoteollisuudessa. Kiichiro vietti vuosia Detroitissa hakeakseen oppia Yhdysvalloista. Toyota Motor Company perustettiin vuonna 1937. Vuonna 1945 Kiichiro Toyoda haastoi yhtiönsä ”saavuttamaan Amerikan”, mutta oli selvää, että Toyota ei voinut saavuttaa heitä käyttämällä amerikkalaista massatuotantomallia. Massatuotanto vaati tuhansien identtisten osien valmistusta, mutta materiaaleja oli vähän ja tilaukset olivat vaihtelevia. Oli siis tarve voida valmistaa tehokkaasti pieniä eriä. Kiichiro Toyodan visio oli juuri oikeaan tarpeeseen *(engl. Just-in-Time, JIT)* -tuotantolinja, jossa tarvittavat osat saapuisivat linjalle juuri silloin, kun niitä tarvitaan. Visio toteutui noin 10 vuotta Kiichiro Toyodan kuoleman jälkeen vuonna 1962. Alettiin puhua Toyotan tuotantojärjestelmästä *(engl. Toyota Production System, TPS)*, jonka yhtiö otti käyttöönsä maailmanlaajuisesti. Toyotan tuotantojärjestelmän perustajana pidetään Taiichi Ohnoa. [PP08, Lik04, Toyb]

Toyotan tuotantojärjestelmää havainnollistava niin sanottu TPS-talokaavio on nähtävillä kuvassa 2.3. Tästä on tullut yksi nykyaikaisen tehdasvalmistuksen tunnistettavimmista symboleista [Lik10, s.32]. Talolla viitataan rakenteelliseen järjestelmään, joka on vahva vain, jos sen kaikki osat ovat vahvoja. Heikko lenkki heikentää koko järjestelmää. Päämääränä on katosta löytyvät asiat: paras laatu, matalimmat kustannukset, lyhyin läpimenoaika, paras turvallisuus ja korkea moraali. Tämän jälkeen tulevat ulkopilarit: juuri oikeaan aikaan, joka on luultavasti TPS:n tunnetuin ominaisuus, ja jidoka, eli TPS:n laadunvalvonta. Järjestelmän keskellä ovat ihmiset ja hukan parantaminen, jotka molemmat ovat hyvin keskeisiä asioita TPS:ssä. Järjestelmän perusta ovat neljä pohjaelementtiä. Heijunka tarkoittaa tuotantoaikataulun tasoittamista sekä volyymin että valikoiman suhteen. Vakaat ja standardoidut prosessit tarvitaan, jotta asiat tehdään aina samalla tavalla. Tällöin toimintatapoja pystytään parantamaan. Visuaalinen johtaminen merkitsee visuaalisen ohjauksen käyttämistä työympäristöissä. Esimerkiksi tavoitetuotantomäärä ja nykyinen tuotantomäärä voivat näkyä valotauluissa. Pohjimmaisena on Toyotan tavan filosofia, mihin kaikki viime kädessä perustuu. [Lik10]



Kuva 2.3: Toyotan tuotantojärjestelmä [Lik10, s. 33]

Toyotan tuotantojärjestelmä, eli TPS, on laaja kokonaisuus, jonka tarkempi käsittely joudutaan rajaamaan tämän tutkielman ulkopuolelle. TPS:stä voi lukea lisää esimerkiksi Jeffrey K. Likerin kirjasta *The Toyota Way* [Lik04].

Lean-tuotanto, joka tunnetaan myös nimellä Lean-valmistus, on Toyotan tuotantojärjestelmästä johdettu yleisempi malli, jota voidaan soveltaa myös muille aloille [Lik04]. Toisin kuin käsityöläisyys ja massatuotanto, Lean-tuotanto keskittyy jatkuvasti parantamaan olemassa olevia prosesseja ja poistamaan hukkaa *(engl. waste)* niistä [MS05, s. 3–4]. Tätä käsitellään tarkemmin luvussa 3. Lean-tuotannossa oleellista on asiakaskeskeisyys, kun massatuotannossa pääpainopiste on teknologiassa. Lean-tuotantoon liittyy myös *juuri oikeaan tarpeeseen* -ajattelu.

Termi Lean esiintyi ensimmäisen kerran syksyllä 1988 John Krafcikin [Kra88] artikkelissa *Triumph of the Lean Production System* [Hol07]. Krafcik oli ollut laadunvalvontatehtävissä Toyotan ja General Motorsin yhteisyrityksessä Nummi-tehtaalla Kaliforniassa. Hän valitsi termin Lean *(suom. hoikka)*, koska Lean-tuotannossa käytetään kaikkea vähemmän kuin massatuotannossa — työvoimaa, valmistustilaa, työkaluja ja työtunteja [WJR07, s.11]. Yleisemmin termi otettiin käyttöön James Womackin ja Daniel Jonesin vuonna 1990 julkaiseman kirjan *The Machine That Changed the World* myötä [PP08, s. 11]. He kutsuivat kirjassa Toyotan lähestymistapaa tuotantoon Lean-tuotannoksi vastakohtana länsimaalaiselle massatuotannolle [Hol07]. Lean-termin synnystä voi lukea tarkemmin Matthias Holwegin [Hol07] artikkelista *The genealogy of Lean production*.

2.4 Yhteenveto

Teollisuus on historiansa aikana käynyt läpi kolme vaihetta: käsityöläisyys, massatuotanto ja uusimpana Lean-tuotanto. Lean-ajattelu pohjautuu Lean-tuotantoon, joka on alun perin peräisin Toyotalta.

Käsityöläisyyttä voidaan pitää ensimmäisenä tuotantojärjestelmänä, joka pohjautuu käytännössä täysin yksilön osaamiseen ja kokemukseen. Se sopii parhaiten pienille ja yksinkertaisille tuotteille, jotka ovat yksilöitä, eli eroavat aina toisistaan.

Käsityöläisyyden jälkeen syntyi massatuotantomalli, jossa perusajatuksena on valmistaa tehokkaasti suuria määriä samanlaisia tuotteita. Massatuotantoparadigman kehittäjänä pidetään Henry Fordia. Massatuotannon yksi suurimpia ongelmia on sen joustamattomuus. Nykypäivän ohjelmistokehityksessä on tunnistettavissa monia piirteitä, jotka ovat peräisin massatuotantoajattelusta. Ei voida kuitenkaan olettaa, että massatuotantoajattelusta periytyneet toimintatavat olisivat luonnollisia tai parhaita tapoja valmistaa ohjelmistoja.

Massatuotannon jälkeen kehitettiin japanissa niin sanottu Lean-tuotanto, joka on myös peräisin autojen valmistuksesta. Lean-tuotanto pohjautuu Toyotan tuotantojärjestelmään, ja sen voidaan katsoa saaneen alkunsa 1920-luvulla. Lean-tuotantoon liittyy läheisesti juuri-oikeaan-tarpeeseen -ajattelu. Eroina massatuotantoon, Lean-tuotannolla pystytään valmistamaan pieniä eriä tehokkaasti, mikä mahdollistaa tuotteiden mukauttamisen tarpeen mukaan, ja Lean-tuotanto on myös massatuotantoa asiakaskeskeisempi.

3 Leanin perusteet

Lean pohjautuu tiettyihin periaatteisiin. Periaatteet ovat noudattajilleen totuuksia, jotka eivät muutu ajan myötä. Käytänteet taas ovat periaatteista johdettuja sovelluksia tiettyihin tilanteisiin. Käytänteet voivat, ja niiden tulee muuttua ympäristön ja tilanteiden muuttuessa. Periaatteet pitää kuitenkin ymmärtää, jotta voidaan luoda toimivat ja yhtenäiset käytänteet.

Lean on erittäin laaja aihe, joten siitä voi kuvata vain murto-osan tässä tutkielmassa. Leaniin ei myöskään ole olemassa yhtä ainoata näkemystä, vaan aihetta voidaan lähestyä usealla eri tavalla, mikä tekee sen käsittelystä ja ymmärtämisestä entistä haastavampaa. Eräs tapa lähestyä asiaa on ajatella Leaniin olevan kaksi eri näkökulmaa: länsimaalainen ja japanilainen. Vaikka nämä kaksi ovat ulkoisesti hyvin samankaltaisia, on niissä myös poikkeavuuksia toisiinsa nähden.

3.1 Leanin määritelmä

Lean on japanilaista alkuperää oleva ajattelutapa, filosofia ja elämäntyyli, jonka alkuperä on Toyotan tuotantojärjestelmässä. Leania on mahdoton määritellä muutamalla lauseella aihepiirin laajuuden vuoksi. Alla on kuitenkin joitain eri lähteistä olevia lainauksia, joissa pyritään kiteyttämään Leanin perusajatus:

- ”Ydinajatuksena on maksimoida asiakkaalle tuotettu lisäarvo minimoimalla hukka. Yksinkertaistettuna Lean tarkoittaa suuremman lisäarvon tuottamista asiakkaille vähemmillä resursseilla.” [Lea]
- ”Lean on mielentila ja ajattelutapa siitä, miten tuotetaan nopeammin lisäarvoa asiakkaalle etsimällä ja eliminoimalla hukkaa.” [HJS09, s.10]
- ”Hukan eliminointi on Leanin keskeisin periaate.” [PP03, s.2]

Sitaateista käy selkeästi ilmi, että lisäarvon tuottaminen asiakkaalle on Leanin olennaisin asia. Ensisijainen tapa päästä tähän on eliminoida hukkaa. Hukka on Leanin termi mille tahansa, mikä vie aikaa mutta ei tuota lisäarvoa asiakkaan kannalta [Lik10, s.10]. Hukkaa ja sen eri muotoja käsitellään tarkemmin myöhemmissä luvuissa.

Leania voidaan lähestyä joko länsimaalaisesta tai japanilaisesta näkökulmasta. Länsimaalaisten lähestymistapa Leaniin on analyyttinen ja japanilaisten holistinen. [MS05, s. xxiv–xxvii]

Useimpien länsimaalaisten käsitys Leanista perustuu Womackin ja Jonesin kirjoihin *The Machine That Changed World* (1990) [WJR07] ja *Lean Thinking* (1996) [WJ03]. Jälkimmäisessä kirjassa Leanista on yleistetty viisi perusperiaatetta: arvo, arvovirta, virtaus, imu ja täydellisyys. Nämä viisi ydinkonseptia ovat olleet keskeisessä roolissa, kun länsimaissa on siirrytty Lean-tuotantoon [MS05]. Kyseisiä prinsiippejä käsitellään tarkemmin myöhemmin.

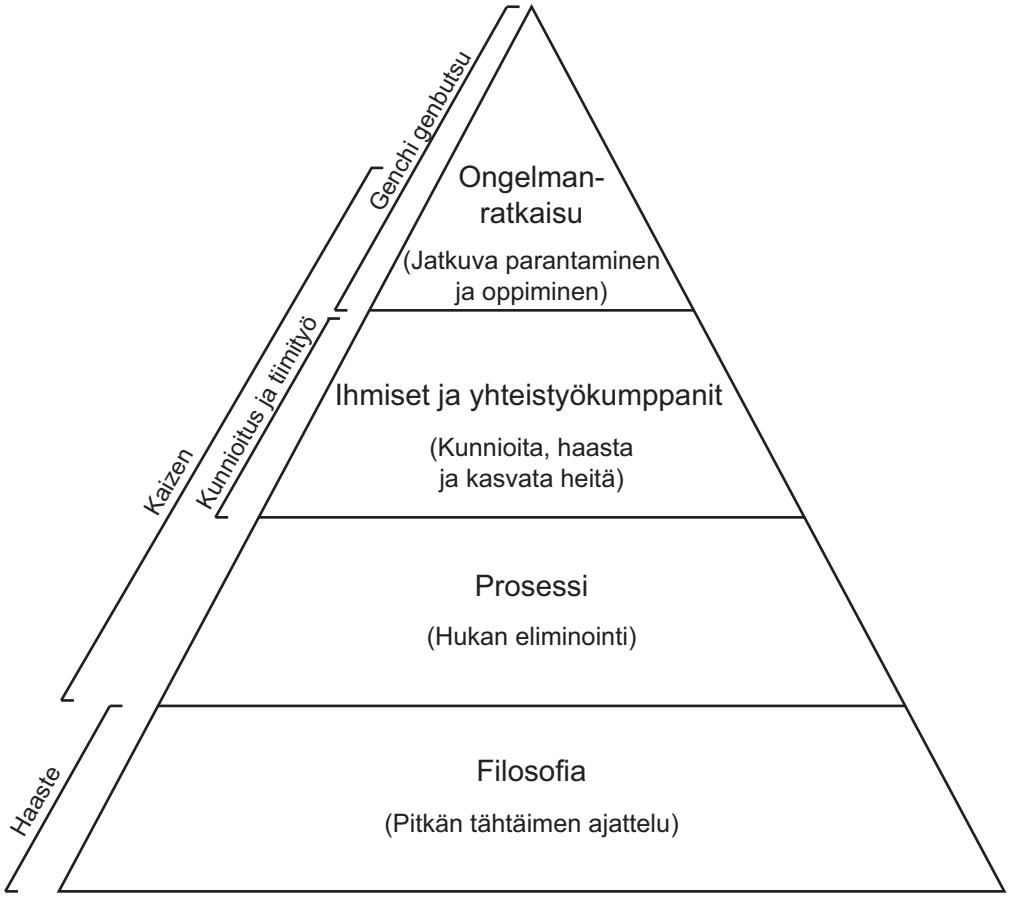
Siinä, missä länsimaalaiset lähestyvät Leania analyyttisesti, on japanilaisten lähestymistapa holistinen [MS05, s. xxv]. Heidän mukaansa yritykset esittää Lean muutamalla prinsiipillä yksinkertaistaa sitä liikaa, mikä voi johtaa väärinymmärryksiin [FC06, s. 2]. Leanin alkuperäisten japanilaisten harjoittajien mukaan Lean on ensisijaisesti tarkoituksen ja kulttuurin luomista. Heidän mukaansa Leania harjoittaakseen täytyy luoda kulttuuri, joka pohjautuu Leanin tarkoituksiin. Japanilaisten mukaan Leanin prioriteetit ovat 1) tarkoitus, 2) kulttuuri, 3) prinsiipit ja 4) tekniikat. Länsimaalainen näkemys ohittaa useimmiten näistä kaksi ensimmäistä, mikä japanilaisten mukaan voi johtaa väärään lopputulokseen. [MS05, s. xxiv–xxvi]

Seuraavissa aliluvuissa on kuvattu Likerin tunnistamia periaatteita sekä näihin liittyviä Lean-käytänteitä ja työkaluja. On kuitenkin huomioitava, että työkalut ja menetelmät eivät ole Leanin avain, vaan yhtiön johdon sitoutuminen jatkuvasti investoimaan työntekijöihin ja edistämään jatkuvaa parantamista [Lik10, s. 10]. Työkalut ja menetelmät ovat peräisin alun perin Toyotalta, ja ne eivät ole välttämättä sellaisenaan käyttökelpoisia esimerkiksi ohjelmistokehityksessä. Tämän takia on oleellisempaa ymmärtää menetelmien takana piilevät Leanin periaatteet, jotka ovat universaaleja ja sovellettavissa lähes mihin tahansa ympäristöön.

3.2 Japanilainen näkökulma

Liker [Lik04] on tunnistanut tutkimuksissaan 14 periaatetta, joita Toyota noudattaa. Nämä periaatteet on jaettu neljään pääkategoriaan: filosofia *(engl. philosophy)*, prosessi *(engl. process)*, ihmiset ja yhteistyökumppanit *(engl. people and partners)* sekä ongelmanratkaisu *(engl. problem solving)* [Lik04, s. 6]. Kategoriat ovat nähtävillä kuvassa 3.1. Kuvassa näkyvää mallia nimitetään 4P-malliksi, koska kategorioiden englanninkieliset nimet alkavat P:llä.

Likerin [Lik04, s. 6] mukaan Toyota itse kuvaa sisäisessä ”Toyota Way” -asiakirjassaan neljä pääperiaatetta, jotka ovat Genchi genbutsu (mene ja katso itse), Kaizen



Kuva 3.1: Toyotan tavan neljän periaateluokan malli [Lik10, s. 6]

(jatkuva parantaminen), Kunnioitus ja tiimityö, sekä Haaste. Nämä periaatteet on kuvattu hieman eri näkökulmasta, mutta käsittelevät samoja asioita. Liker on suhteuttanut kyseiset periaatteet omiin pääkategorioihinsa. Suhteet ilmenevät kuvan 3.1 vasemmasta reunasta.

3.2.1 Filosofia

Likerin mukaan useimmat yritykset puuhastelevat hänen kuvaamansa 4P-mallin (kuva 3.1) prosessitasolla unohtaen muut tasot kokonaan ja kutsuvat silti itseään Lean-yrityksiksi [Lik10, s.13]. Japanilaisen näkemyksen mukaan kaikki lähtee kuitenkin liikkeelle filosofiasta. Tämä on koko mallin perusta, kuten kuvassa 3.1.

Toyota Motor Manufacturing North America
MISSIO

1. Antaa panoksensa amerikkalaisena yrityksenä yhteiskunnan ja Yhdysvaltain talouskasvuun.
2. Antaa panoksensa itsenäisenä yhtiönä tiimin jäsenten vakauteen ja hyvinvointiin.
3. Toyota-konsernin jäsenenä antaa panos Toyotan kokonaiskasvuun tuottamalla lisäarvoa asiakkaille.

Kuva 3.2: Toyotan pohjoisamerikkalaisten operaatioiden toiminta-ajatus [Lik10, s.80]

Likerin ensimmäinen ja ainoa filosofiaan liittyvä periaate on ”Tee päätöksiä pitkän tähtäimen filosofian pohjalta, mutta myös lyhyen tähtäimen taloudellisten tavoitteiden kustannuksella”. Tämä tarkoittaa hänen mukaansa, että Toyotalla ei tehdä päätöksiä sen mukaan, että pystyttäisiin maksimoimaan vuosineljänneksen tulos. Sen sijaan hänen mukaansa viesti Toyotan sisällä on yhtenäinen: ”Tee sitä, mikä on oikein yhtiön, sen työntekijöiden, asiakkaiden ja yhteiskunnan kannalta” [Lik10, s.72]. Esimerkkinä tästä on, että myynnin väliaikaisesti laskiessa ei irtisanota työntekijöitä. Likerin mukaan Toyotan tavassa ihmisillä on tehtävä, joka on rahan ansaitsemista suurempi. Ideologiasta saa pienen esimerkin kuvasta 3.2, jossa on Toyotan pohjoisamerikkalaisten operaatioiden toiminta-ajatus *(engl. mission statement)*.

Toiminta-ajatuksessa ei esimerkiksi mainita osakkeenomistajia, mikä löytyy useiden muiden pörssiyritysten toiminta-ajatuksista.

Japanilaisen Lean-näkemyksen mukaan lyhytnäköisyyttä tulisi pyrkiä välttämään ja pitkän aikavälin ratkaisuihin tulisi investoida. Näin saavutetaan jatkuvuus ja voidaan kestävästi luoda lisäarvoa asiakkaille. [Lik10, s.71–84]

3.2.2 Prosessi

Japanilainen näkökulma prosessiin on ”Oikea prosessi tuottaa oikeat tulokset”. Ensisijainen lähestymistapa prosessien parantamiseen on hukan eliminointi. Toyota tunnisti alun perin seitsemän perustavaa laatua olevaa hukkaa [Lik04, s. 28–29][MS05, s. 29]:

- **Ylituotanto** *(engl. Overproduction)*: Tavaran tuottaminen ilman tilausta (liian aikaisin tai liian suuren erän tuottaminen), mikä aiheuttaa muita hukkia, kuten kuljetusta ja varastointia.
- **Odottaminen** *(engl. Waiting)*: Esimerkiksi materiaalien, osien, työkalujen tai prosessin edellisen vaiheen odottaminen, tuotannon pullonkaulat tai automaattisen koneen suorittaman toimenpiteen odottaminen. Kaikki odottaminen on käytännössä hukkaa ja lisäarvoa tuottamatonta.
- **Kuljetus** *(engl. Transportation)*: Keskeneräisen tuotteen tai materiaalien kuljettaminen eri työvaiheiden välillä sekä materiaalien, osien tai valmiiden tavaroiden siirto varastoon tai pois varastosta on pääsääntöisesti lisäarvoa tuottamatonta työtä.
- **Yliprosessointi** *(engl. Overprocessing)*: Tarpeettomien työvaiheiden tekeminen, huonosta suunnittelusta tai huonoista työkaluista johtuva prosessointi tai laadukkaampien tuotteiden tekeminen kuin olisi tarpeellista.
- **Varastointi** *(engl. Inventory)*: Esimerkiksi raakamateriaalin ja valmiiden tuotteiden varastointi tai keskeneräinen työ on hukkaa. Varastointi aiheuttaa muun muassa pitempiä läpimenoaikoja, vanhentumista, vahingoittuneita tuotteita, ylimääräistä kuljetusta, varastointikustannuksia ja viiveitä. Varastointi myös piilottaa ongelmia, kuten vaihteluita tuotannon tehokkuudessa, toimituspään myöhästymisiä, virheitä tuotteissa ja laitteiden vikoja.
- **Liike** *(engl. Motion)*: Kaikki ylimääräinen liike mitä työntekijät joutuvat tekemään. Esimerkiksi kävely ja työkalujen tai henkilöiden etsiminen ovat turhaa liikettä.

- **Virheet** *(engl. Defects)*: Virheellisten osien tai tuotteiden valmistus, tarkastus sekä virheiden korjaaminen. Korjaus on jo kerran tehdyn työn tekemistä uudelleen, mikä on hukkaa ja voi aiheuttaa muita hukkia, kuten odottamista. Myöskään laaduntarkastus ei sinällään tuota asiakkaalle lisäarvoa.

Taiichi Ohnon (Toyotan tuotantojärjestelmän perustaja) mukaan keskeisin hukka on ylituotanto, koska se aiheuttaa suurimman osan muista hukista [Lik04, s. 29]. Toyotan tunnistamat hukat ovat tuotantoon liittyviä. Luvussa 4 tarkastellaan, mitkä ovat ohjelmistokehityksen vastaavat hukat.

Liker on tunnistanut 4P-mallin (kuva 3.1) prosessitasolle liittyvän seitsemän periaatetta [Lik10, s.85]. Kyseiset periaatteet käydään alla ainoastaan lyhyesti läpi. Lukija voi halutessaan perehtyä niihin tarkemmin Jeffrey K. Likerin kirjassa Toyotan Tapaan. Prosesseihin liittyviä asioita käsitellään myös tämän tutkielman luvussa 3.3. Prosesseihin liittyvät periaatteet ovat:

- **Luo jatkuva prosessin virtaus tuodaksesi ongelmat esille.** Virtaus tarkoittaa, että kun asiakas käynnistää prosessin, eli tekee tilauksen, pitäisi koko prosessin tästä hetkestä toimitettuun tuotteeseen kestää viikkojen tai kuukausien sijaan muutaman tunnin tai päivän. Virtausta parannetaan poistamalla hukkaa. Hukan eri tyyppejä käytiin läpi aiemmin. Useimmissa prosesseissa 90% on hukkaa, esimerkiksi odottelua, ja 10% on lisäarvoa tuottavaa työtä. Lean-ajattelussa uskotaan, että raaka-aineista valmiisiin tuotteisiin (tai palveluihin) kuluvan ajan lyhentäminen johtaa parhaaseen laatuun, pienimpiin kustannuksiin ja luonnollisesti lyhimpään toimitusaikaan. Virtauksen parantaminen myös nostaa useimmiten esille prosessissa aiemmin piilossa olleita ongelmia, mikä katsotaan Leanissa hyväksi asiaksi.

- **Käytä imujärjestelmiä välttääksesi ylituotantoa.** Imuohjattu järjestelmä tarkoittaa, että annetaan asiakkaalle juuri se, mitä hän haluaa, silloin kun hän sitä haluaa ja juuri sen verran kuin hän haluaa. Asiakas voi tässä tapauksessa olla myös tuotantoprosessin seuraava vaihe. Ihanteellinen imujärjestelmä olisi niin sanottu nollavarastojärjestelmä, jossa eri työvaiheiden välillä ei ole ollenkaan varastoja. Käytännön elämässä tapahtuu kuitenkin luonnollisia katkoksia, joten puskureita tarvitaan. Näitä tulee kuitenkin olla mahdollisimman vähän ja niiden koko tulee minimoida. [Lik10, s.104–110]

  Reaalimaailman esimerkkinä imuohjauksesta voisi olla lähikauppa. Työntöohjauksessa *(engl. push system)*, joka on imuohjauksen vastakohta, tukusta toimitetaan kauppaan joka maanantai tietty määrä sokeria. Toimitetun sokerin

määrä ei riipu siitä, kuinka paljon kaupassa on kulloinkin sokeria jäljellä, vaan sopimuksen mukaisesti kauppa saa joka maanantai saman määrän sokeria. Imuohjauksessa kaupasta tiedotetaan sokerintoimittajalle milloin ja kuinka paljon sokeria tarvitaan. Näin asiakas (kauppa) saa sokeria juuri oikean määrän juuri oikeaan aikaan. Työntöohjaus on peräisin massatuotannosta ja imuohjaus Lean-tuotannosta. Imuohjauksella pyritään välttämään ylituotantoa, joka on hukkaa.

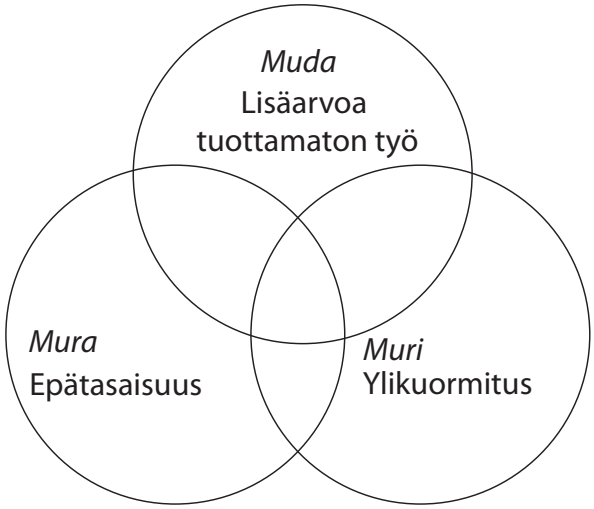
Eräs käytännön työkalu imuohjauksen toteuttamiseen on *kanban*. *Kan* on japania ja tarkoittaa korttia. *Ban* taas tarkoittaa signaalia. Kanban on siis suomeksi signaalikortti. Esimerkki signaalikortista on lentolippu. Se kertoo lentokentällä omistajalleen hänen lentonsa numeron, jonka perusteella hän osaa mennä oikeaan paikkaan. Kortit voivat olla fyysisiä tai sähköisiä. Kanban-työkalusta puhuttaessa tarkoitetaan usein järjestelyä, jossa jokin signaali (kanban) kertoo materiaalin tarpeesta. Esimerkiksi kokoonpanolinjan työntekijöiden alkaessa käyttää osia laatikosta, he ottavat laatikossa päällimmäisenä olevan kanban-kortin ja sijoittavat sen sovittuun, näkyvään paikkaan. Tietyin aikavälein varastotyöntekijä käy keräämässä kanban-kortit. Tämän perusteella varastosta toimitetaan kokoonpanolinjastolle uudet laatikolliset oikeita osia. Mikäli sama toteutettaisiin työntöohjauksena, toimitettaisiin linjastolle tietyin aikavälein aina saman verran kutakin osaa riippumatta siitä, kuinka paljon kyseisiä osia on käytetty. [PP08, s. 138] [Lik10, s.104–110]

Kanbanin haasteena ei ole se, että ihmiset eivät saisi selville, mitä tehdä seuraavaksi. Todellinen haaste on löytää keinot, joilla varmistua, että kanban-korttien sisältö on oikea. [PP08, s. 138]

- **Tasapainota työmäärää (heijunka).** Tuotannon tasapainottamisena tavoite on, että tuotannon määrä on aina mahdollisimman lähelle vakio. Tämä tarkoittaa, että tuotannossa ei ole suuria päivä- tai kuukausikohtaisia vaihteluita. Tasapainotettu tuotanto mahdollistaa mm. imuohjauksen käyttämisen.

  Kun puhutaan hukasta, tarkoitetaan yleensä lisäarvoa tuottamatonta työtä. Leanissa termi muda tarkoittaa nimenomaan tätä. Aiemmin esitellyt seitsemän hukan tyyppiä ovat mudaa. Leanissa on myös kaksi muuta hukkatyyppiä: muri ja mura (ks. kuva 3.3). Murilla tarkoitetaan ihmisten ja laitteiden ylikuormitusta, josta aiheutuu laatuongelmia, katkoksia ja vikoja. Muralla taas epätasaisuutta, jota voidaan ajatella mudan ja murin seurauksena. Epätasaisuus tarkoittaa, että käsillä pitää olla enemmän ihmisiä, työvälineitä ja materiaalia, mitä keskimääräiset vaatimukset todellisuudessa vaativat. Heijungalla

pyritään poistamaan muraa. [Lik10, s.114–115]



Kuva 3.3: Leanin m-termit [Lik10, s.114–115]

Heijunka on tuotannon tasoittamista sekä tuotantomäärien että tuotevalikoiman suhteen. Tuotteita valmistetaan siinä suhteessa kuin niitä keskimäärin tilataan. Esimerkiksi tuotteita A ja B voitaisiin valmistaa tuotantolinjastolla seuraavasti: A, B, B, A, B, B. Valmistus tehtäisiin näin jokaisena viikonpäivänä riippumatta siitä kuinka paljon esimerkiksi maanantaina tilataan tuotetta A. Todellisten tilausten mukaan valmistettaessa ongelmana on suuri epätasaisuus tuotannossa. Jos maanantaina tilauksia tulisi tuplamäärä normaalin nähden ja tiistaina ei tulisi yhtään tilausta, pitäisi maanantaina työntekijöille maksaa ylityökorvauksia ja tiistaina heidät pitäisi lähettää kotiin töiden puuttuessa. Tuotteita ei myöskään valmisteta suuria määriä varastoon. Ei esimerkiksi tehdä niin, että maanantaina valmistettaisiin pelkästään tuotetta A ja tiistaina tuotetta B. Tämä olisi tasapainottamaton tuotanto. [Lik10, s.113–127]

Heijunka sotii selkeästi ”tilauksen mukaan valmistamista” vastaan, koska tasapainotetussa tuotannossa asiakas voi joutua odottamaan. Toyota on kuitenkin osoittanut, että nämä eivät ole toisiaan poissulkevia periaatteita. Toyota Motor Salesin varatoimitusjohtaja Alan Cabito on todennut: *”Toyotan järjestelmä ei ole 'valmista tilauksen mukaan' -järjestelmä. Se on 'muutu tilauksen mukaan' -järjestelmä”*. Toyotalla on mahdollisuus valita hihnalta mikä tahansa auto ja muuttaa sitä tietyssä määrin. Toyota kykenee lennosta mukautumaan uusiin tilauksiin. [Lik10, s.113–127]

- **Luo kulttuuri, jossa pysähdytään korjaamaan ongelmia, jotta laatu saataisiin kuntoon heti ensimmäisellä kerralla (jidoka).** Lean-tuotanto korostaa

asioiden kuntoon saamisen tärkeyttä heti ensimmäisellä kerralla, koska ei ole suuria varastoja, joihin voitaisiin turvautua ongelmien esiintyessä. Ideana tässä periaatteessa on ongelman ilmetessä keskeyttää prosessi laadun sisäänrakentamiseksi. [Lik10, s.128–139]

Alkuperäisessä mallissa kone havaitsi automaattisesti vian ja ilmoitti siitä koneenkäyttäjälle. Menetelmän keksi Sakichi Toyoda vuonna 1902, kun hänellä oli tarve havaita virheitä automaattisissa kangaspuissa. Jidoka vapautti koneenkäyttäjän resursseja mahdollistaen hänen hoitaa useampaa konetta yhtä aikaa.

Jidoka on laadunvalvontaprosessi, jossa toimitaan seuraavan nelivaiheisen prosessin mukaisesti:

1. Havaitse vika.
2. Pysähdy.
3. Suorita välitön korjaustoimenpide.
4. Tutki perimmäinen syy ja aseta vastatoimet.

Etenkin viimeinen kohta on tärkeä. Vastatoimien asettamisella pyritään siihen, että vastaavaa vikaa ei enää synny. Lisäksi vastatoimet asetetaan perimmäiselle syylle, jolloin samalla eliminoidaan useiden vastaavanlaisten vikojen synty. Perimmäinen syy voidaan selvittää esimerkiksi käyttämällä 5 miksi-kysymystä -analyysiä. Tämä tarkoittaa, että kun kysytään ensimmäisen kerran ”Miksi ongelma ilmeni?”, ei tyydytä ensimmäiseen vastaukseen, vaan kysytään tämän jälkeen uudelleen ”Miksi?”. Tätä jatketaan viiden miksi-kysymyksen verran tai niin kauan, kunnes perimmäinen syy saadaan selville. [MS05] [Lik10, s.252–254]

Jidokaan liittyy läheisesti *Andon*. Sana tarkoittaa japanilaista bambukehyksestä ja paperista tehtyä lyhtyä. Toyota käytti kyseistä sanaa vivusta, jota vetämällä työntekijät pystyivät pysäyttämään tuotantolinjaston, ja tämä aiheutti yleensä valojen vilkkumisen huomion kiinnittämiseksi. Nykyisin andonilla tarkoitetaan mitä tahansa visuaalista viestintävälinettä tai näyttötaulua. Esimerkiksi rautatieasemilla olevat junien lähtö- ja saapumisaikoja näyttävät taulut ovat Lean-termein andon-tauluja. [PP08, s. 139–140]

- **Standardisoidut tehtävät ovat jatkuvan parantamisen ja työntekijöiden sitouttamisen perusta.** Tämän periaatteen ajatus on yksinkertainen: jos tehtävää tai prosessia ei ole standardoitu, ei ole mitään, mitä parannettaisiin. Ilman

standardointia on mahdollista, että tehtävä tehdään joka kerta eri tavalla. Standardointia voi ajatella parhaana vaihtoehtona, mikä nyt tiedetään, mutta se ei ole missään nimessä työskentelytapojen kiveen kirjoittamista. Standardointia tehdään nimenomaan sen takia, että tapoja tehdä asioita voidaan parantaa, mikä tarkoittaa, että standardit muuttuvat jatkuvasti. Työntekijät itse etsivät parhaita vaihtoehtoja suorittaa työtehtävänsä, mikä sitouttaa heitä työhönsä. Leaniin liittyy ajattelumalli, että päätökset tulee tehdä yrityshierarkiassa niin matalalla tasolla kuin mahdollista. [Lik10, s.140–148]

- **Käytä visuaalista ohjausta, jotta ongelmat eivät jää piiloon.** ”Visuaalinen ohjain on mikä tahansa työympäristössä käytetty viestintäväline, joka kertoo yhdellä silmäyksellä, kuinka työ pitäisi tehdä ja poikkeaakko se standardista.” Jos päällikkö voi kävellä työympäristön läpi ja yhdellä silmäyksellä havaita, noudatetaanko standardeja ja toimintaohjeita, sekä nähdä, onko poikkeamia tai ongelmia, toimii visuaalinen ohjaus. Esimerkiksi pajalla vasaran säilytyspaikan kohdalle voidaan maalata ”haamuvasara”, jolloin havaitaan, mikäli vasara puuttuu.

  Lean kehottaa myös käyttämään visuaalista ohjausta virtauksen parantamiseen. Tämä voi tarkoittaa esimerkiksi, että tavoitetuotantomäärä ja nykyinen tuotantomäärä näkyvät numeroina taululla. Nykyinen tuotantomäärä ei saisi olla tavoitetta pienempi eikä suurempi aiemmin läpikäydyn tuotannon tasapainottamisen (heijunka) vuoksi. [Lik10, s.149–158]

- **Käytä ainoastaan luotettavaa, perusteellisesti testattua teknologiaa, joka palvelee ihmisiä ja prosesseja.** Prosessia tulee ensin parantaa mahdollisimman paljon edellä kuvatuin ”manuaalisin” keinoin ennen kuin käytetään teknologiaa. Jos huonosti toimiva, ongelmia aiheuttava prosessi automatisoidaan, moninkertaistetaan ongelmien määrä. Lisäksi monimutkaisten prosessien automatisointi monimutkaistaa asioita entisestään.

  Uuden teknologian tulisi auttaa työntekijöitä suoriutumaan työtehtävästään paremmin. Teknologian käytöllä voidaan saavuttaa merkittäviä hyötyjä, mutta sitä on käytettävä harkitusti, ja teknologia on testattava huolellisesti ennen käyttöönottoa. Teknologian tulee tukea ihmisiä ja prosesseja eikä päinvastoin. Uudet teknologiat eivät saisi häiritä varsinaista työprosessia, jossa tuottavan työn tekeminen tapahtuu, vaan niiden pitäisi luontevasti sulautua osaksi prosessia. Uudet teknologiat eivät myöskään saa estää prosesseihin jatkuvan parannuksen myötä tulevien muutosten tekemistä. Lean-mallissa standardoidut työtehtävät ja prosessit ovat jatkuvan muutoksen kohteena. [Lik10, s.159–168]

3.2.3 Ihmiset ja yhteistyökumppanit

Ihmiset ja yhteistyökumppanit -tasolle liittyy 4P-mallissa (kuva 3.1) kolme periaatetta, jotka on käsitelty lyhyesti alla. Kyseisillä periaatteilla pyritään tuottamaan lisäarvoa organisaatiolle ihmisiä ja kumppaneita kehittämällä. [Lik10, s.169]

- **Kasvata johtajia, jotka ymmärtävät työn perusteellisesti, noudattavat filosofiaa ja opettavat sitä muille.** Toyotalle ei pääsääntöisesti palkata ihmisiä ulkopuolelta johtotehtäviin, vaan heidät kasvatetaan organisaation sisällä. Johtajien täytyy noudattaa ja ymmärtää Toyotan kulttuuria päivittäin. Leanin yksi tärkeä elementti on *genchi genbutsu*, joka tarkoittaa paikan päälle menemistä ja todellisen tilanteen selvittämistä ja ymmärtämistä. Tämän vuoksi johtajien tulee ymmärtää, miten todellista työtä ”lattiatasolla” tehdään, jotta he ymmärtävät, mitä todella tapahtuu. Toyotalla työntekijät aloittavat työnsä lattiatasolta, josta heillä on mahdollisuus edetä vähitellen urallaan eteenpäin. [Lik10, s.171–183]

- **Kehitä poikkeuksellisen eteviä ihmisiä ja ryhmiä, jotka noudattavat yhteistä filosofiaa.** Toyotalla työ tehdään tiimeissä. Siellä myös ymmärretään, että ryhmän kasvaminen itsenäiseksi, suorituskykyiseksi tiimiksi saattaa viedä jopa useita vuosia.

  Lean-ideologiassa työntekijät, jotka suorittavat lisäarvoa tuottavat työt, ovat arvokkaimpia. Kyseessä on siis tiiminjäsenet, eli lattiatason työntekijät. Nämä ihmiset ovat Lean-organisaatiossa hierarkian huipulla, koska organisaation tehtävä on tuottaa lisäarvoa asiakkailleen. Muun hierarkian eli johdon tehtävänä on tukea ja auttaa näitä työntekijöitä. Tiiminjäsenet ovat avainasemassa myös ongelmienratkaisussa, koska he tuntevat käytännön tilanteet johtoa paremmin. [Lik10, s.184–198]

- **Kunnioita yhteistyökumppaneilla ja alihankkijoilla laajennettua verkostoa tarjoamalla heille haasteita ja auttamalla heitä kehittymään.** Lean-organisaatiossa sen lisäksi, että itsellä on ”juuri oikeaan aikaan” -menetelmä ja muut Lean-käytänteet käytössä, täytyy myös alihankkijoille kouluttaa näitä käytänteitä. Muutoin tuotantoketju ei ole kokonaisuudessaan Lean. Ideana on kasvaa yhteistyössä kumppanien ja alihankkijoiden kanssa ja tavoitella pitkän tähtäimen hyötyjä. [Lik10, s.199–220]

Yllä esitetyistä periaatteista käy ilmi, että Lean on asiakaskeskeisyyden lisäksi myös työntekijäkeskeinen. Työntekijöiden ei oleteta olevan vaihdettavissa ilman menetyksiä, ja uudet johtajat kasvatetaan organisaation sisällä.

3.2.4 Ongelmanratkaisu

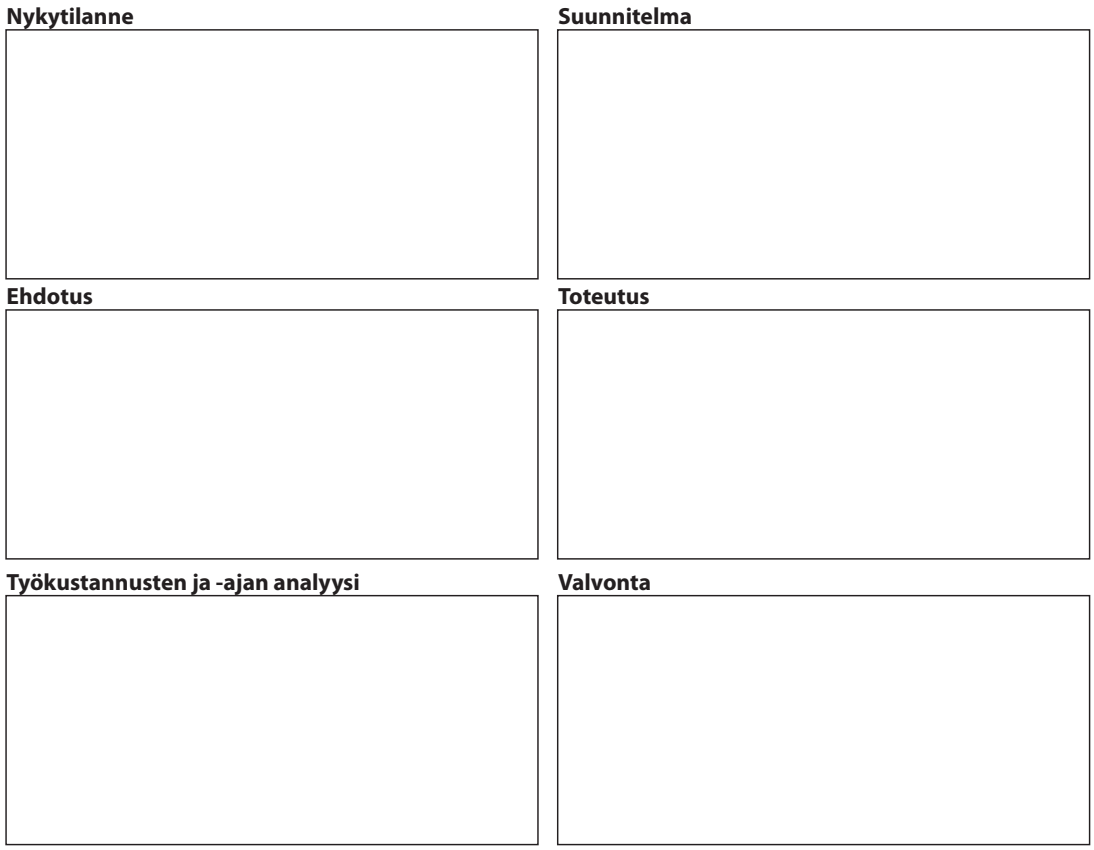
Ongelmanratkaisu on olennainen osa Leania. Siihen liittyy jatkuva parantaminen ja oppiminen. Liker on tunnistanut kolme ongelmanratkaisuun liittyvää periaatetta:

- **Mene itse paikan päälle katsomaan ymmärtääksesi tilanteen perusteellisesti (genchi genbutsu).** Genchi genbutsulla tarkoitetaan menemistä paikan päälle katsomaan omin silmin, jotta ymmärrettäisiin todellinen tilanne. Tätä pidetään Toyotan tavassa eräänä erona muihin johtamisjärjestelmiin. Asioita ei tule pitää itsestään selvänä tai turvautua muiden tekemiin raportteihin. Tarkoituksena on ymmärtää asiat syvällisesti ja tehdä päätökset vahvistetun informaation pohjalta. [Lik10, s.221–236]

- **Tee päätöksiä hitaasti yksimielisyyden pohjalta kaikkia vaihtoehtoja perusteellisesti harkiten; toteuta päätökset nopeasti.** Tämän periaatteen mukaan päätöksenteossa tulee käyttää perusteellista harkintaa. Vaihtoehtoisia ratkaisuja tulee harkita laajasti joukkopohjaisella menetelmällä. Perusteelliseen harkintaan liittyy vahvasti todellisen tilanteen ymmärtäminen eli edellä mainittu genchi genbutsu -periaate. Tämän pohjalta mietitään vaihtoehtoisia ratkaisuja ja parhaana pidetylle ratkaisulle tulee löytyä yksityiskohtainen perustelu.

  Tosielämässä ei ole aina mahdollista päästä paikanpäälle toteamaan itse tilannetta. Tällöin tarvitaan tehokasta viestintää. Eräs Toyotan kehittämä työkalu tähän ovat A3-raportit. Näiden raporttien ideana on sisällyttää kaikki tieto aina yhdelle A3:lle, oli kyse miten laajasta asiasta hyvänsä. Näin raportin laatija joutuu miettimään esitystapaa, ja turha teksti tulee karsittua pois. Tekstin sijaan käytetäänkin monesti kuvia ja graafeja havainnollistamaan asioita. Kun raportti on ainoastaan yhden sivun mittainen, on todennäköisempää, että useampi henkilö lukee sen läpi. Siihen on myös matalampi kynnys palata jälkeenpäin, kun yhdellä silmäyksellä näkee, mistä on kysymys. Kuvassa 3.4 on esimerkki A3-raportin pohjasta. [Lik10, s. 157,244], [PP08, s. 149–175]

- **Tee yrityksestäsi oppiva organisaatio väsymättömän arvioinnin (hansei) ja jatkuvan parantamisen (kaizen) kautta.** Kaizen on jatkuvaa parantamista. Kuten aiemmin on mainittu, jatkuvaa parantamista voi tapahtua vain sen jälkeen, kun prosessit ja toimintatavat ovat vakaat ja standardoidut. Kaizen on kuitenkin viimekädessä asenne ja ajattelutapa. Länsimaissa usein vältellään ja varotaan virheiden tekemistä. Jos sellainen tehdään, ei sitä helposti myönnetä, koska tätä pidetään monesti heikkouden merkkinä. Lean-ideologiassa virheitä pidetään mahdollisuutena oppia. Kaizeniin kuuluu ”minä olen vastuussa”



Kuva 3.4: Esimerkkipohja A3-raportille [Lik10, s.245]

-asenne. Kun yksittäinen työntekijä tuo ilmi pieleen menneen asian ja ehdottaa tähän vastatoimenpidettä, pidetään tätä suurena vahvuutena. Oppivassa organisaatiossa on kyse siitä, että jokainen osallistuu toimintatapojen parantamiseen. [Lik10, s.37–249]

Hansei tarkoittaa karkeasti ottaen arviointia. Se on osa japanilaista kulttuuria, ja siten länsimaalaisille hankala ymmärtää ja oppia. Japanissa vanhemmat saattavat sanoa lapselleen ”Tee hansei”, kun tämä on tehnyt jotain pahaa. Se tarkoittaa, että hänen täytyy katua ja parantaa käytöstään. Tähän sisältyy sekä henki että asenne. Lapsi ymmärtää kuitenkin täysin, mitä vanhemmat haluavat hänen tekevän. Liker kertoo, että George Yamashinan (Toyota Techical Centerin johtaja) mukaan ilman hanseita on mahdotonta saavuttaa kaizenia. *”Japanilaisessa hanseissa täytyy tuntea olonsa ensin todella surulliseksi, kun on tehnyt jotain väärää. Luodaan tulevaisuudensuunnitelma, jolla ongelma ratkaistaan, ja sen lisäksi pitää vilpittömästi uskoa, että ei enää koskaan tee samantyyppistä virhettä. Hansei on ajattelutapa, asenne.”*. Toyota on onnistuneesti juurruttanut hansein myös Amerikan maaperälle, joten asia on mahdollista sisäistää myös muissa kulttuureissa. [Lik10, s.250–265]

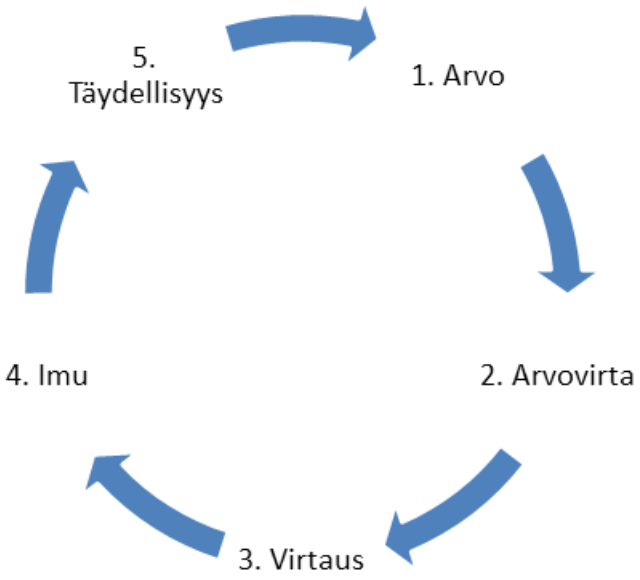
3.3 Länsimaalainen näkökulma

Länsimaalaisesta näkökulmasta puhuttaessa käytetään usein termiä Lean-ajattelu, mikä pohjautuu Womackin ja Jonesin Lean Thinking teokseen [WJ03]. Womack ja Jones ovatkin luoneet pohjan mallille, joka länsimaissa käsitetään Leanina.

Lean-ajattelun perustana ovat viisi ydinkonseptia: arvo, arvovirta, virtaus, imu ja täydellisyys [WJR07] (kuva 3.5). Kyseiset viisi ydinkonseptia ovat yleistyksiä Lean-tuotannosta ja Toyotan tuotantojärjestelmästä siten, että Leania voidaan soveltaa muuhunkin kuin valmistukseen ja tuotantoon.

3.3.1 Asiakkaan arvon määrittäminen

Lean-ajattelu lähtee liikkeelle arvon *(engl. value)* määrittelystä asiakkaan näkökulmasta käsin. Arvolla tarkoitetaan Leanissa niiden asioiden joukkoa, jotka ovat asiakkaalle merkityksellisiä [MS05, s. 23]. Arvolla ei siis tarkoiteta esimerkiksi tuotteen, hintaa vaan syytä, minkä takia se asiakkaan mielestä on hänelle hyödyllinen tai olemassa. Arvojen määrittäminen vastaa kysymykseen ”Miksi asiakas haluaisi tuotteen?”. Tämä ajattelutapa eroaa huomattavasti teknologiakeskeisestä ajattelutavasta, jossa keskitytään huipputeknologiaan ja jätetään asiakas vähemmälle huomiol-



Kuva 3.5: Lean ydinkonseptit länsimaalaisittain

le. Teknologiakeskeinen ajattelutapa juontaa juurensa länsimaalaiseen massatuotantoon, kun taas Lean-ajattelu on peräisin japanilaisesta Lean-tuotannosta. Lean-ajattelu on teknologiakeskeistä ajattelutapaa asiakaskeskeisempi ja ihmisläheisempi.

3.3.2 Arvovirran määrittäminen

Arvojen tunnistamisen jälkeen seuraava askel on kartoittaa arvovirta *(engl. value stream)*. Arvovirta on kaikkien niiden toimenpiteiden joukko, jotka tarvitaan tuotteen tai palvelun tuottamiseksi asiakkaalle [MS05, s. 26]. Arvo siis syntyy arvovirrassa. Arvovirta alkaa ja loppuu aina asiakkaaseen [PP08, s. 83]. Arvovirran määritys paljastaa normaalisti suuria määriä hukkaa *(engl. waste)*, eli aikaa vieviä, asiakkaalle lisäarvoa tuottamattomia toimenpiteitä, jotka voidaan karsia pois [WJR07, s. 19–20]. Järjestelmällinen hukan eliminointi on yksi Leanin keskeisimpiä asioita. Lean eroaa perinteisestä prosessinparannuksesta siten, että arvovirran kartoitus ohjaa optimoimaan kokonaisuutta sen sijaan, että tehtäisiin paikallisia parannuksia osaprosesseihin, jotka voivat olla kokonaisuuden kannalta merkityksettömiä. Osaprosessit voivat olla jopa kokonaan hukkaa, eli turhia työvaiheita, jolloin ne voidaan poistaa arvovirrasta.

3.3.3 Prosessin virtaus

Arvovirran tunnistamisen ja selkeästi turhien toimenpiteiden karsimisen jälkeen seuraava tavoite on saada jäljellejääneet toimenpiteet virtaamaan sujuvasti ja keskeytyksettä. Tämä tapahtuu poistamalla hukkaa. Kyse on siis täsmälleen samasta asiasta, jota käsiteltiin japanilaisen näkökulman yhteydessä luvussa 3.2.2. Kyseisessä luvussa esiteltiin myös hukan eri tyypit. Virtauksella pyritään optimoituun malliin, jossa prosessissa siirretään kerrallaan yksi osa eteenpäin *(engl. single piece flow)* suurten joukkojen sijaan. Myös välivarastot ja puskurit eri työvaiheiden välillä pyritään poistamaan, tai niiden koko minimoidaan, koska ne luokitellaan hukaksi, kuten edellä havaittiin.

3.3.4 Imuohjaus asiakkaasta taaksepäin

Imuohjattu järjestelmä *(engl. pull system)* tarkoittaa, että tehdään juuri se, mitä asiakas tarvitsee juuri silloin, kun hän sitä tarvitsee. Sama asia voidaan sanoa myös toisin: ei tehdä mitään ennen kuin asiakas tarvitsee sitä. Asiakkaalla tarkoitetaan tässä yhteydessä varsinaisen asiakkaan lisäksi myös seuraavan työvaiheen tekijää. Tästä käytetään nimitystä *just in time* -malli (JIT). Suomessa puhutaan joskus myös *juuri oikeaan tarpeeseen* -mallista (JOT). Imuohjauksen vastakohta on työntöohjaus. [WJR07, s. 24–25]

Imuohjaus löytyy myös japanilaisesta näkökulmasta, josta se on alun perin lähtöisin. Sitä käsiteltiin luvussa 3.2.2, josta löytyy myös käytännön esimerkit imu- ja työntöohjauksesta.

3.3.5 Täydellisyyden tavoittelu

Täydellisyysperiaatteella viitataan jatkuvaan parannukseen ja täydellisyyteen pyrkimiseen. Tästä käytetään japaninkielessä sanaa *kaizen* [WJR07, s. 25–28], joka tarkoittaa muutosta parempaan [PP08, s. 173]. Täydellistä Leania voidaan ajatella platonisena ideana arvovirrasta, jossa asiakas saa ilmaiseksi juuri haluamansalaisen tuotteen juuri silloin, kun sitä tarvitsee. Tätä ei tietenkään voida koskaan saavuttaa, mutta tavoitteena on pyrkiä koko ajan lähemmäksi sitä. Aina voidaan kehittää arvovirtaa toimivammaksi, löytää ja poistaa hukkaa prosesseista ja keksiä uusia tapoja tehostaa virtausta. Näin saadaan tuotettua asiakkaalle enemmän arvoa vähemmillä resursseilla.

Kaizen-työpajat ovat tunnettu työkalu, jossa organisaation eri osa-alueiden edustajat kokoontuvat yhteen muutamasta päivästä viikkoon ratkaisemaan tarkasti ra-

jattuja, kriittisiä ongelmia. Kaizen-työpajojen aikana kaizen-työryhmä tekee kaikki tarvittavat muutokset organisaation prosesseihin, jotta ongelma saadaan ratkaistua. Työpajan jälkeen muutokset otetaan välittömästi käyttöön, ja kaizen-työryhmä arvioi niiden toteuttamista viikoittain. [PP08, s. 173], [Lik10, s. 275–284]

Täydellisyyteen pyrkimiseen liittyy Lean-tuotannosta peräisin oleva laadunparannuskonsepti *jidoka*, jolla tarkoitetaan ihmisavusteista automaatiota. Jidokaa käsiteltiin luvussa 3.2.2.

Yksi tiedon suurimpia ongelmia on, että unohdetaan helposti, miksi joku päätös on tehty. Ei siis tiedetä kaikkea opittua. Usein ajatellaan, että tieto on tallessa, kun se on kirjoitettu ylös. Käytännössä kirjoitetut muistiot ja sähköpostit usein hautautuvat jonnekin, eikä kukaan lue niitä enää jälkeenpäin. Täydellisyyden tavoittelussa oleellista on kuitenkin tietää ja muistaa organisaation oppimat asiat. Leanin aputyökalu tähän ovat A3-raportit jotka esiteltiin aiemmin luvussa 3.2.4. Kun tieto on tiivistetyssä muodossa, on siihen helppo palata jälkikäteen. [PP08, s. 149–175]

3.4 Länsimaalaisen näkökulman suhde japanilaiseen näkökulmaan

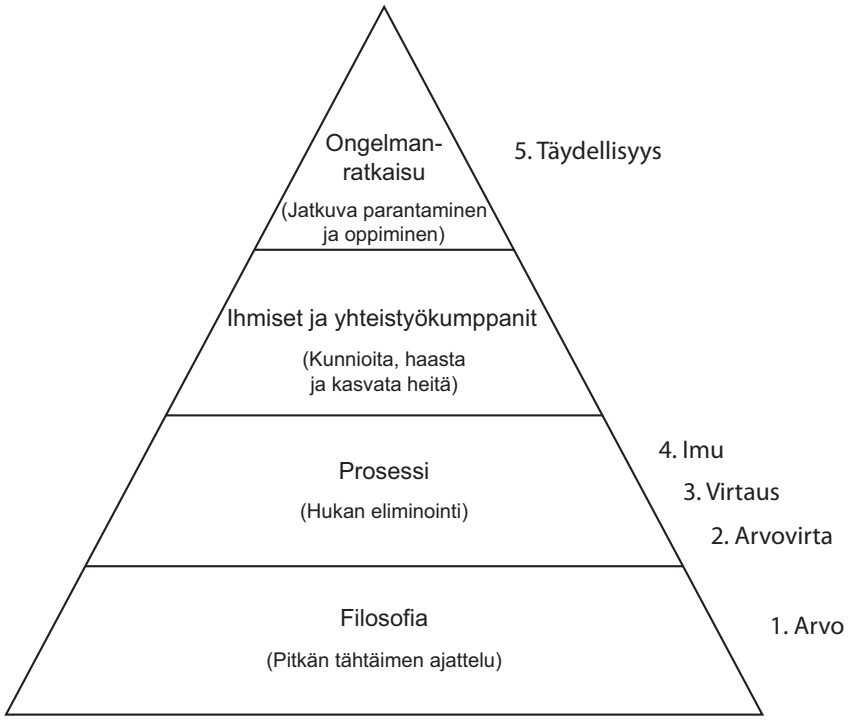
Kuvaan 3.6 on hahmoteltu tutkijan oma näkemys kuinka länsimaalainen käsitys Leanista suhtautuu Likerin 4P-malliin, joka kuvaa Toyotan periaatteita.

Kuvasta huomataan, että länsimaalainen näkökulma on hyvin prosessikeskeinen. *Arvovirta*, *virtaus* ja *imu* liittyvät suoraan prosessien kehittämiseen. Likerin mallissa, jota käsiteltiin tarkemmin luvussa 3.2.2, löytyy hyvinkin samanlaisia periaatteita:

- Luo prosessin ”virtaus”, jotta ongelmat tulisivat esille.
- Käytä imuohjausta ylituotannon välttämiseksi.

Länsimaalainen malli ohittaa filosofia-kerroksen lähes tyystin. Ainoastaan arvo sijoittuu tälle tasolle. Arvo-perusperiaatteessa ei kuitenkaan sinällään puhuta pitkän tähtäimen ajattelusta, vaikkakin pitkäkestoisten asiakassuhteiden muodostamiseen edellytetään tavallisesti pystyttävän tuottamaan jatkuvasti lisäarvoa asiakkaille. Arvo-perusperiaate vastannee pientä siivua 4P-mallin filosofia-kerroksesta.

Huomionarvoista on, että yksikään länsimaalaisen Lean-ajattelun perusperiaatteista ei sijoitu 4P-mallin *ihmiset ja yhteistyökumppanit* -tasolle. Tämä on mielenkiintoista erityisesti ohjelmistotuotannon kannalta, sillä esimerkiksi McConnell on todennut, että ”henkilöstöasiat vaikuttavat ohjelmistotuottavuuteen ja ohjelmiston laatuun enemmän kuin mikään muu asia” [McC02, s.12]. Tom DeMarco ja Timothy Lister ovat myös tutkineet, että ihmiset mainitsevat projektien epäonnistumisen syyksi



Kuva 3.6: Länsimaalaisten Lean-perusperiaatteiden suhde 4P-malliin

useimmiten ”politiikan” [DL99]. DeMarcon ja Listerin mukaan tällä termillä viitataan kaikkeen ihmisiin ja ihmisten välisiin suhteisiin liittyviin asioihin.

4P-mallin ylin taso, eli *ongelmanratkaisu*, ei periaatteiltaan eroa Leanin länsimaisessa ja japanilaisessa näkökulmassa. Länsimaisessa näkemyksessä jätetään tosin usein luvussa 3.2.4 esitelty kaikaku pois, mikä johtunee sen vahvoista sidoksista japanilaiseen kulttuuriin. Perusperiaatteeltaan molemmissa näkökulmissa pyritään kuitenkin erinomaisuuteen jatkuvalla parannuksella, eli kaizenilla.

Japanilainen ja länsimaalainen näkemys Leanista puhuu siis täysin samasta asiasta, mutta länsimaista näkemystä voidaan mielestäni pitää japanilaisen näkökulman osajoukkona, koska se ei tuo mukanaan mitään, mitä japanilainen näkökulma ei sisältäisi. Länsimaalainen näkemys on huomattavasti japanilaista vastinettaan prosessikeskeisempi. Lisäksi länsimaalaisen näkemyksen esitystapa (ks. kuva 3.5) vastaa kysymykseen ”Miten?”, kun japanilainen esitystapa kuvaa pikemminkin ”Mitä?”. Tämä heijastaa kansojen kulttuurieroja: Japanissa lähdetään liikkeelle filosofiasta, länsimaissa tekemisen kautta.

3.5 Yhteenveto

Lean on japanilaista alkuperää oleva ajattelutapa, filosofia ja elämäntyyli, joka sisältää tiettyjä periaatteita. Periaatteet ovat noudattajalleen totuuksia, jotka eivät muutu ajan myötä. Periaatteista voidaan johtaa käytänteitä, jotka voivat ja joiden tuleekin muuttua olosuhteiden muuttuessa. Leanin ydinajatuksena on maksimoida asiakkaalle tuotettu lisäarvo minimoimalla hukka. Leaniin on olemassa kaksi eri lähestymistapaa: japanilainen ja länsimaalainen.

Japanilaiseen lähestymistapaan liittyy neljä pääperiaatetta: ongelmanratkaisu, ihmiset ja yhteistyökumppanit, prosessi sekä filosofia. Näkökulman perusajatuksena on, että kaikki lähtee liikkeelle filosofiasta. Kaikki pääperiaatteet tulee kuitenkin ottaa huomioon.

Länsimaalainen näkökulma on japanilaista prosessikeskeisempi. Sen perustana ovat viisi ydinkonseptia: arvo, arvovirta, virtaus, imu ja täydellisyys. Nämä ydinkonseptit ovat yleistyksiä Lean-tuotannosta ja Toyotan tuotantojärjestelmästä siten, että Leania voidaan käyttää muuhunkin kuin valmistukseen ja tuotantoon.

Japanilainen ja länsimaalainen näkemys Leanista puhuvat samasta asiasta, mutta länsimaalaista voidaan pitää tietyllä tapaa japanilaisen näkemyksen osajoukkona. Voidaan ajatella, että länsimaalainen lähestymistapa vastaa kysymykseen ”Miten?”, kun japanilainen lähestymistapa vastaa kysymykseen ”Mitä?”

4 Lean-ohjelmistokehitys

Lean-ohjelmistokehitys on Leanin soveltamista ohjelmistokehitykseen. Tässä tutkielmassa Lean-ohjelmistokehityksen lähtökohtana käytetään pääasiassa Womackin ja Jonesin [WJ03] länsimaista, analyyttista näkemystä Leanista. Perusteluina tähän ovat kyseisen lähestymistavan yleisyys länsimaissa ja Lean-kirjallisuudessa. Tämä lähestymistapa on myös tapausyrityksessä tutumpi kuin japanilainen lähestymistapa. Leanin soveltamista ohjelmistokehitykseen pohdittaessa yrityskulttuuri ei olisi mahtunut tämän tutkielman piiriin. Lean-ajattelulla tarkoitetaan tässä tutkielmassa jatkossa Womackin ja Jonesin viiteen perusperiaatteeseen pohjautuvaa länsimaista ajattelutapaa.

Ohjelmistojen kehitys poikkeaa huomattavasti autojen valmistuksesta, josta Lean on lähtöisin, joten valmiita toimintatapoja ei voida kopioida esimerkiksi Toyotalta. Sen sijaan Leania täytyy luovasti soveltaa omaan ympäristöön ja kulttuuriin sopivaksi, jolloin saavutetaan paras mahdollinen lopputulos. Monet ohjelmistoyritykset ovat tehneet virheen yrittäessään omaksua käytänteitä suoraan Lean-tuotannosta vailla ymmärrystä, että tuotanto on eri asia kuin kehitys. Kehityksessä muutetaan idea valmiiksi tuotteeksi. Tuotannossa valmista tuotetta monistetaan. Näiden erilaisuutta voi verrata reseptin tekemiseen ja ruoan valmistukseen kyseisellä reseptillä. [PP03, s. 15–16]

Tuotannon puolelta voi olla joitain asioita kuitenkin siirrettävissä ohjelmistokehitykseen. Toyota kehitti Lean-tuotannon alun perin pienien erien tuottamista varten ja ohjelmistokehityksessä tuotetaan ainoastaan pieniä eriä (usein vain yksi kappale) [MS05, s. 93].

Toyotalla on tuotantojärjestelmänsä lisäksi olemassa myös tuotekehitysjärjestelmä *(Toyota product development system)*, jota käytetään uusien automallien kehitykseen. Uusien automallien kehitys muistuttaa kaaosmaisuudeltaan uuden ohjelmiston kehitystä. Näin ollen myös Toyotan tuotekehitysjärjestelmästä voi hakea viitteitä, kuinka Leania sovelletaan kehitykseen.

Monet yritykset ovat kopioineet Toyotalta joitain käytänteitä tai työkaluja, mutta ovat saavuttaneet vain keskinkertaisia tuloksia. Tämä johtunee siitä, että yrityksillä on prosesseissaan paljon hukkaa, koska he eivät ole omaksuneet Lean-ajattelua. Samalla tavalla ohjelmistoyritykset voivat ottaa käyttöönsä uusia kehityskäytänteitä, mutta jos ne eivät omaa syvempää ymmärrystä, miksi asiat tehdään tietyllä tavalla,

eivät ne saavuta maksimaalisia tuloksia. [PP08, s. 20]

Lean-ohjelmistokehityksen tunnetuimmat vaikuttajat ovat Mary ja Tom Poppendieck [HJS09, s. 16]. He toivat vuonna 2003 kirjassaan [PP03] *Lean Software Development: An Agile Toolkit* esille oman näkemyksensä Leanin soveltamisesta ohjelmistoalalla ja tarkensivat näkemystään vuonna 2006 julkaistussa teoksessaan [PP08] *Implementing Lean Software Development: From Concept to Cash*. Poppendieckit esittivät teoksissaan seitsemän ohjelmistokehitykseen soveltuvaa perusperiaatetta:

- poista hukka (eliminate waste)
- rakenna laatu ohjelman sisään (build quality in)
- luo tietoa (create knowledge)
- lykkää sitoutumista (defer commitment)
- toimita nopeasti (deliver fast)
- kunnioita ihmisiä (respect people)
- optimoi kokonaisuus (optimize the whole)

Poppendieckien tunnistamissa periaatteissa on selkeästi vaikutteita sekä Womackin ja Jonesin [WJ03] Lean-ajattelusta että Likerin [Lik04] japanilaisemmasta näkökulmasta. Useimmat Lean-ohjelmistokehityksestä kirjoittavat ovat seuranneet Poppendieckien periaatteita [HJS09, s. 16].

Perinteisempi tapa lähestyä Leania on soveltaa suoraan Womackin ja Jonesin viittä Lean-ajattelun perusperiaatetta omaan ympäristöön toimialasta riippumatta. Peter Middleton ja James Sutton noudattavat Lean-ohjelmistokehityksessä tätä lähestymistapaa teoksessaan *Lean Software Strategies* [MS05].

Kuten havaitaan, Lean-ohjelmistokehitykseen ei ole olemassa yhtä ainoaa ja oikeaa lähestymistapaa. Tässä tutkielmassa Lean-ohjelmistokehitystä tarkastellaan sekä Womackin ja Jonesin Lean-ajattelusta liikkeelle lähtevän mallin että Poppendieckien Lean-ohjelmistokehitysmallin mukaan. Nämä eivät ole toisiaan poissulkevia, kun muistaa Poppendieckien mallin olevan vain yksi lähestymistapa Lean-ohjelmistokehitykseen. Womackin ja Jonesin Lean-ajattelun viisi ydinkonseptia ovat yleisesti tunnetumpia kuin Poppendieckien ohjelmistoalalle tunnistetut periaatteet, ja heidän periaatteiden toimivuudesta on vahvaa näyttöä erittäin monelta toimialalta, joten heidän ydinkonseptejaan ei tule unohtaa tarkasteltaessa Lean-ohjelmistokehitystä. Lisäksi Leanin soveltaminen ohjelmistokehitykseen, kuten kaikille muillekin aloille, tulee aloittaa Lean-ydinkonseptien ymmärtämisestä. Vaikka Poppendieckien

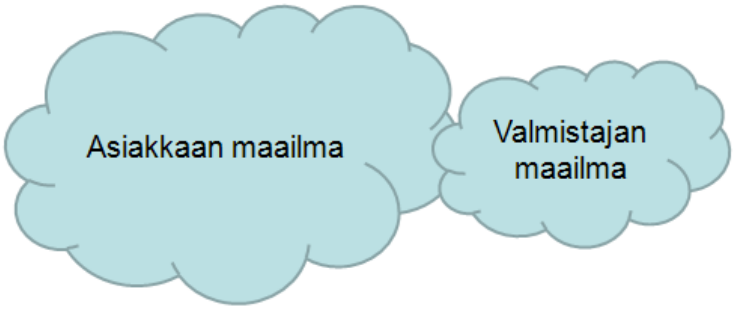
tunnistamat periaatteet ovat todennäköisesti hyviä ja toimivia, on mahdollista, että ne ovat yhdelle toimialalle sovellettuja versioita. Tällöin pelkästään niitä seuraamalla syvempi ymmärrys Leanista voi jäädä saavuttamatta, mikä taas voi olla riski muuttuvassa maailmassa.

Koska pyörää ei kannata keksiä uudelleen, hyvä lähestymistapa lähteä soveltamaan Leania ohjelmistokehitykseen on tarkastella olemassa olevia kehityskäytäntöjä ja -prosesseja ja tutkia, mitkä niistä ovat Lean-ajattelun mukaisia. Tämä on eräs tapa yhdistää Lean-ohjelmistokehitys ja ketterä ohjelmistokehitys [Wan11] [PP08] [PP03]. Ohjelmiston elinkaarimalli voidaan laatia Lean-ajattelun ja valittujen käytänteiden pohjalta, ja sitä tulee parantaa jatkuvasti. Valmiiden käytänteiden hyödyntämisen vuoksi Lean-ohjelmistokehitys muistuttaa ulkoisesti hyvin paljon ketteriä menetelmiä. Molemmissa esimerkiksi pyritään tekemään nopeasti näkyvää ja välttämään asiakkaalle lisäarvoa tuottamatonta työtä, kuten ylimääräistä dokumentointia. Molempia tehdään myös usein iteratiivisesti. Lean-ohjelmistokehitystä ja ketterää ohjelmistokehitystä ei kuitenkaan tule sekoittaa keskenään. Lean voidaan nähdä enemmänkin filosofiana ja ketterä kehitys käytännönläheisempänä mallina [Wan11].

4.1 Prosessi

4.1.1 Arvoselvitys

Lean-ajattelu lähtee liikkeelle arvosta. Muut ydinkonseptit ovat arvovirta, virtaus, imu ja täydellisyys. Arvon määritys asiakkaan näkökulmasta tarkoittaa ohjelmistokehityksessä arvoselvityksen tekemistä.



Kuva 4.1: Asiakkaan maailma ja valmistajan maailma

Voidaan ajatella, että on olemassa kaksi käsitteellistä maailmaa: asiakkaan maailma ja valmistajan maailma (kuva 4.1). Valmistajan maailma on se, jonka ohjelmis-

tokehittäjät tuntevat hyvin ja asiakas tuntee huonosti. Asiakkaan maailma on taas se toimiala ja käsitteiden joukko, joka on asiakkaalle tuttu, mutta ohjelmistokehittäjille vieras. Perinteiset menetelmät ja mallit keskittyvät lähes yksinomaan valmistajan maailmaan. Yritys voi olla esimerkiksi CMM-mallin tasolla viisi ja toteuttaa projektin erittäin tarkasti määriteltyä prosessia noudattaen sekä budjetissa että aikataulussa pysyen. CMM ei kuitenkaan ota juurikaan kantaa asiakkaan maailmaan. Tästä voi seurata se, että asiakkaan tarpeita ei ole ymmärretty, ja toimitettu ohjelmisto tekee vääriä asioita tai ratkaisee vääriä ongelmia. Näin ollen asiakkaan maailmaa voidaan pitää lopputuloksen kannalta merkittävämpänä kuin valmistajan maailmaa. Lean-ohjelmistokehitys on asiakaskeskeistä, ja tarkoituksena on oppia tuntemaan asiakkaan maailma mahdollisimman hyvin ennen kuin kehitys aloitetaan, jotta edellä mainitun kaltaisilta ongelmilta vältyttäisiin. Tämä tapahtuu tekemällä niin sanottu arvoselvitys. [MS05]

Arvoselvitys vastaa kysymyksiin ”Mitä?”, ”Milloin?” ja ”Miksi?”. Huomionarvoista on, että asiakkaan arvot eivät vastaa kysymykseen ”Miten?”, johon vaatimusmäärittely saattaa monesti pureutua. Asiakkaan maailmassa toteutustavalla ei ole yleensä suurta merkitystä, vaan toteutus liittyy valmistajan maailmaan. Arvot listataan asiakkaan käyttämillä termeillä, ja asiakkaan tulee ymmärtää ne täysin. Arvoselvityksen tuotoksena voidaan käyttää esimerkiksi priorisoitua käyttäjäkertomuslistaa. Esimerkkikäyttäjäkertomus voisi olla *”Asiakkaana haluan maksaa ostokseni verkkopankkitunnuksilla, koska minulla ei ole luottokorttia”*. Käyttäjäkertomukset ovat XP-prosessimallin tapa listata asiakasvaatimuksia, mutta ne sopivat hyvin myös arvoselvitykseen, koska ne priorisoituina vastaavat vaadittuihin kysymyksiin eivätkä ota kantaa toteutustapaan. Käyttäjäkertomuksia laadittaessa tulee kiinnittää huomiota, että ne liittyvät asiakkaan eikä valmistajan maailmaan. Esimerkki huonosta käyttäjäkertomuksesta olisi *”Asiakkaana haluan valita maksutavan hiirellä pudotusvalikosta, koska en voi maksaa luottokortilla”*. Pudotusvalikko on jo toteutustapaa, eli se kuuluu valmistajan maailmaan, eikä sen siten tule esiintyä listatuissa asiakkaan arvoissa. Hyvä käyttäjäkertomus on tyypillisesti kooltaan sellainen, että sen toteutus kestää puolesta päivästä neljään työpäivään. [MS05], [PP08, s. 185]

Arvoselvityksessä asiakkaan maailmaa tarkastellaan kolmesta eri perspektiivistä [MS05, s. 104]:

1. **Toimiala:** Arvot, jotka ovat yhteisiä tietyn tyyppisille asiakkaille. Toisin sanoen vastaavat kysymykseen ”Mitä kaikki samanlaiset asiakkaat haluavat?”.

2. **Tiedostetut erityisarvot:** Arvot, jotka ovat erityisiä juuri tietylle asiakkaalle. Nämä arvot vastaavat kysymykseen ”Mitä juuri tämä asiakas tietää haluavan-

sa?”.

3. **Tiedostamattomat erityisarvot:** Arvot, jotka ovat erityisiä juuri tietylle asiakkaalle, mutta joista hän ei itse ole tietoinen. Nämä vastaavat kysymykseen ”Mitä juuri tämä asiakas tarvitsee, mutta ei tiedä haluavansa?”.

Ensimmäinen perspektiivi auttaa luomaan paremman ymmärryksen asiakkaan toimialasta, ja luomaan sitä kautta enemmän arvoa asiakkaalle. Samalla pystytään lisäämään ohjelmiston monistettavuutta, mikäli samalle toimialalle tehdään useita projekteja. Arvojen määritys tästä perspektiivistä tarvitse tehdä pääsääntöisesti vain kerran yhdelle toimialalle. Toisessa saman toimialan projektissa voidaan hyödyntää ensimmäisessä projektissa tunnistettuja arvoja.

Toinen perspektiivi on kolmikosta tutuin. Se on ainoa perspektiivi, jota tavallisesti perinteisessä vaatimusmäärittelyssä käytetään.

Kolmas perspektiivi on vaativin, mutta sillä voidaan ylittää asiakkaan odotukset ja saavuttaa siten maksimaalinen asiakastyytyväisyys. Tästä, sekä ensimmäisestä perspektiivistä, käy myös ilmi miksi Lean-ohjelmistokehitystä tulisi kutsua asiakaskeskeiseksi eikä asiakaslähtöiseksi. Ohjelmistoihin toteutetaan myös ominaisuuksia, jotka eivät ole lähtöisin asiakkaalta, mutta joita asiakas tarvitsee.

”Käsityöläis- ja massatuotanto-ohjelmistokehittäjät tekevät vaatimusmäärittelyitä [MS05, s. 104].” Sekä arvoselvitys että vaatimusmäärittely kuvaavat asioita joihin kehittäjien täytyy kiinnittää huomiota. Nämä eroavat kuitenkin toisistaan, kuten nähdään taulukosta 4.1.

Taulukko 4.1: Arvoselvitys vs. vaatimusmäärittely [MS05, s. 105]

	Arvoselvitys	**Vaatimusmäärittely**
Painopiste	Asiakkaan tavoitteet	Tuotteen toiminnallisuudet
Uudelleen-käytettävyys	Tuoteperheeseen / tulevaisuuteen keskittynyt	Yhteen tuotteeseen / nykyhetkeen tai menneisyyteen keskittynyt
Suurimmat huolet	Asiakastyytyväisyys, liiketoiminta	Organisaation ”realiteetit”
Tarkkuus	”Mitä” ja ”Milloin”	”Mitä”, ”Milloin” ja ”Miten”
Järjestelmällisyys	Järjestelmällinen	Hajanainen
Asiakkaan osallistuminen	Keskeinen	Toisarvoinen
Vastuulliset osapuolet	Kehittäjät	”Ylemmän tason ryhmä”
Tulosten ”rikkaus”	Toimiala, tiedostetut, tiedostamattomat, priorisoitu	Yleensä vain tiedostetut

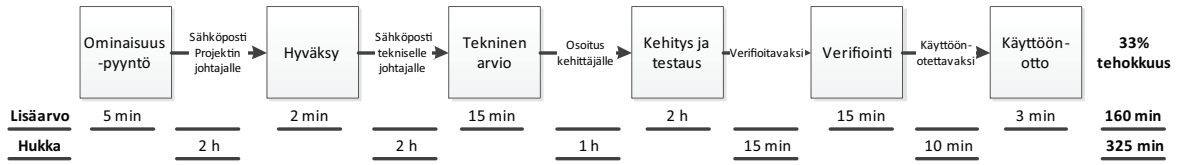
4.1.2 Arvovirta

Arvovirran kartoituksessa on yksinkertaistettuna kyse kahden pisteen välisten työvaiheiden tunnistamisesta sekä niiden vaatiman ajan mittaamisesta. Alun perin aloituspisteenä on ollut tilauksen saapuminen asiakkaalta ja lopetuspisteenä valmiin tuotteen toimitus asiakkaalle.

Ohjelmistokehityksessäkin arvovirran kartoitus aloitetaan valitsemalla kartoitettavan ketjun aloituspiste ja lopetuspiste. Nämä ovat ne hetket, jolloin ajanotto aloitetaan ja lopetetaan. Paras tulos saavutetaan valitsemalla mahdollisimman aikainen aloituspiste, jolloin optimoidaan kokonaisuutta ja vältytään mikrotason optimoinnilta. Organisaatiossa, jossa tehdään ohjelmistoprojekteja tilaustyönä, hyvä aloituspiste on asiakkaalta tuleva tilaus. Se, mitä tilaus tarkalleen ottaen tarkoittaa, vaihtelee organisaatioittain. Vastaavasti hyvä päätepiste tällaisessa organisaatiossa on käyttöönotetun ohjelmiston hyväksyntä asiakkaalta. [PP08, s. 83–93]

Arvovirtakarttoja voi olla eritasoisia riippuen käytetyistä aloitus- ja lopetuspisteistä. Käytännössä on usein helpompi lähteä liikkeelle pienemmästä ketjusta, jolloin ymmärrys organisaation prosesseista ja niiden suhteista asiakkaaseen kasvaa. Tästä voidaan helposti siirtyä myöhemmin laajempiin ketjuihin. Aloituspiste voi ol-

la esimerkiksi ominaisuuspyynnön saapuminen asiakkaalta ja päätepiste pyydetyn ominaisuuden käyttöönotto tuotantoon, kuten kuvan 4.2 arvovirtakartassa.



Kuva 4.2: Korkean prioriteetin ominaisuuspyynnön arvovirta pienessä yrityksessä [PP08, s. 86]

Kun aloitus- ja lopetuspisteet on päätetty, tunnistetaan nykyinen prosessi näiden välillä. Sopiva tarkkuus tähän on yleensä noin kymmenen työvaihetta [PP08, s. 83–93]. Työvaiheiden tunnistuksen jälkeen kellotetaan eri vaiheisiin käytetty aika.

Kuvassa 4.2 on Poppendieckien [PP08, s. 86] laatima esimerkki arvovirtakartasta. Kuvan alareunassa näkyy arvon tuottamiseen käytetty aika (value) ja hukaksi laskettava aika (waste). Kyseinen notaatio arvovirtakartalle on vain yksi monista mahdollisista. Notaation voi valita vapaasti itselle sopivaksi. Pääpaino tulee kuitenkin olla helppolukuisuudessa, koska arvovirtakartan tekemistä tärkeämpää on sen tulkitseminen. Kun arvovirta on tunnistettu, poistetaan tai optimoidaan selkeästi turhat toiminnot, ja etsitään tapoja poistaa hukkaa, jota on yleensä havaittavissa arvovirtakartasta.

Kuvan 4.2 arvovirtakartta kuvaa korkean prioriteetin ominaisuuspyyntöä pienessä yrityksessä. Siinä pyyntö uudesta ominaisuudesta tulee sähköpostitse työnjohtajalle, joka hyväksyy sen keskimäärin kahdessa tunnissa ja lähettää sähköpostitse eteenpäin. Tämän jälkeen on nopea tekninen arviointi ja työn määrääminen kehittäjälle. Koska kyseessä on korkean prioriteetin ominaisuus, kehittäjä löytyy noin tunnissa. Ominaisuuden kehittäminen ja testaus eivät vie itsessään kuin kaksi tuntia, ja verifioinnin odotus ja verifiointi kestävät yhteensä 30 minuuttia. Myös käyttöönotto on erittäin nopea. Yhteenvetona lisäarvoa tuottavaa työtä on yhteensä kaksi tuntia ja 40 minuuttia, mikä tarkoittaa 33% tehokkuutta. Tämä on jo erittäin tehokas prosessi, vaikka prosessin alkupäässä tehostamisen varaa yhä onkin [PP08, s. 86]. Monissa ohjelmistoyrityksessä optimoinnin varaa on huomattavasti enemmän.

Arvovirtakartat ovat työkalu asiakkaan näkökulman ymmärtämiseksi ja hukan poistamiseksi eivätkä itsessään tuota lisäarvoa. Siksi arvovirtakartat tulisi pitää yksinkertaisina, ja niiden laatimisen jälkeen tulisi pyrkiä välittömiin toimenpiteisiin, jotta saavutetaan tuloksia. Kun arvovirtakartta on valmis, vastataan seuraaviin kysymyksiin [PP08, s. 85]:

1. Kuinka kauan tuotteen valmistus (tai asiakkaan vaatimuksen täyttäminen) kestää?

2. Kuinka monta prosenttia ajasta käytetään arvon tuottamiseen?

Eräs tehokas tapa lisätä prosessin tehokkuutta on laatia tulevaisuuden arvovirtakartta. Tulevaisuuden arvovirta on ketju, jossa on yhdestä kolmeen parannusta. Esimerkiksi suurimmat hukat tai pullonkaulat on poistettu. Tulevaisuuden arvovirta tulee olla toteuttavissa lyhyellä aikavälillä (3-6 kk).

Taulukko 4.2: Seitsemän hukkaa

Seitsemän tuotannon hukkaa	**Seitsemän ohjelmistokehityksen hukkaa**
Ylituotanto	Ylimääräiset ominaisuudet
Odottaminen	Viiveet
Kuljetus	Tuotosten siirrot
Yliprosessointi	Uudelleenoppiminen
Varastointi	Osittain tehty työ
Liike	Tehtävien vaihdot
Virheet	Virheet

Edellisessä luvussa käytiin läpi Toyotan tunnistamat seitsemän perustavanlaatuista olevaa hukkaa. Poppendieckit ovat tunnistaneet [PP08, s. 74–82] näiden vastineet ohjelmistokehityksessä, jotka ovat näkyvillä taulukossa 4.2. Näiden tiedostaminen auttaa tunnistamaan hukkaa. Hukan tunnistaminen vaatii harjoittelua, koska ihmiset sokeutuvat helposti omille toimintatavoilleen pitäen niitä kaikkia tärkeinä ja lisäarvoa tuottavina. Kuinka paljon hukkaa voidaan käytännössä poistaa? Middleton ja Sutton mainitsevat [MS05, s. 29], että Lockheed Martinin Lean-käyttöönotto -projekti vastaa tähän seuraavasti: ”Tyypillinen analyysi osoittaa arvon lisääntymistä tapahtuvan noin prosentin verran kokonaisajasta.” 99 prosenttia ajasta (ei hinnasta) on siis hukkaa ei-Lean-projektissa. Seuraavassa on tarkemmin esitelty ohjelmistokehityksen seitsemän hukkaa:

- **Ylimääräiset ominaisuudet.** Useissa ohjelmistoissa on paljon enemmän ominaisuuksia kuin käyttäjät tarvitsevat. ”Käyttäjät ovat yleensä vähemmän kiinnostuneita monimutkaisista ominaisuuksista kuin markkinointi ja kehitys, ja monimutkaiset ominaisuudet kasvattavat kehittämisaikataulua suhteettomasti [McC02, s. 29].” Kehittäjiä saattaa houkutella kokeilunhaluisina ihmisinä li-

sätä tuotteeseen joitain uusia teknisiä ominaisuuksia nähdäkseen, kuinka heidän ideansa toimivat käytännössä, ja oppiakseen uutta, mutta tätä kannattaa välttää. Jokainen ylimääräinen bitti ohjelmistossa lisää sen monimutkaisuutta ja siten myös virheiden määrää. Jokainen bitti on myös testattava, käyttöönotettava ja ylläpidettävä, joten hukan määrä kertautuu ohjelmiston elinkaaren aikana. Eräs käytännön tapa välttää ylimääräisiä ominaisuuksia on noudattaa päivittäisessä työssä niin sanottua YAGNI-prinsiippiä (You aren't gonna need it). Kyseinen prinsiippi on peräisin XP-prosessimallista [LZE04], joka on yksi ketteristä ohjelmistokehitysmenetelmistä. Prinsiipillä tarkoitetaan, että kun mieleen tulee jokin asia tai ominaisuus, jota tarvitaan tulevaisuudessa, se jätetään tekemättä. Tehdään vain niitä asioita, joita tarvitaan juuri tässä ja nyt, koska yksinkertaisin mahdollinen järjestelmä on toimivin ratkaisu.

- **Viiveet.** Viiveet ja odottaminen ovat yleisiä ohjelmistokehityksessä. Jopa niin yleisiä, että pahimmassa tapauksessa niiden oletetaan kuuluvan ohjelmistokehitykseen. Viiveitä voi olla esimerkiksi projektin aloituksessa, vaatimusmäärittelyssä, dokumentoinnissa, katselmoinnissa tai virheiden korjauksessa. Odotuksesta seuraa se, että arvon tuottaminen asiakkaalle hidastuu. Kun tärkeä asiakas soittaa kiireellisestä ominaisuudesta, joka pitää saada tuotantoon niin pian kuin mahdollista, on arvovirran viiveillä suuri merkitys siihen, kuinka nopeasti ominaisuus on käyttöönotettu.

  Kehittäjä tekee kriittisen päätöksen keskimäärin 15 minuutin välein [PP08, s. 80]. Kaikkea päätöksiin tarvittavaa tietoa on käytännössä mahdoton dokumentoida. Mikäli kehittäjällä on hyvä ymmärrys asiakkaan maailmasta, pystyy hän tekemään viiveettä oikeita päätöksiä. Usein päätöksentekoon tarvittavia tietoja tarvitsee kuitenkin kysyä asiakkaalta. Mikäli vastauksen saaminen on hidasta, istuu kehittäjä suurimman osan työpäivästään odottelemassa vastauksia, tai vaihtoehtoisesti hän arvaa parhaan ratkaisun kyllästyessään odottamiseen. Tällaisten viiveiden minimoimiseksi tulisi asiakkaan maailmaa tuntevan henkilön olla helposti kehittäjien saavutettavissa. Paras tilanne on, mikäli kyseinen henkilö työskentelee samassa tilassa kehittäjien kanssa. Tällöin kommunikointi on viiveetöntä. Usein kehittäjillä on myös tuotteeseen liittyviä ohjelmistoteknisiä kysymyksiä, joihin osaavat vastata saman projektin muut kehittäjät. Vastausten saaminen näihin helpottuu, mikäli kehittäjät työskentelevät samassa tilassa.

- **Tuotosten siirrot.** Tuotosten siirtoa on esimerkiksi vaatimusdokumenttien luovutus järjestelmäanalyytikolta kehittäjille. Tuotosten siirroissa aiheutuu tiedon

häviämistä, koska alkuperäinen työntekijä ei voi kirjoittaa kaikkea tietämäänsä ja oppimaansa paperille. Tämä on tietotyössä erittäin suuri hukka. Vielä suurempi hukka aiheutuu, mikäli tuotos siirtyy ryhmältä toiselle ryhmälle, koska tässä tapauksessa suhteellisesti vieläkin suurempi osa tiedosta jää ainoastaan alkuperäisen ryhmän tietoon. [PP03, s. 7-8]

Kaikkea tietoa ei voi dokumentoida. Kyse on niin sanotusta hiljaisesta tiedosta *(engl. tacit knowledge)*. Hiljainen tieto on kokemuksen ja kehon tietoa, jota on hyvin vaikea jakaa. Esimerkiksi pieni lapsi oppii ajamaan polkupyörällä vain harjoittelemalla. Hän tuskin oppisi ajamaan ainoastaan kuuntelemalla ohjeita, kuinka polkupyörällä ajetaan. Samasta syystä autokouluissa on ajotunteja. Myöskään pianonsoittoa ei opi vain lukemalla, kuinka sitä soitetaan. Koska hiljaisen tiedon siirtäminen on hankalaa, sitä häviää aina tuotoksia siirrettäessä.

Helpoin tapa vähentää tiedon häviämistä on pienentää tuotosten siirtojen lukumäärää. Muita Poppendieckien [PP08, s. 78] löytämiä tapoja ovat esimerkiksi moniosaavien *(engl. cross-functional)* tiimien käyttäminen, keskustelut kasvokkain dokumenttien sijaan ja keskeneräisten töiden julkaisu palautteen saamiseksi.

- **Uudelleenoppiminen.** Uudelleenoppimisen voisi määritellä ehkä parhaiten seuraavasti: ”Todeta jotain, joka aiemmin tiedettiin, mutta unohdettiin”. Tätä voi tapahtua esimerkiksi projektipalavereissa, ohjelmiston arkkitehtuuripäätöksissä, käyttöliittymäsuunnittelussa tai millä tahansa ohjelmistokehityksen osa-alueella.

- **Osittain tehty työ.** Keskeneräisen työn suurin ongelma on, että sen toimivuudesta ei ole takeita. Keskeneräiseksi työksi tulee pääsääntöisesti katsoa kaikki työ, mikä on aloitettu, mutta ei ole käyttöönotettu tuotantoon. Toisin sanoen Lean-projektissa tehdyn työn määritelmä *(engl. definition of done)* on ”käyttöönotettu tuotantoon”. Osittain tehty työ käy myös usein jälkeenpäin tarpeettomaksi ja haittaa vain muuta kehitystä. Taloudellisesti keskeneräiseen työhön sitoutuu resursseja ja pääomaa. Entä jos osittain tehty työ estää koko ohjelmiston käyttöönoton? Minimoimalla osittain tehdyn työn määrä pienennetään samalla ohjelmistoprojektin riskejä. Esimerkkejä osittain tehdystä työstä ovat toteuttamattomat määrittelyt vaatimukset, integroimaton koodi, testaamaton koodi, dokumentoimaton koodi ja käyttöönottamaton koodi. [PP08, s. 74]

- **Tehtävien vaihdot.** Kun kehittäjä vaihtaa tehtäviä (esimerkiksi siirtyy projek-

tista toiseen), kuluu aina tietty aika ennen kuin kehittäjä pääsee sisään uuteen tehtävään. Ohjelmistokehitys vaatii paljon ajattelua, ja kuluu merkittävä määrä aikaa ennen kuin ihminen kykenee suorittamaan uutta tehtävää täydellä teholla. Keskeytyksillä on samankaltainen vaikutus ohjelmistokehittäjän työhön kuin tehtävien vaihdolla. McConnellin mukaan [McC02, s. 506] kehittäjät tarvitsevat 15 minuuttia saavuttaakseen keskittyneisyyden tilan. "Jos kehittäjää häiritään 11 minuutin välein, he pystyvät tuskin koskaan saavuttamaan keskittyneisyyttä eivätkä siten saavuta korkeinta tuottavuusastettaan."

- **Virheet.** Virheistä aiheutuvan hukan määrä riippuu ohjelmistokehityksessä siitä, kuinka aikaisin virhe havaitaan. Kriittinen virhe, joka havaitaan kahden minuutin kuluttua sen tekemisestä, on paljon pienempi hukka kuin pieni virhe, joka havaitaan kahden kuukauden kuluttua. Oikea tapa poistaa virheisiin liittyvää hukkaa on pyrkiä estämään virheiden syntyminen. Tämän jälkeen tulee keskittyä löytämään virheet mahdollisimman aikaisessa vaiheessa sekä pyrkiä oikeanlaisella kehitysprosessilla helpottamaan tuotosten verifiointia. Virheen löytyessä tulee miettiä kuinka vastaavan virheen syntyminen voitaisiin jatkossa estää. Lisäksi vähemmän tuotoksia tarkoittaa vähemmän verifioitavaa ja vähemmän virheitä.

  Käytännön keinoja virheiden vähentämiseen ovat esimerkiksi testivetoinen kehitys *(engl. test driven development, TDD)* ja hyväksymistestien tekeminen vaatimukselle ennen vaatimuksen toteutusta.

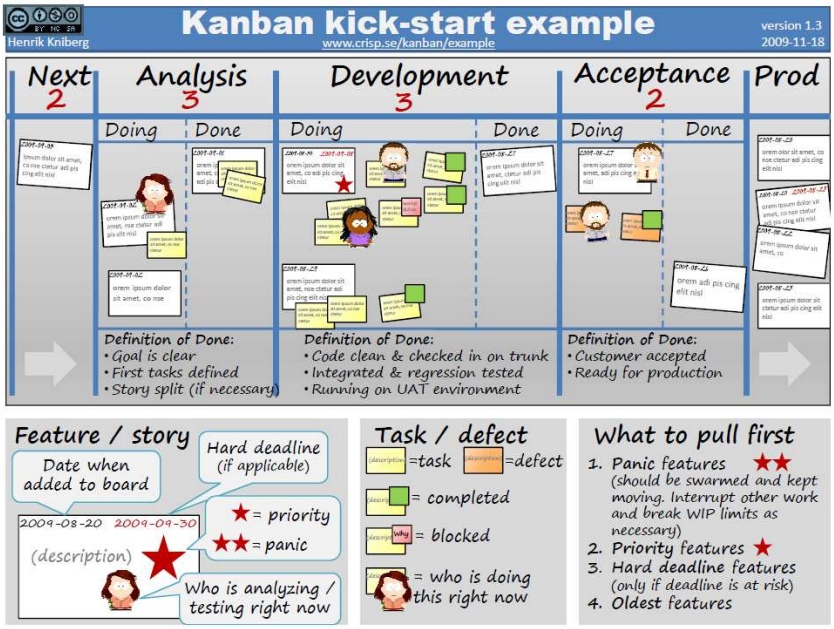
4.1.3 Virtauksen tehostaminen

Virtauksen tehostamisen lähtökohta on taulukossa 4.2 listattujen seitsemän hukan poistaminen. Nopein tie poistaa hukkaa on ottaa käyttöön jo hyväksi havaittuja käytänteitä tai prosesseja, jotka eivät ole ristiriidassa Lean-ajattelun kanssa.

Virtauksen tehostamiseen ja hukan poistamiseen löytyy ohjelmistokehityksessä aiemmin mainittujen lisäksi useita valmiita tapoja, kuten Kanban-prosessimalli, iteratiivinen ohjelmistokehitys, jatkuva integraatio, automatisoidut testit ja verifiointivetoinen kehitys *(engl. Verification Driven Development)*. Näistä Kanban-prosessimalli soveltuu erittäin hyvin Lean-ohjelmistokehitykseen, koska sen ideologia on peräisin Lean-maailmasta ja siinä pyritään yksiosaiseen virtaukseen. Tämä ei ole kuitenkaan missään nimessä ainoa tapa tehdä Lean-ohjelmistokehitystä, vaan on mietittävä omaa ympäristöä ja siihen parhaiten soveltuvia kehitysprosesseja. Ohjelmistokehityksessä Kanban mielletään tavallisesti ketteräksi kehitysmenetelmäksi, kuten Scrum (mitä ne ovatkin). Leanin tapauksessa sitä kuitenkin kannattaa ajatella pi-

kemminkin toimintatapana tai työkaluna, jonka tarkoituksena on tehostaa virtausta. Lean-ohjelmistokehityksessä prosessit eivät ole myöskään staattisia, vaan niitä parannetaan jatkuvasti. Mikäli prosessit pysyisivät aina samanlaisina, ei hukkaa pystyttäisi poistamaan.

Ohjelmistokehityksen Kanban-prosessimallia ei tule sekoittaa yleisemmin tunnettuun Lean-maailmasta peräisin olevaan kanban-käsitteeseen *(suom. merkki, kyltti, kortti)*, jolla tarkoitetaan signaalia yrityksen prosesseissa. Esimerkiksi tyhjän astian lähettäminen tehtaan tuotantolinjalla voi olla signaali (kanban), että se pitää täyttää tietyn tyyppisillä osilla ja palauttaa täytenä tiettyyn paikkaan [Lik04, s. 106–107].



Kuva 4.3: Esimerkki Kanban-tehtävätaulusta [Kni]

Kanban-prosessimallin syvempi käsittely vaatisi oman tutkielmansa, eikä se siten mahdu tämän tutkielman piiriin. Henrik Knibergin laatima esimerkki Kanban-prosessimallin tehtävätaulusta on nähtävissä kuitenkin kuvassa 4.3. Tehtävätaulu on kehittäjien työtilan seinällä oleva taulu, joka sisältää tavallisesti käyttäjäkertomuksia ja tehtäviä. Käyttäjäkertomukset tai vaatimukset pilkotaan tehtäviksi, joita kehittäjät tekevät. Molemmantyyppiset laput etenevät tehtävätaululla sovitusti sitä mukaa, kun kehittäjät saavat tehtäviä tehtyä.

4.1.4 Imuohjattu kehittäminen

Imuohjatussa kehityksessä perusideana on, että tehtäviä ei määrätä kehittäjille vaan he itsenäisesti valitsevat tehtävät. Myöskään vaatimuksia ei suoraan määrätä hen-

kilöille vaan ne priorisoidaan, ja kehitystiimi toteuttaa niitä ottaen uusia sitä mukaa työn alle, kun vanhat vaatimukset valmistuvat. Tämä voidaan tehdä joko iteraatioittain tai välittömästi, kuten Kanban-prosessimallissa. Edellä mainittu tehtävätaulu on yksi käytännön työkalu imuohjatun kehittämisen toteuttamiseksi. [PP03, s. 2–3]

Kanban-prosessimalli on imuohjattu, joten se on yksi tapa toteuttaa kyseinen Lean-ajattelun ydinkonsepti. Ideana Kanbanissa on, että seuraava työvaihe toimii asiakkaana edelliselle työvaiheelle ja ”imee” vaatimukset eteenpäin prosessissa (ks. kuva 4.3). Imuohjaus on mahdollista toteuttaa myös muunlaisilla prosesseilla, esimerkiksi soveltamalla Scrum-prosessimallia.

4.1.5 Täydellisyyteen pyrkiminen

Täydellisyyteen pyrkiminen toimii Lean-ohjelmistokehityksessä samoin periaattein kuin muillakin aloilla: pysähdytään tietyin aikavälein miettimään, kuinka luodaan enemmän arvoa, saadaan aikaiseksi parempi arvovirta, tehostetaan virtausta tai parannetaan imuohjausta. Ohjelmistoalalla hyvä tilaisuus tähän on iteraatioiden vaihtuessa, mikäli käytössä on iteraatiopohjainen kehitysmalli. Näin saavutetaan jatkuva parannus ja päästään koko ajan lähemmäksi täydellisyyttä. Ohjelmistokehityksessä täydellisyyteen pyrkimiseen liittyy kaksi eri osa-aluetta: tuote ja prosessi. [MS05, s. 316]

Tuotettaessa materialistisia tavaroita täydellisyyteen pyrkiminen tarkoittaa varianssin minimointia siten, että tuotetut tavarat ovat mahdollisimman lähellä täydellistä tavaraa. Fysiikan lakien takia täydellisyyttä ei voida kuitenkaan koskaan saavuttaa, koska järjestelmässä on aina tietty määrä entropiaa. [MS05, s. 316–317]

Ohjelmistojen osalta tilanne on kuitenkin eri. Ohjelmistot ovat matematiikkaa jolloin ne teoriassa voidaan osoittaa matemaattisesti oikeiksi. Tämä tarkoittaa, että ne toimivat täydellisesti eli eivät sisällä yhtäkään virhettä. Käytännön tasolla matemaattinen todistus on usein lähestulkoon mahdotonta, mutta on huomionarvoista, että ohjelmistojen osalta tällainen mahdollisuus on olemassa. Eikö ohjelmistojen tulisi tämän ansiosta olla helpompia valmistaa virheettömiksi kuin materialististen tuotteiden? Täysin virheetönkin ohjelma voi kuitenkin tehdä täysin vääriä asioita. Sen lisäksi, että tekee asiat oikein, tulee tehdä oikeita asioita. [MS05, s. 316–319]

Prosessin osalta täydellisyys tarkoittaa kaiken hukan eliminoimista. Jotta täydellisyyttä voitaisiin lähestyä, on tärkeää nähdä mielessä kuva täydellisestä prosessista sekä esittää se muille [WJ03, s. 90]. Tämä vaatii kaikkien neljän aiemman ydinkonseptin läpikäymistä ja ymmärtämistä. Yksinkertaistettuna täytyy ymmärtää asiakkaan tarpeet (arvot), tuntea nykytila (arvovirta), ymmärtää syvällisesti prosessit ja

työkalut (virtaus ja imuohjaus) sekä nähdä mielessään kuva mallista, johon halutaan päästä (täydellisyys). Tämän jälkeen voidaan täydellisyyttä lähestyä askel kerrallaan, ja samalla opitaan lisää muista neljästä ydinkonseptista.

Luvussa 3.3 esiteltiin jidokan (ihmisavusteinen automaatio) neljä vaihetta: 1) havaitse vika, 2) pysähdy, 3) suorita välitön korjaustoimenpide sekä 4) tutki perimmäinen syy ja aseta vastatoimet. Nämä ovat yhtälailla käytettävissä myös ohjelmistokehityksessä. Esimerkiksi kirjoittamalla sovellukset käyttäen testivetoista kehitystä ja automaattista testausta, saadaan kone havaitsemaan sovelluksessa olevia virheitä. Vastatoimien osalta tulee muistaa, että vastatoimet asetetaan perimmäistä syytä vasten. Tämä tarkoittaa, että automaattitesti, joka paljastaa esiintyneen vian, ei vältämättä ole jidokan mukainen.

4.2 Ihmiset ja yhteistyökumppanit

Luvussa 3.2.3 käytiin läpi ihmisiin ja yhteistyökumppaneihin liittyviä periaatteita, joita japanilaisen näkemyksen mukaan kuuluu Leaniin. Kyseiset Likerin [Lik10] esittämät periaatteet ovat:

- Kasvata johtajia, jotka ymmärtävät työn perusteellisesti, noudattavat filosofiaa ja opettavat sitä muille.
- Kehitä poikkeuksellisen eteviä ihmisiä ja ryhmiä, jotka noudattavat yhteistä filosofiaa.
- Kunnioita yhteistyökumppaneilla ja alihankkijoilla laajennettua verkostoa tarjoamalla heille haasteita ja auttamalla heitä kehittymään.

Koska nämä ovat perusperiaatteita, voi niitä soveltaa sellaisenaan myös ohjelmistokehitykseen. Näiden lisäksi ihmisiin ja yhteistyökumppaneihin liittyviä periaatteita ja käytänteitä on esitetty tai johdettu myös suoraan Lean-ohjelmistokehitykseen.

Lean-ohjelmistokehityksen suurimpina vaikuttajina voidaan pitää Poppendieckejä, joten on luonnollista lähteä tarkastelemaan heidän seitsemää perusperiaatettaan, jotka esiteltiin luvussa 4. Nämä seitsemän periaatetta ovat: poista hukka, rakenna laatu ohjelman sisään, luo tietoa, lykkää sitoutumista, toimita nopeasti, kunnioita ihmisiä ja optimoi kokonaisuus [PP08]. Periaatteista havaitaan, että ”kunnioita ihmisiä” liittyy suoraan ihmisiin.

Poppendieckit esittävät kunnioita ihmisiä -periaatteeseen liittyen neljä työkalua [PP03, s.95–123]:

- **Itsemääräämisoikeus.** Työntekijöitä tulee kunnioittaa, ja heidän tulee osallistua aktiivisesti oman työnsä suunnitteluun ja parantamiseen.

- **Motivaatio.** Ihmiset tarvitsevat enemmän kuin työlistan. Heidän työllään tulee olla tarkoitus, joka on saavutettavissa. Tiimin tulee voida myös kommunikoida suoraan asiakkaan kanssa. Motivaation rakennuspalikat ovat: yhteenkuuluvuus, turvallisuus, osaaminen ja edistyminen.

- **Johtajuus *(engl. leadership).*** Ohjelmistokehittäjät tarvitsevat enemmän kunnioitettuja johtajia *(engl. respected leaders)* kuin resursseja ja aikatauluja hallinnoivia managereita *(engl. managers)*.

- **Ammattitaito.** Ohjelmistoalalla tarvitaan paljon osaamista. Alan osaamista on karkeasti kahdenlaista: teknistä osaamista ja toimialaosaamista (esimerkiksi terveydenhuoltoalan osaaminen, jos tekee kyseisen alan ohjelmistoja). Molemmantyyppinen osaaminen on yritykselle tärkeää kilpailuedun saavuttamiseksi muihin yrityksiin nähden.

Yllä esitetyistä Poppendieckien näkemyksistä havaitaan selviä yhtäläisyyksiä Likerin näkemyksiin. Esimerkiksi johtajuus ja ammattitaito esiintyvät molemmissa. Näiden lisäksi Poppendieckiet kirjoittavat teoksessaan *Implementing Lean Software Development* [PP08, s.126–148] tiimien ja tiimityön merkityksestä, työn visualisoinnista sekä itseohjautuvasta työstä. Poppendieckit myös varoittavat rahallisten bonusten riskeistä kehottaen etsimään tehokkaampia ja vähemmän riskialttiita tapoja tehostaa suorituskykyä. Lukija voi kiinnostuessaan lukea näistä aiheista lisää Poppendieckien kirjasta.

Poppendieckien mukaan [PP08, s.207–222] kumppanuuksissa ei ole kyse halvemmista hinnoista, riskinhallinnasta tai kapasiteetin nostamisesta. Niissä on kyse synergiasta. Ihmiset ja yritykset saavuttavat parempia tuloksia tekemällä yhteistyötä. Synergiaa tulisi hakea muun muassa ulkoistuksissa ja sopimusneuvotteluissa. Lean-ohjelmistokehityksen sopimuksiin Poppendieckit esittävät perinteisistä sopimuksista poikkeavia suhdesopimuksia *(engl. relational contracts)*, jotka keskittyvät kuvaamaan yritysten käytännön yhteistyötä ja kumppanuutta.

Middleton ja Sutton toteavat Lean-valmistuksesta opittuna asiana, että järjestelmää ei ole mahdollista ajaa korkealla käyttöasteella ennen kuin poistaa järjestelmään suurta varianssia aiheuttavat tekijät. Ohjelmistokehityksessä varianssia aiheuttavat tekijät ovat yleensä tietyt ohjelmistokehittäjät. Ohjelmistokehittäjät ovat riippuvaisia toistensa työstä, joten hyväkään kehittäjä ei pysty parhaimpaansa huonossa ”järjestelmässä”, joka sisältää varianssia. ”Huono järjestelmä päihittää hyvän

kehittäjän joka kerta". Middleton ja Sutton perustelevat, että on validia parantaa tai poistaa vuosittain 10% heikoiten työstään suoriutuvaa. Ideana on, että heille tarjotaan koulutusta ja valmennusta suorituskykynsä parantamiseksi. Jos he kieltäytyvät tästä, pyydetään heitä etsimään työtä muista yrityksistä. Samaan ajatukseen liittyy, että tunnistetaan vuosittain myös 10% parhaiten työstään suoriutuvaa ja etsitään keinoja kuinka he saadaan suoriutumaan vielä paremmin, esimerkiksi tarjoamalla ylennystä tavoitteiden täyttyessä. Tällöin koko yrityksen suorituskyky työntekijöiden osalta kasvaa jatkuvasti. Kyseinen johtamistapa on ollut Middletonin mukaan käytössä General Electricillä (GE). Heillä suoritettiin vuosittain myös nimetön työntekijäkysely. Yllättävänä yksityiskohtana GE:n henkilöstö oli aina sitä mieltä, että kitkemisprosessia ei suoritettu tarpeeksi ankarasti. Middletonin ja Suttonin mukaan kyseinen tulos johtuu nimenomaan siitä, että kehittäjät ovat riippuvaisia toisistaan ja tarvitsevat toimivan "järjestelmän" pystyäkseen tekemään työnsä hyvin. [MS05, s.343–348]

4.3 Teknologia

Teknologia liittyy hyvin läheisesti ohjelmistokehitykseen. Leanissa suhtaudutaan kuitenkin teknologiaan melko varovaisesti, vaikka se onkin yksi tärkeä osa-alue. Teknologia-ajattelu on peräisin massatuotannosta ja Leanissä pyritään olemaan enemmän asiakas- ja ihmiskeskeisempiä, mikä selittää varovaista suhtautumista teknologiaan. Likerin 4P-mallissa on periaate 'Käytä ainoastaan luotettavaa, perusteellisesti testattua teknologiaa, joka palvelee ihmisiä ja prosesseja' [Lik10, s.159–168]. Tämä kuvaa hyvin Leanin ja teknologian suhdetta. Leanissa ihmiset ja prosessit ovat ykkösasia, ja vasta kun nämä ovat kunnossa, voidaan teknologian avulla helpottaa tai tehostaa toimintoja.

Poppendieckien seitsemässä perusperiaatteessa [PP08] ei suoraan puhuta teknologiasta mitään. Tämäkin havainnollistaa, että Lean-ohjelmistokehityksessä teknologia ei ole ykkösasia, vaan pikemmin tapa toteuttaa Lean-periaatteita.

Hibbs, Jewett ja Sullivan [HJS09] esittävät kirjassaan *The Art of Lean Software Development* kuusi Lean-ohjelmistokehityksen käytäntöä, joista kolme liittyy suoraan teknologiaan. Teknologiaan liittyvät ovat:

- **Lähdekoodinhallinta ja skriptatut käännökset.** Lähdekoodin versionhallinta helpottaa tiimin yhteistyötä ja kommunikointia. Skriptatut käännökset *(engl. scripted builds)* estävät virheitä, koska tietokone rakentaa ohjelmiston joka kerta samalla tavalla. Tämä myös helpottaa uusien kehittäjien mukaantuloa pro-

jektiin.

- **Automaattinen testaus.** Virheiden estäminen on yksi Leanin kulmakivi. Ohjelmistomaailmassa tätä voidaan tehdä automaattisella testauksella. Automaattista testausta voidaan tehdä monella eri tasolla: yksikkötestaus, integraatiotestaus, hyväksymistestaus, suorituskykytestaus, testivetoinen kehitys jne. Yhteistä näille kaikille on, että ihminen kirjoittaa testit ja tietokone suorittaa testit sekä ilmoittaa löytämistään virheistä.

- **Jatkuva integraatio.** Integraatiolla tarkoitetaan tilannetta, jossa kaikki kehitysponnistelujen tuotokset yhdistetään. Yksittäiset komponentit, käyttöliittymät, tietokannat ja kaikki muu ohjelmistoon liittyvä sidotaan toisiinsa ja verifioidaan, että järjestelmä toimii kokonaisuutena. Jatkuva integraatio tarkoittaa, että kokonaisuuteen tuodaan jatkuvasti pieniä muutoksia, ja nähdään suoraan, toimivatko ne. Tällöin ohjelmiston elinkaari ei tarvitse enää erillistä integrointivaihetta. Jatkuva integraatio vaatii toimiakseen lähdekoodin versionhallinnan, automaattisen testauksen sekä skriptatut käännökset. Jatkuva integraatio tukee erinomaisesti Leanin juuri-oikeaan-aikaan-ajattelua sekä yksiosaista virtausta.

Yllä esitetyt teknologiat ovat käytössä lähes kaikissa Lean- tai ketterissä ohjelmistokehitysprojekteissa [HJS09]. Niitä voidaan täten pitää luotettavina ja perusteellisesti testattuina, joten ne sopivat Lean-ohjelmistokehitykseen. Maailmasta löytyy näiden lisäksi lukematon määrä erilaisia ohjelmistokehitykseen liittyviä teknologioita sekä työkaluja, ja mitkään eivät ole kaikkiin ympäristöihin sopivia. Jokainen ympäristö on yksilönsä, ja teknologiat tulee valita sen mukaisesti. Lean-ohjelmistokehityksessä oleellista on nimenomaan huomioida, että ihmisten ja prosessien ei tule sopeutua teknologiaan, vaan päinvastoin: valittujen teknologioiden tulee tukea ihmisiä ja prosesseja.

4.4 Yhteenveto

Lean-ohjelmistokehitys on Leanin soveltamista ohjelmistokehitykseen. Lean-valmistus on lähtöisin automaailmasta, mikä poikkeaa ohjelmistokehityksestä. Tästä syystä käytänteitä ei voida suoraan kopioida ohjelmistoalalle, vaan periaatteita täytyy soveltaa luovasti omaan ympäristöön.

Lean-ohjelmistokehityksen suurimpina vaikuttajina voidaan pitää Mary ja Tom Poppendieckiä. He ovat esittäneet seitsemän Lean-ohjelmistokehityksen perusperi-

aatetta: poista hukka, rakenna laatu ohjelman sisään, luo tietoa, lykkää sitoutumista, toimita nopeasti, kunnioita ihmisiä ja optimoi kokonaisuus [PP08]. Lean-ohjelmistokehityksessä ei tule kuitenkaan unohtaa Womackin ja Jonesin Lean-ajattelun viittä ydinkonseptia: arvo, arvovirta, virtaus, imu ja täydellisyys.

Lean-ohjelmistokehityksen prosesseihin liittyy arvoselvitys, arvovirran tunnistaminen arvovirtakartan avulla, hukan poistaminen, virtauksen tehostaminen esimerkiksi kanban-prosessimallin avulla, imuohjattu kehittäminen ja täydellisyyteen pyrkiminen pysähtymällä tietyin aikavälein miettimään, kuinka toimintoja voidaan parantaa. Ohjelmistokehityksen seitsemän hukkaa ovat ylimääräiset ominaisuudet, viiveet, tuotosten siirrot, uudelleenoppiminen, osittain tehty työ, tehtävien vaihdot ja virheet [PP08].

Ihmiset liittyvät läheisesti Leaniin ja niin myös Lean-ohjelmistokehitykseen. Ihmisiin liittyviä asioita ovat muun muassa itsemääräämisoikeus, motivaatio, johtajuus ja ammattitaito. Lisäksi on hyvä huomata, että huono ”järjestelmä” voittaa aina hyvän kehittäjän.

Leanissa suhtaudutaan teknologiaan melko varovaisesti. Myös Lean-ohjelmistokehityksessä tulisi käyttää ainoastaan luotettavaa ja perusteellisesti testattua teknologiaa, joka palvelee ihmisiä ja prosesseja. Prosessien ei siis tulisi olla teknologiasidonnaisia. Ohjelmistokehitykseen liittyviä perusteknologioita ovat mm. lähdekoodin versionhallinta, skriptatut käännökset, automaattinen testaus ja jatkuva integraatio. Lean-ohjelmistokehityksessä voidaan käyttää mitä hyvänsä teknologiaa, kunhan se tukee Lean-ajattelua, on huolellisesti testattu ja palvelee sekä ihmisiä että prosesseja.

5 Tapausyritys Sysdrone

Tapaustutkimuksen kontekstin ymmärtämisen kannalta on hyvä tuoda esille taustatietoja tutkittavasta tapauksesta. Tutkimuksen tapausyrityksenä toimii ohjelmistoja valmistava Sysdrone Oy.

5.1 Tapausyrityksen esittely

Sysdrone Oy on Jyväskylässä toimiva Protacon-konserniin kuuluva ohjelmistotalo, joka valmistaa asiakaskohtaisesti räätälöityjä ohjelmistoja ketteriä menetelmiä käyttäen. Ohjelmistoprojektien pääpainopiste on terveysteknologian sovelluksissa, mutta projekteja tehdään myös muille toimialoille.

Sysdrone on perustettu vuonna 2006. Perustajina toimivat Ilkka Laitinen, Jani Lehtinen sekä allekirjoittanut. Sysdrone liittyi osaksi Protacon-konsernia joulukuussa 2008, kun Protacon Solutions Oy osti Sysdronen osake-enemmistön. Tutkimuksen tekemisen aikaan, maaliskuussa 2011, Sysdrone työllisti 13 työntekijää. Sysdrone toimii samoissa tiloissa ja tekee erittäin tiivistä yhteistyötä Protacon Solutions Oy:n kanssa. Allekirjoittanut toimii tätä kirjoitettaessa ohjelmistokehitysjohtajana sekä Sysdrone Oy:ssä että Protacon Solutions Oy:ssä vastuualueena Protacon Solutions -konsernin ohjelmistotuotanto.

Vuonna 2010 Sysdronen liikevaihto oli 553 000 euroa ja tilikauden tulos 88 000 euroa. Liikevaihdossa kasvua edelliseen vuoteen oli 112,70 prosenttia, eli kasvua on tapahtunut.

Sysdronen erityisosaamista ovat terveysteknologian sovellukset sekä niihin liittyvät erityispiirteet ja viranomaisvaatimukset. Ohjelmistojen kehittämisen lisäksi terveysteknologian sovelluksissa kehitystiimi laatii viranomaisten vaatimat dokumentaatiot tietojärjestelmästä ja sen kehityksestä. Tätä varten yrityksessä on kehitetty niin sanottu täydennetty Scrum-malli, joka täyttää viranomaisvaatimukset sovelluskehityksen osalta. Mallia käyttämällä voidaan terveysteknologian sovelluksia kehittää ketterillä ohjelmistokehitysmenetelmillä. Kyseisestä mallista löytyy lisätietoja Joni Purojärven pro gradu -tutkielmasta [Pur10].

Ohjelmistot valmistetaan pääosin Windows-ympäristöihin Microsoftin .NET-teknologioita käyttäen. Web-kehityksessä hyödynnetään paljon myös Linux-alustaa ja PHP:tä. Näiden lisäksi kehitystä tehdään muillakin ohjelmointikielillä ja -ympäris-

töillä asiakkaiden ja projektien tarpeiden mukaan.

Ohjelmistokehitys tapahtuu Sysdronella tiimeissä. Tiimi koostuu tiiminvetäjästä sekä kehittäjistä, jotka kaikki työskentelevät samassa avotilassa. Myös tiiminvetäjä osallistuu ohjelmiston kehittämiseen. Tiimi vastaa yhdessä työnjäljestä ja asiakkaalle toimitetusta lopputuotoksesta. Ohjelmistokehityksen peruskäytänteet ovat tiimien kesken yhtenäiset, mutta lähtökohtana on, että jokainen tiimi vie omia käytänteitään eteenpäin kahden viikon välein pidettävien jälkipuintipalaverien kautta. Tiimi voi itsenäisesti ottaa kokeiluun uusia toimintatapoja, ja toimiviksi todetut käytänteet yhtenäistetään kaikkien tiimien osalta.

Sysdronen ohjelmistokehityksessä käytetään alalla hyviksi todettuja käytänteitä. Näitä ovat esimerkiksi lyhyet iteraatiot (kaksi viikkoa), testivetoinen kehitys, jatkuva integraatio ja automaattinen testaus. Lisäksi Sysdronella kaikki kirjoitettu koodi katselmoidaan. Tässä hyödynnetään sisäisesti kehitettyä työkalua, joka automatisoi katselmointiprosessia. Katselmoijana toimii kehitystiimin toinen jäsen.

Kehitystä pyritään tekemään asiakaskeskeisesti, ja asiakaskommunikoinnin merkitystä on korostettu. Käytännössä tämä tarkoittaa, että iteraatiot suunnitellaan asiakkaan kanssa yhteistyössä ja asiakasta tavataan kasvotusten. Iteraation jälkeen asiakkaalle julkaistaan toimiva versio ohjelmistosta, jolloin asiakaspalautetta saadaan nopeasti.

5.2 Lean-ohjelmistokehitys tapausyrityksessä

Tutkimuksen tavoitteena oli selvittää, mitkä ovat suurimmat ohjelmistokehityksen nopeutta rajoittavat tekijät tapausyrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Tässä yhteydessä Lean-ohjelmistokehityksellä viitataan nimenomaan Sysdronen tapaan tehdä Lean-ohjelmistokehitystä. Osa Sysdronen käytänteistä on tullut uusina Lean-ohjelmistokehityksen käyttöönoton myötä ja osa on ollut jo aiemmin käytössä. On hyvä huomioida, että työkalut ja menetelmät ovat vain apukeinoja Leanin periaatteiden toteuttamiseen, eivät Leanin ydin tai onnistumisen tae. Työkalujen ja käytänteiden tulee pohjautua Leanin ajattelumalliin ja ydinkonsepteihin.

Sysdronessa aloitettiin Lean-kokeilu kahden projektin osalta kesällä 2010. Eri työkaluja on otettu kokeiluun tämän jälkeen vähitellen, ja hyväksi todetuista on pidetty kiinni. Tulokset olivat lupaavia, joten Lean-ohjelmistokehitys päätettiin tuoda kaikkiin projekteihin mukaan vuoden 2011 alusta lähtien.

Lean on ensisijaisesti ajattelumalli ja filosofia, joka ohjaa päivittäistä toimintaa. Käytännössä länsimaissa on kuitenkin totuttu lähtemään filosofian sijaan liikkeel-

le jostain konkreettisesta, eli prosesseista ja käytänteistä. Tapausyrityksessä Lean-ohjelmistokehitys määriteltiin sen käyttöönottoa varten siten, että projektissa toteutetaan seuraavia käytänteitä:

- **Käyttäjäkertomukset:** Tunnistetut asiakasvaatimukset on listattu käyttäjäkertomuksina tuotoslistaan *(engl. product backlog)*. Tämä liittyy Leanin arvoydinkonseptiin.

- **Arvovirta:** Projektin vaatimusten arvovirta on tunnistettu ja havainnollistettu visuaaliseen muotoon. Virrasta käy ilmi vaatimusten työvaiheet sekä tiedon kulku asiakkaalta ja asiakkaalle. Työvaiheista on tunnistettu ovatko ne lisäarvoa tuottavia vai lisäarvoa tuottamattomia vaiheita.

- **Kanban:** Projektissa on käytössä Kanban-prosessi. Tämä tarkoittaa, että teon alla olevaa työn määrää on rajoitettu *(engl. Work-In-Progress Limit)* ja työn kulku on visualisoitu.

- **Imuohjaus:** Kehitysprosessi on imuohjattu. Tämä tarkoittaa, että kehittäjät valitsevat itse, minkä tehtävän tekevät seuraavaksi. Kukaan ei määrää tehtäviä kehittäjille, mutta seuraavaa tehtävää valittaessa tulee noudattaa sovittua kehitysprosessia. Kehittäjät itse pilkkovat käyttäjäkertomukset tehtäviksi, joten he ymmärtävät mistä tehtävissä on kyse.

  Jokaiselle vaatimuksen työvaiheelle on myös määritelty, milloin vaatimus on kyseisen vaiheen osalta valmis (engl. definition of done) ja se voidaan siirtää odottamaan seuraavaa työvaihetta. Vaatimus voidaan siirtää seuraavaan työvaiheeseen, mikäli sallittu teon alla olevien vaatimusten määrä ei kyseisen työvaiheen osalta ylity. Esimerkiksi testaustyövaiheen osalta voi olla määriteltynä, että testattavana saa olla korkeintaan kaksi vaatimusta kerrallaan.

- **Kertomuspiste-estimointi:** Vaatimukset estimoidaan käyttäen niin sanottuja käyttäjäkertomuspisteitä. Jokaiselle tiimin jäsenelle jaetaan kortit 0, 1, 2, 3, 5, 8, 13, 20 ja 40. Kortissa oleva lukuarvo tarkoittaa kertomuspisteiden määrää. Asiakasta edustava henkilö, yleensä tuotteen omistaja, kuvailee käyttäjäkertomuksen, ja tämän jälkeen jokainen kehittäjä lyö pöytään kortin, joka hänen mielestään kuvaa vaatimuksen kokoa parhaiten, kuvapuoli alaspäin. Kun kaikki ovat valmiit, käännetään kortit esiin. Mikäli korttien lukuarvot eroavat toisistaan, jatketaan keskustelua maksimissaan parin minuutin ajan, ja tämän jälkeen lyödään korttia uudelleen. Tätä jatketaan, kunnes löydetään yksimielisyys. Saatu tulos kirjataan vaatimuksen, eli käyttäjäkertomuksen, kooksi. Ker-

tomuspisteet kuvastavat vaatimusten suhteellista kokoa toisiinsa nähden. Yhdelle kertomuspisteelle ei ole määriteltävissä absoluuttista kokoa, vaan kertomuspisteiden koot vaihtelevat projektista toiseen. Menetelmästä käytetään nimitystä suunnittelupokeri *(engl. planning poker)*, ja sen on ensimmäisenä kuvaillut James Grenning vuonna 2002 [Gre02].

- **Läpimenoaika ja tahtiaika:** Projektissa mitataan, kuinka kauan vaatimuksen työstämiseen käytetään aikaa kussakin työvaiheessa. Jokaisen vaatimuksen osalta kerätään ylös tieto, milloin sen työstäminen kyseisessä työvaiheessa aloitettiin ja milloin se oli tämän vaiheen osalta valmis.

  Vaatimuksista kerättyjen aikatietojen perusteella voidaan määrittää läpimenoaika ja tahtiaika. Läpimenoaika tarkoittaa aikaa, joka kestää siitä, kun vaatimus on saatu asiakkaalta, siihen, kun se on toimitettu ja käyttöönotettu asiakkaan tuotantoympäristöön. Tahtiaika taas tarkoittaa tapausyrityksessä aikaa, joka menee vaatimuksen työstämisen aloittamisesta (yleensä ensimmäinen työvaihe vaatimukselle on sen analysointi, mikä karkeasti tarkoittaa sen pilkkomista tehtäviksi) siihen, kun vaatimus on valmis käyttöönotettavaksi. Läpimenoaikaa ja tahtiaikaa käsitellään lisää luvussa 6.3.2.

- **Jälkipuintipalaverit:** Projekteissa pidetään kahden viikon välein palaveri, jossa mietitään, kuinka projektin eri osa-alueita voitaisiin parantaa. Parannuksen kohteena voi olla mikä tahansa, minkä koetaan hidastavan kehitystä. Tässä tutkielmassa tilaisuudesta käytetään nimeä jälkipuintipalaveri (engl. retrospective meeting), mikä tulee ketterien ohjelmistokehitysmenetelmien puolelta. Kyseessä on kuitenkin sama perusajatus kuin Leanin kaizenissa, eli jatkuva inkrementaalinen parannus ja erinomaisuuden tavoittelu.

- **Käännösnäytöt:** Kehitystiimeillä on käytössään niin sanottuja käännösnäyttöjä, jotka ovat punaisena tai vihreänä sen mukaan, kääntyykö uusin versio ohjelmakoodista tällä hetkellä ja läpäiseekö se yksikkö- ja hyväksymistestit. Automaattinen käännös- ja tarkastusprosessi voi sisältää myös muita tarkastuksia. Näyttö on Leanin kannalta andon: signaali, joka varoittaa ongelmista prosessissa.

- **Jidoka:** Mikäli verifioinnissa tai jo tuotannossa olevasta ohjelmasta löydetään virhe, on virheestä ensimmäisenä tietävä kehittäjä velvollinen ilmoittamaan asiasta tiimin muille jäsenille. Tällöin tiimin kaikki kehittäjät lopettavat sen hetkisen työnsä ja nousevat ylös kuuntelemaan mistä on kyse. Tämän jälkeen

tiimi pohtii viiden miksi-kysymyksen analyysiä käyttäen, mikä oli perimmäinen syy ohjelmistovirheen syntymiselle, ja mahdollisia keinoja estää vastaavankaltaiset virheet jatkossa. Kyseessä on siis Jidoka-laadunvalvontaprosessi ohjelmistokehitykseen sovellettuna. Tarkoituksena on välttää ajattelumallia, jossa ohjelmistovirheiden mielletään kuuluvan osaksi ohjelmistokehitystä.

- **Automaattinen hyväksymistestaus:** Hyväksymistestit automatisoidaan siltä osin kuin mahdollista, vaikka testien kirjoittaminen veisi sillä hetkellä pidempään kuin käsityönä tehtävä testaus. Tavoitteena on automatisoida 80% testeistä. Testauksen automatisoinnilla estetään ohjelmistovirheiden syntymistä jatkossa, kun toteutetaan uusia vaatimuksia tai muokataan olemassa olevia toiminnallisuuksia. Tämä voidaan mieltää yhdeksi ohjelmistokehityksen poka-yokeksi.

Näiden käytänteiden noudattaminen ei sanele projekteissa käytettäviä prosessimalleja tai muita toimintatapoja. Projektissa voidaan esimerkiksi käyttää Scrumia ja todeta sen silti olevan Lean-ohjelmistokehityksen mukainen. Kyseiset käytäntänteet ovat olleet lähtökohta käyttöönotettaessa Lean-ohjelmistokehitystä tapausyrityksessä, mutta ne eivät ole kiveen kirjoitettuja lopullisia toimintatapoja, koska tällöin toimittaisi Lean-ajattelun vastaisesti eikä annettaisi mahdollisuutta jatkuvaan parannukseen.

Taulukossa 5.1 on kuvattu Sysdronella käytössä olevia Lean-menetelmiä Leanin termein sekä merkitty, mitkä niistä ovat tulleet uutena Lean-ohjelmistokehityksen myötä. Ne, mitkä eivät ole tulleet uutena, ovat jo olleet tapausyrityksessä aiemmin käytössä. Tiedot on kerätty haastattelemalla yrityksen eri ihmisiä sekä tutkijan oman työkokemuksen ja havaintojen myötä kyseisessä yrityksessä. Tapausyrityksessä käytetään useista asioista eri nimitystä, koska useat käytänteet ovat olleet käytössä jo ennen Lean-ohjelmistokehityksen käyttöönottoa. Selvyyden vuoksi tässä tutkielmassa käytetään kuitenkin Leanistä peräisin olevia termejä. Taulukko sisältää ainoastaan tässä tutkielmassa esillä olleita työkaluja. Leanissa ja Lean-tuotannossa on olemassa näiden lisäksi paljon muitakin työkaluja, joihin voi perehtyä esimerkiksi tämän tutkielman lähteinä olevasta kirjallisuudesta.

Viisi miksi-kysymystä on käytössä aiemmin kuvatussa Jidoka-prosessissa. Lisäksi tätä voidaan hyödyntää missä tahansa tilanteessa, kun halutaan selvittää jonkun asian perimmäinen syy. Andon tarkoittaa Sysdronella käytännössä käännösnäyttöjä. Arvovirran tunnistaminen ja määrittäminen saattaa olla merkittävimpiä Leanin mukanaan tuomia uudistuksia. Se auttaa ymmärtämään kehitysprosessia asiakkaan näkökulmasta käsin. Genchi genbutsu toteutuu tapausyrityksessä siten, että

Taulukko 5.1: Tapausyrityksen käytössä olevat Lean-työkalut

Työkalu	**Käytössä**	**Uusi**
5 miksi-kysymystä	x	x
A3-raportit		
Arvovirran määrittäminen	x	x
Andon	x	
Genchi genbutsu	x	
Heijunka		
Jidoka	x	x
Kaizen	x	
Kanban	x	x
Poka-Yoke	x	
Tahtiaika ja läpimenoaika	x	x

tuotteen omistaja, joka vastaa projektista, istuu tiimin kanssa samassa tilassa eikä omassa toimistossaan. Kaizen toteutuu jälkipuintipalaverien muodossa, joista kerrottiin aiemmin. Kanban on kehittäjien kannalta yksi näkyvimpiä muutoksia päivittäisessä työssä. Kanbanista kerrottiin jo ylempänä. Poka-yoken näkyvin muoto ovat automaattiset hyväksymistestit. Tahtiaika ja läpimenoaika, jotka liittyvät läheisesti arvovirran määrittämiseen ja sen parantamiseen, kuvattiin myös aiemmin.

5.3 Yhteenveto

Tapausyrityksenä toimii Sysdrone Oy, joka on Jyväskyläläinen ohjelmistotalo. Sysdrone valmistaa asiakaskohtaisesti räätälöityjä ohjelmistoja, joiden painopiste on terveysteknologiassa.

Tapausyrityksessä otettiin kaikissa projekteissa käyttöön Lean-ohjelmistokehitys vuoden 2011 alusta alkaen. Tämä tarkoittaa, että projekteissa on käytössä seuraavat käytänteet: käyttäjäkertomukset, arvovirta, kanban, imuohjaus, kertomuspisteestimointi, läpimenoaika ja tahtiaika, jälkipuintipalaverit, käännösnäytöt, jidoka ja automaattinen hyväksymistestaus. Osa käytänteistä oli tapausyrityksessä uusia ja osa oli ollut jo aiemmin käytössä.

6 Aineisto ja menetelmät

Tutkimuksessa sovelletaan tapaustutkimusta. Siinä hyödynnetään useita tutkimusaineistoja, jotka on kerätty noin puolen vuoden aikajaksolta. Tulokset analysoidaan pääosin sisällönanalyysia ja sisällönerittelyä käyttämällä.

6.1 Tutkimuskysymykset

Kuten johdanto-luvussa jo käytiin läpi, tutkimuskysymykseksi muodostui *"Mitkä ovat merkittävimmät ohjelmistokehityksen nopeutta rajoittavat tekijät tutkittavassa yrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen?"*. Apukysymyksiä ovat *"Mitä Lean-ohjelmistokehitys on?"* ja *"Mitä Lean-ohjelmistokehitys tarkoittaa tutkittavassa yrityksessä?"*.

Ensimmäiseen apukysymykseen on annettu vastauksia kuvaamalla Leania ja Lean-ohjelmistokehitystä aiemmissa luvuissa tutkielman teoriaosassa. Seuraavan apukysymyksen asioita, eli Lean-ohjelmistokehitystä Sysdronella, käsiteltiin luvussa 5.2. Tutkielman loppuosa keskittyy varsinaiseen tutkimuskysymykseen vastaamiseen.

Ohjelmistokehityksen nopeutta rajoittavien tekijöiden tiedostaminen on tapausyritykselle tärkeää, jotta he voivat palvella asiakkaitaan paremmin. TOC-teorian mukaan rajoitteiden poistaminen nopeuttaa ”järjestelmää”. Tällöin asiakas saa tarvitsemansa ohjelmiston edullisemmin ja nopeammin. Ohjelmiston kehitykseen kuluva aika on usein kriittinen tekijä ohjelmistojen tilaajille, jotta he saavat oman tuotteensa markkinoille mahdollisimman nopeasti. Edullisemmat ohjelmistokehityskustannukset auttavat tapausyritystä myös tarjouskilpailuissa.

6.2 Aiemmat tutkimukset

Ohjelmistoprojektien menestystekijöistä ja käytänteistä on kirjoitettu paljon, mutta ohjelmistokehityksen esteitä, rajoitteita, pullonkauloja tai rajoitteiden tunnistamista ei ole juuri tutkittu. Aiheesta löytyi kaksi Ericssonin vuosina 2008 ja 2010 Göteborgissa teettämää tutkielmaa: *Asta Murauskaite ja Vaidas Adomauskas, Bottlenecks in Agile Software Development Identified Using Theory of Constraints (TOC) Principles*

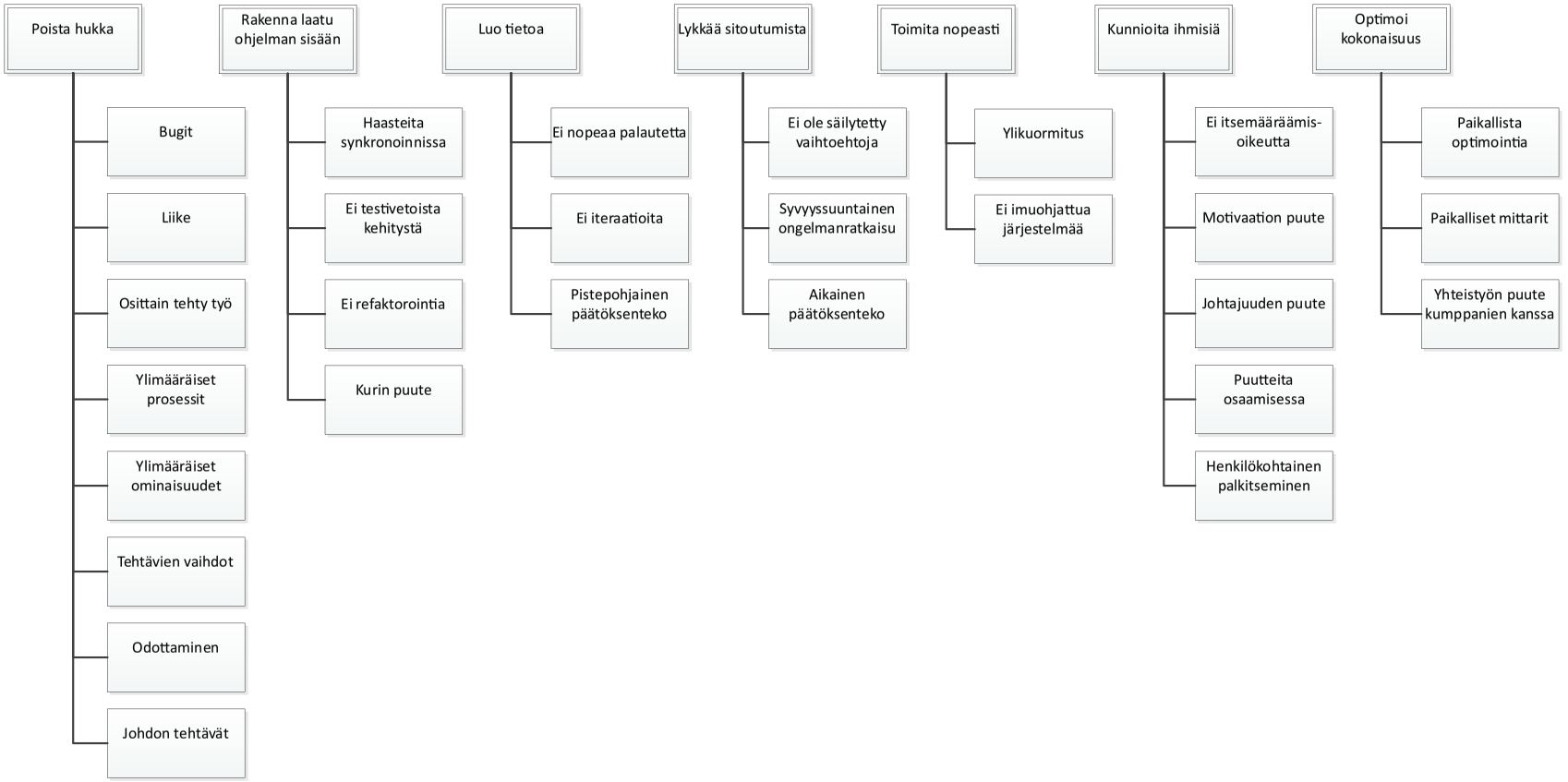
[MA08] sekä *Andrei Antanovich, Anastasia Sheyko ja Brian Katumba, Bottlenecks in the Development Life Cycle of a Feature - A Case Study Conducted at Ericsson AB* [ASK10].

Vuoden 2008 tutkielmassa on kehitetty teoreettinen malli Lean-ohjelmistokehityksen pullonkaulojen tunnistamiseksi. Malli pohjautuu Eliyahu M. Goldrattin kehittämään esteiden teoriaan (Theory of Constraints, TOC).

Murauskaite ja Adomauskas ovat tunnistaneet viisi mahdollista korkean tason pullonkaulaa Lean-ohjelmistokehityksessä tutkimalla, mihin ketterän kehityksen periaatteisiin Leanissä kiinnitetään vain vähän tai ei ollenkaan huomiota. Nämä mahdolliset pullonkaulat ovat:

1. yksilöiden motivaation puute
2. tiheän kanssakäynnin puute sidosryhmien kanssa
3. ei kasvotusten kommunikointia
4. jatkuvuuden puute (kehitysnopeus vaihtelee)
5. yksinkertaisuuden puute

Näiden lisäksi tutkielmassa on tunnistettu matalammalla tasolla mahdollisia pullonkauloja. Kyseiset pullonkaulat ovat nähtävissä kuvassa 6.1. Tässä Murauskaite ja Adomauskas ovat ottaneet lähtökohdaksi Poppendieckien esittämät seitsemän Lean-ohjelmistokehityksen perusperiaatetta (ks. luku 4), jotka ovat kaavion ylärivillä. Oletuksena on, että näiden periaatteiden noudattamatta jättämisestä voi syntyä kaaviossa esiintyviä pullonkauloja. Jokaisen periaatteen alla on siihen liittyvät mahdolliset pullonkaulat. Mallissa myös oletetaan, että Poppendieckien näkemys Lean-ohjelmistokehityksestä on sellaisenaan täysin toimiva.



Kuva 6.1: Mahdollisia pullonkauloja Lean-ohjelmistokehityksessä [MA08]

Ajanpuutteen vuoksi Murauskaite ja Adomauskas valitsivat mahdollisista pullonkauloista ainoastaan seitsemän ehdokasta, joihin he keskittyivät haastatteluissaan. Näistä seitsemästä ehdokkaasta he tunnistivat kuusi mahdollista pullonkaulaa Ericssonilla tutkimassaan yksikössä. Tulokset ovat nähtävissä taulukossa 6.1

Taulukko 6.1: Murauskaiteen ja Adomauskasin tunnistamat pullonkaulat Ericssonilla tutkimassaan yksikössä [MA08]

Lean-perusperiaate	**Lean-pullonkaula**	**Mahdollinen pullonkaula Ericssonilla tutkitussa yksikössä**
Poista hukka	Odottaminen	Tuotepäälliköiden käyttämä aika alhaisen prioriteetin ominaisuuksien arviointiin ja dokumentointiin.
Rakenna laatu ohjelman sisään	Haasteita synkronoinnissa	Testauksen läpimenoaika.
Luo tietoa	Ei nopeaa palautetta	Virallisen suunnittelija–suunnittelija-kommunikointiprosessin puuttuminen.
Lykkää sitoutumista	Aikainen päätöksenteko	-
Toimita nopeasti	Ylikuormitus	Testausproseduurit vievät enemmän aikaa mitä testausresurssit voivat käsitellä.
Kunnioita ihmisiä	Puutteita osaamisessa	Erilaiset eivätkä täysin yhteensopivat osaamismallit organisaation eri yksiköissä.
Optimoi kokonaisuus	Paikallista optimointia	Globaalien mittareiden läpinäkyvyys.

Kyseisistä tuloksista on hankala sanoa kuinka suuren prioriteetin pullonkauloja ne oikeasti ovat, koska mahdollisista Lean-pullonkauloista esivalittiin seitsemän ja kysymyksenasettelu oli tutkimuksessa sikäli jossain määrin rajattu. Ei voida tietää olisiko muista mahdollisista Lean-pullonkaulaehdokkaista löytynyt joitain suurempia pullonkauloja. Tämän tutkielman tekijät itsekin tekstissään toteavat.

Göteborgissa vuonna 2010 tehty uudempi tutkielma [ASK10] lähestyy asiaa hieman eri kantilta. Siinä tutkitaan yksittäisen ominaisuuden elinkaaressa esiintyviä pullonkauloja. Tämä tutkimus on tehty haastattelemalla eri sidosryhmiä. Kysymyksenasettelu on ollut avoin: ”Mitkä ovat mahdolliset pullonkaulat ominaisuuden ke-

hityksen elinkaaren aikana?”. Tutkimuksessa todettiin seuraavat suurimmiksi pullonkauloiksi:

1. **Tehtävien vaihdot** *(engl. task switching).* Lähes kaikki vastaajat totesivat tämän pullonkaulaksi.

2. **Ammatillinen osaaminen.** Uusien teknologioiden oppiminen vie aikaa, jolloin osaamisen ja kokemuksen puute muodostuu pullonkaulaksi.

3. **Palaute.** Palautteen saaminen tiimin ulkopuolelta kestää joskus pitkään, mikä voi aiheuttaa odottamista.

4. **Johdon reagointi parannusehdotuksiin.** Projektien lopussa esitetään parannusehdotuksia, mutta johto ei aina regoi näihin. Tästä seuraamuksena samat ongelmat toistuvat seuraavissakin projekteissa.

5. **Kommunikointiketju.** Tiimien, johdon ja asiakkaiden välillä oli pitkä hierarkinen kommunikointiketju, mistä voi aiheutua tiedon muuttumista tai katoamista.

6. **Ymmärrys ominaisuuden kontekstista.** Kehittäjät eivät aina ymmärtäneet ominaisuuden tarkkaa kontekstia, minkä takia optimaalista toteutustapaa tai -teknologiaa ei välttämättä osattu valita.

7. **Ymmärrys kehitysprosessista.** Mikäli ei ymmärrä kehitysprosessia, on siihen melkein mahdotonta esittää parannusehdotuksia.

8. **Dokumentointi.** Dokumentointi tekee kehitysprosessista helpommin ymmärrettävän ja ohjelmistosta helpommin ylläpidettävän. Sellaisen dokumentaation kirjoittaminen, mitä kukaan ei lue, on kuitenkin hukkaa. Dokumentoinnin osalta tulisi löytää oikea tasapaino, muutoin se voi muodostua pullonkaulaksi.

Jos Antanovichin ja Sheykon tutkimustuloksia vertaa Murauskaiteen ja Adomauskasin tunnistamiin mahdollisiin Lean-ohjelmistokehityksen pullonkauloihin (kuva 6.1), huomataan, että useimmat niistä löytyvät kuvasta. Ainoastaan ”ymmärrys kehitysprosessista” ja ”dokumentointi” ovat sellaisia, joille kaaviosta ei löydy suoraa vastinetta. ”Ymmärrys kehitysprosessista” kuuluisi joko ”rakenna laatu ohjelman sisään” tai ”optimoi kokonaisuus” -perusperiaatteen alle. ”Dokumentointi” taas kuuluisi todennäköisesti ”luo tietoa” -periaatteen alle. Antanovich ja Sheyko eivät harmillisesti itse viittaa tekstissään aiempaan Ericssonin teettämään tutkielmaan, joten heidän osaltaan ei tulosten vertailua kyseiseen malliin ole saatavilla.

Tässä tutkimuksessa ei käytetä suoraan Murauskaiteen ja Adomauskasin mallia, koska se rajaa rajoitteiden tai pullonkaulojen löytämisen heidän tunnistamiinsa mahdollisiin pullonkauloihin. Antanovichin ja Sheykon tuloksista on nähtävissä, että pullonkauloja voi löytyä mallin ulkopuoleltakin. Vaikka kyseistä mallia ei käytetä, tutkielman lopussa verrataan tuloksia esitettyyn malliin.

6.3 Tutkimusaineistot

Tutkimuksen aineistoa kerättiin välillä tammikuu 2011 - kesäkuu 2011. Käytettyä aineistoa on kahdenlaista: ohjelmistokehitystiimien jälkipuintipalaverien pöytäkirjoja sekä vaatimuksista kerättyä mittausdataa niiden eri työvaiheisiin kuluvasta ajasta. Lisäksi hyödynnettiin tapausyrityksessä jo aiemmin kertynyttä vaatimusten mittausdataa väliltä kesäkuu 2010 - joulukuu 2010.

Tutkittavia tiimejä on kolme. Jokainen tiimi koostuu 2-4 kehittäjästä, joista yksi toimii tiiminvetäjänä. Tiimien kokoonpanot ja koot vaihtelivat hieman tutkimuksen aikana johtuen projektien vaihtumisesta. Pääsääntöisesti tiimit kuitenkin pysyivät alkuperäisessä kokoonpanossaan. Tutkimuksessa ei erotella eri tiimejä toisistaan, vaan näitä kolmea tiimiä käsitellään yhtenä kokonaisuutena.

6.3.1 Jälkipuintipalaverien pöytäkirjat

Jälkipuintipalavereita esiteltiin hieman jo luvussa 5.2. Ne ovat siis palavereita joita pidetään iteraatioiden välillä, mikä tarkoittaa tapausyrityksessä keskimäärin kahden viikon välein. Tiimi pitää palaverit itsenäisesti, eikä niihin osallistu asiakasta tai yrityksen johtoa. Palaverien kesto on yleensä puolesta tunnista tuntiin, kuitenkin siten, että kesto on etukäteen määrätty ja sitä ei ylitetä. Palaverien tarkoituksena on parantaa käytänteitä, prosessia tai itse kehityksen alla olevaa tuotetta. Palavereissa tarkastellaan päättynyttä iteraatiota. Palavereilla ei ole kiinteää agendaa, mutta yleisimmin palaverin vetäjä esittää kysymykset:

1. Mitä teimme hyvin edellisessä iteraatiossa?

2. Missä jäi parantamisen varaa?

Palaveriin osallistujat voivat vapaasti heittää ilmoille vastauksia näihin kysymyksiin. Tämän jälkeen esitetyt vastaukset priorisoidaan ja niitä lähdetään käymään läpi. Oleellista on löytää ”Missä jäi parantamisen varaa” -otsikon alle tulleisiin kohtiin konkreettisia toimenpiteitä, joilla löytyneet epäkohdat voidaan korjata

tai käytänteitä pystytään parantamaan. Nämä löytyneet korjaavat toimenpiteet tulisi ottaa kokeiluun välittömästi palaverin jälkeen. Seuraavassa palaverissa katsotaan onko kokeilu ollut toimiva vai ei. Ei-toimivat käytänteet luonnollisesti hylätään.

Jokaisesta jälkipuintipalaverista tehdään lyhyt määrämuotoinen pöytäkirja, joka sisältää ranskalaisilla viivoilla palaverin pääkohdat. Aineiston salaisuudesta johtuen aineistoa ei esitetä tässä tutkielmassa sellaisenaan, vaan ainoastaan koontina. Pöytäkirjojen rakenne on seuraava:

1. viime kerran asioiden seuranta
2. keskustellut aiheet
3. päätökset

Kyseessä on siis puhtaasti laadullinen, eli kvalitatiivinen, aineisto. Tutkimuksessa oletetaan, että palavereissa esille otetut ”Missä jäi parantamisen varaa” -aiheet, ovat niitä, joihin kehityksen esteet ja rajoitteet sillä hetkellä liittyvät.

6.3.2 Vaatimusten eri työvaiheisiin käytetty aika

Tapausyrityksessä oli jo puoli vuotta ennen tutkimuksen aloittamista otettu yhden tiimin osalta kokeiluun käytäntö mitata jokaisen vaatimuksen osalta kuhunkin työvaiheeseen käytetty aika. Tutkimuksen alusta lähtien kokeilua laajennettiin kaikkiin kolmeen tiimiin, jotka olivat tutkimuksessa mukana. Käytännössä aikojen mittaus tarkoittaa, että kunkin työvaiheen alkaessa merkitään ylös työvaiheen aloitusaika ja työvaiheen loppuessa lopetusaika. Ajat eivät ota kantaa tiimin kokoon. Tässä tapauksessa ei siis mitata käytettyjen työtuntien määrää, vaan kellonaikoja, mikä on lähempänä Lean-ajattelumallia. Kyseessä on määrällinen eli kvantitatiivinen aineisto.

Vaatimusten eri työvaiheet ovat:

1. **Uusi vaatimus (asiakkaalta).** Asiakkaalta saadaan uusi vaatimus esimerkiksi sähköpostitse tai palaverissa ja se kirjataan tuotteen työlistaan.
2. **Vaatimuksen analysointi.** Vaatimus, eli käyttäjäkertomus, pilkotaan tehtäviksi. Tässä työvaiheessa suunnitellaan myös kuinka vaatimus tullaan testaamaan sekä hahmotellaan mahdollinen käyttöliittymäluonnos. Lisäksi työvaiheessa piirretään tarpeen vaatiessa esimerkiksi luokkakaavioita tai muita malleja ohjelmiston rakenteesta.

3. **Kehitys.** Vaatimus toteutetaan eli tämä työvaihe sisältää pääasiassa ohjelmointia.

4. **Verifiointi.** Vaatimus testataan. Suurin osa testauksesta pyritään tekemään automaattitestauksella. Tämä tarkoittaa, että tässä työvaiheessa kirjoitetaan automaattitestit analyysivaiheessa laaditun testaussuunnitelman mukaisesti. Tämän lisäksi suoritetaan mahdolliset manuaaliset hyväksymistestit vaatimukselle.

5. **Käyttöönotto.** Vaatimus otetaan käyttöön projektista riippuen joko asiakkaan testaus- tai tuotantoympäristöön.

Työvaiheisiin käytettyjen aikojen perusteella voidaan vaatimuksille laskea tahtiaika ja läpimenoaika, joita käytiin läpi luvussa 5.2. Tahtiajan ja läpimenoajan suhteet edellä esitettyihin työvaiheisiin ilmenevät kuvasta 6.2. Kuvan määritelmät ovat tapausyrityksessä tehtyjä. Tässä tutkimuksessa käytetään näitä määritelmiä, mutta on hyvä huomata, että joissain lähteissä määritelmät saattavat olla hieman erilaiset. Epäyhtenäisyys selittyy sillä, että kyseiset termit ovat Toyotan tuotantojärjestelmän peruja eivätkä sovellu kovin hyvin ohjelmistokehitykseen alkuperäisessä muodossaan.

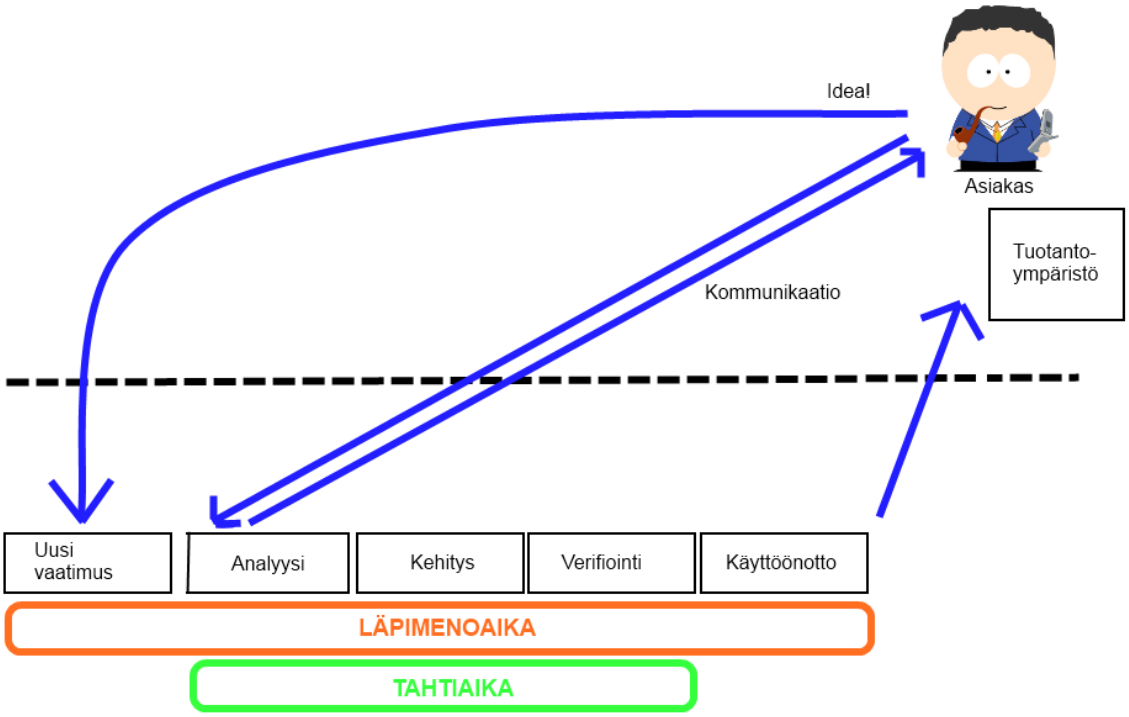
6.3.3 Muut aineistot

Aiemmin esitettyjen aineistojen lisäksi tutkija on pitänyt tutkimuksen aikana eräänlaista tutkimuspäiväkirjaa, johon on kirjattu ylös hänen tekemiään havaintoja. Koska tutkija työskentelee tapausyrityksessä, hänellä on ollut mahdollisuus tarkkailla toimintaa ja tehdä huomioita mahdollisista rajoitteista. Tutkimuspäiväkirjaa ei käydä yksityiskohtaisesti läpi tässä tutkielmassa, vaan se pikemminkin tukee muuta aineistoa. Aineisto sisältää joitain yrityssalaisuuden piiriin luokiteltavia asioita, joten itse aineistoa ei julkaista sellaisenaan.

6.4 Aineiston analysointimenetelmät

Tutkimuksen kaksi pääaineistoa analysoidaan molemmat erikseen toisistaan riippumatta. Tämän jälkeen tulosten perusteella tehdään johtopäätöksiä rajoittavista tekijöistä.

Jälkipuintipalaverit analysoidaan käyttäen sisällönanalyysia ja sisällönerittelyä. Sisällönanalyysi on tekstianalyysia, jossa aineisto voi olla monenlaista: kirjoja, puheita, muistiinpanoja, puheluita ja raportteja. Aineisto tulee kuitenkin muuttaa teks-



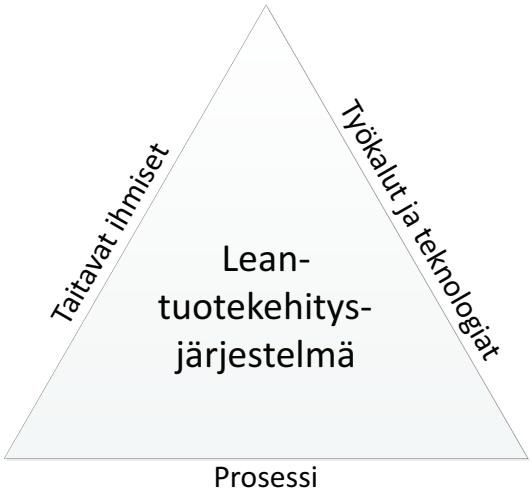
Kuva 6.2: Tapausyrityksen agile coachin näkemys läpimenoajan ja tahtiajan suhteista vaatimusten eri työvaiheisiin tapausyrityksessä

timuotoiseksi, mikäli se on alun perin jossain muussa muodossa (esimerkiksi puheet). ”Sisällönanalyysissä aineistoa tarkastellaan eritellen, yhtäläisyyksiä ja eroja etsien ja tiivistäen.” [SKP06]

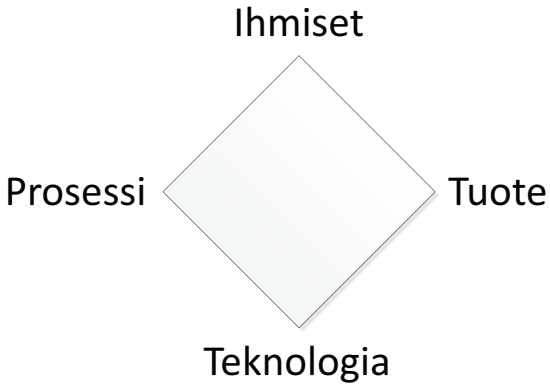
Sisällönerittelyllä tarkoitetaan kvantitatiivista aineiston analyysia, jossa kuvataan määrällisesti jotakin tekstin tai dokumentin sisältöä [TS02]. Joskus puhutaan myös sisällönanalyysista, kun tarkoitetaan sisällönerittelyä, mutta Saaranen-Kauppisen ja Puusniekan [SKP06] mukaan sisällönanalyysillä tarkoitetaan nimenomaan sanallista tekstin sisällön kuvailua ja sisällönerittelyllä aineiston jonkun sisällön määrällistä kuvausta.

Ideana on tarkastella jälkipuintipalavereja ensin käyttäen sisällönerittelyä. Tavoitteena on kategorisoida tietoa ja löytää yksi tai kaksi kategoriaa, joihin suurimmat ohjelmistokehitystä rajoittavat tekijät mahdollisesti liittyvät. Kun nämä kategoriat ovat selvillä, analysoidaan niitä tarkemmin käyttäen sisällönanalyysia. Tämä tarkoittaa, että pöytäkirjoja käydään kyseisten kategorioiden osalta kohta kohdalta läpi ja pyritään analysoimaan, mitkä varsinaiset rajoittavat tekijät ovat. Näihin kategorioihin liittyvä tieto lajitellaan myös alakategorioihin, mikäli tällaisia on selkeästi löydettävissä ja mikäli se todetaan tarpeelliseksi analyysin kannalta.

Tutkimuksen alkuvaiheessa sisällönerittelyä varten luotiin valmiiksi pääkategoriat, joiden alle tietoa lähdettiin lajittelemaan. Morgan ja Liker [ML06] esittävät, että Lean-tuotekehitysjärjestelmä koostuu kolmesta pääosa-alueesta: taitavista ihmisistä, työkaluista ja teknologioista sekä prosessista. Tämä on mallinnettu kuvassa 6.3. Mielenkiintoisesti Steve McConnell on lähes samoilla linjoilla pohtiessaan, mistä ohjelmistoprojektin kehitysnopeus muodostuu. Hän kuvaa neljä ulottuvuutta, jotka ovat ihmiset, prosessi, tuote ja teknologia (kuva 6.4).



Kuva 6.3: Lean-tuotekehitysjärjestelmä [ML06]



Kuva 6.4: Kehittämisnopeuden neljä ulottuvuutta [McC02]

Kuten kuvista nähdään, Morganin ja Likerin sekä McConnelin näkemykset ovat käytännössä samat lukuun ottamatta ”Tuote”-ulottuvuutta, joka löytyy McConnellin mallista. Ohjelmistokehitys on tuotekehityksen erikoistapaus [PP08, s. 12], joten mallien samankaltaisuus on sikäli luonnollista.

Näiden kahden näkemyksen perusteella oli selkeää valita kolme pääkategoriaa: ihmiset, prosessi ja teknologia. Tutkimuksen kategorioihin valittiin mukaan myös McConnellin neljäs ulottuvuus, eli tuote. Sen valinta oli hieman hankalammin perusteltavissa, koska se puuttuu toisesta mallista. McConnellin perustelut tälle ulottuvuudelle ovat kuitenkin järkevät. Alla on kerrottuna millaisia asioita eri kategorioihin kuuluu, ja siitä ilmenee myös, mitä tuote-kategorialla tarkoitetaan.

- **Ihmiset.** ”Henkilöstoasiat vaikuttavat ohjelmistotuottavuuteen ja ohjelmiston laatuun enemmän kuin mikään muu asia” [McC02, s.12–13]. Ihmiset-kategorian alle liittyvät kaikki henkilöstöön, motivaatioon, tiimityöhön, työilmapiiriin, osaamiseen, johtamiseen tai koulutukseen liittyvät tekijät. McConnelin mukaan ihmisiin liittyvät asiat vaikuttavat tuottavuuteen enemmän kuin muut kategoriat yhteensä, eli ihmisiin liittyvät rajoitteet tai esteet ovat myös kaikista kriittisimpiä. Esimerkkejä rajoittavista tekijöistä voisivat olla huono työilmapiiri tai puutteet osaamisessa.

- **Prosessi.** Tähän kategoriaan liittyvät mm. ketterän kehityksen käytänteet, riskienhallinta ja työn toiston välttäminen. Tämän kategorian alle liittyvät rajoitteet voisi mieltää olevan ongelmia kehityskäytänteissä ja toimintatavoissa. Leanin termeillä prosessi on se, mitä katsoo, kun tekee arvovirtakarttaa [ML06]. Esimerkki tämän kategorian rajoitteesta voisi olla, että uuden vaatimuksen käyttöönotto tarvitsee esimiehen hyväksynnän, mutta hän on vaikeasti saavutettavissa.

- **Tuote.** Tuotteeseen pätee 80/20-sääntö: 80 prosentin tekemiseen menee vain 20 prosenttia ajasta [McC02, s.17]. Jos tuotteen ominaisuuslista on joustava, voidaan ensin toteuttaa kyseiset 80 prosenttia tuotteesta ja loput myöhemmin, mikäli ne katsotaan yhä tarpeellisiksi ominaisuuksiksi. Jos tuotteen osalta tehdään vääriä valintoja, voi se muodostua pullonkaulaksi. Esimerkkinä, että keskitytään pienen prioriteetin ominaisuuksiin, ja tämän takia ei pystytä tekemään korkean prioriteetin ominaisuuksia, joita asiakas oikeasti tarvitsisi.

- **Työkalut ja teknologia.** Vanhentuneet, hankalat tai huonosti toimivat teknologiat voivat muodostua rajoitteeksi. Esimerkiksi, jos nykyisin kehittäisi sovellusta assemblerilla vaikkapa Javan sijaan, olisi ohjelmointikieli kehitysnopeutta rajoittavana tekijänä. Ohjelmistotyökalujen lisäksi tämä kategoria pitää sisällään ns. ”pehmeät” työkalut, joita ovat esimerkiksi viisi miksi-kysymystä ja tehtävätaulu.

Vaatimusten eri työvaiheisiin käytetyt ajat analysoidaan käyttäen kumulatiivista vuokaaviota *(engl. cumulative flow diagram)* sekä arvovirtakarttoja. Kumulatiivinen vuokaavio ilmaisee, kuinka paljon vaatimuksia on työn alla lukumääräisesti kussakin työvaiheessa milläkin ajanhetkellä. Arvovirtakartoista taas nähdään, kuinka paljon vaatimuksen työstöön käytetään aikaa kussakin työvaiheessa ja kuinka pitkään vaatimus odottaa työvaiheiden välillä. Aikojen mittauksesta saadaan selville, missä kohtaa prosessia rajoitteet ovat, mutta ei varsinaisia syitä näille. Jälkipuintipalaverien pöytäkirjat sekä tutkimuspäiväkirja auttavat löytämään syitä prosessin rajoitteille.

6.5 Yhteenveto

Tutkimuksen tarkoituksena on löytää, mitkä ovat tapausyrityksessä suurimmat ohjelmistokehitysnopeutta rajoittavat tekijät puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Ericsson on teettänyt vastaavasta asiasta, eli ohjelmistokehityksen pullonkauloista, kaksi tutkimusta vuosina 2008 ja 2010 Göteborgissa.

Tutkimusaineistona ovat jälkipuintipalaverien pöytäkirjat sekä tiedot vaatimusten eri työvaiheisiin käytetystä ajasta. Lisäksi tutkija tehnyt merkintöjä omista havainnoistaan tutkimuksen aikana.

Jälkipuintipalaverien pöytäkirjojen ”Missä jäi parantamisen varaa?” -kohdat analysoidaan käyttäen sisällönanalyysia ja sisällönerittelyä. Sisällönerittely on aineiston jonkun sisällön määrällistä kuvausta ja sisällönanalyysilla tarkoitetaan aineiston tekstimuotoista kuvausta. Pöytäkirjojen kohdat lajitellaan neljän pääkategorian (ihmiset, prosessi, teknologia ja tuote) alle. Näistä valitaan yksi tai kaksi kategoriaa joita analysoidaan käyttäen sisällönanalyysia. Lisäksi näille kategorioille muodostetaan tutkimuksen aikana alakategorioita, mikäli näitä on selvästi tunnistettavissa ja ne osoittautuvat tarpeelliseksi.

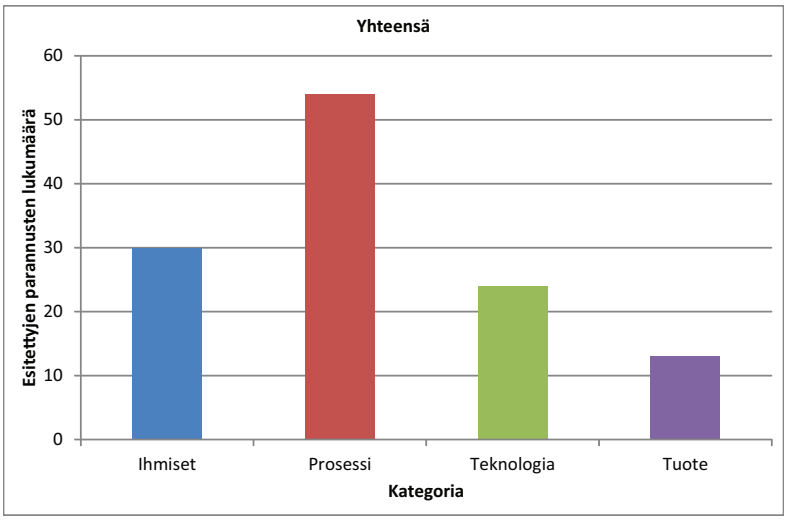
Vaatimusten eri työvaiheisiin käytetyistä aikatiedoista muodostetaan kumulatiivinen vuokaavio. Kaaviosta on mahdollista nähdä, mitkä työvaiheet ovat milloinkin olleet rajoittavina tekijöinä. Tulokset kertovat rajoittavana tekijänä olevan prosessin työvaiheen, mutta ei varsinaista syytä tälle. Itse syitä etsitään jälkipuintipalaverien pöytäkirjoja analysoimalla.

7 Tutkimustulokset ja niiden analysointi

Tutkimustulokset analysoidaan käyttäen luvussa 6 kuvattuja menetelmiä. Jälkipuintipalaverien pöytäkirjoista ja vaatimusten mittausdatasta saadut tulokset käsitellään molemmat itsenäisesti toisistaan riippumatta.

7.1 Tutkimustulokset jälkipuintipalaverien pöytäkirjojen perusteella

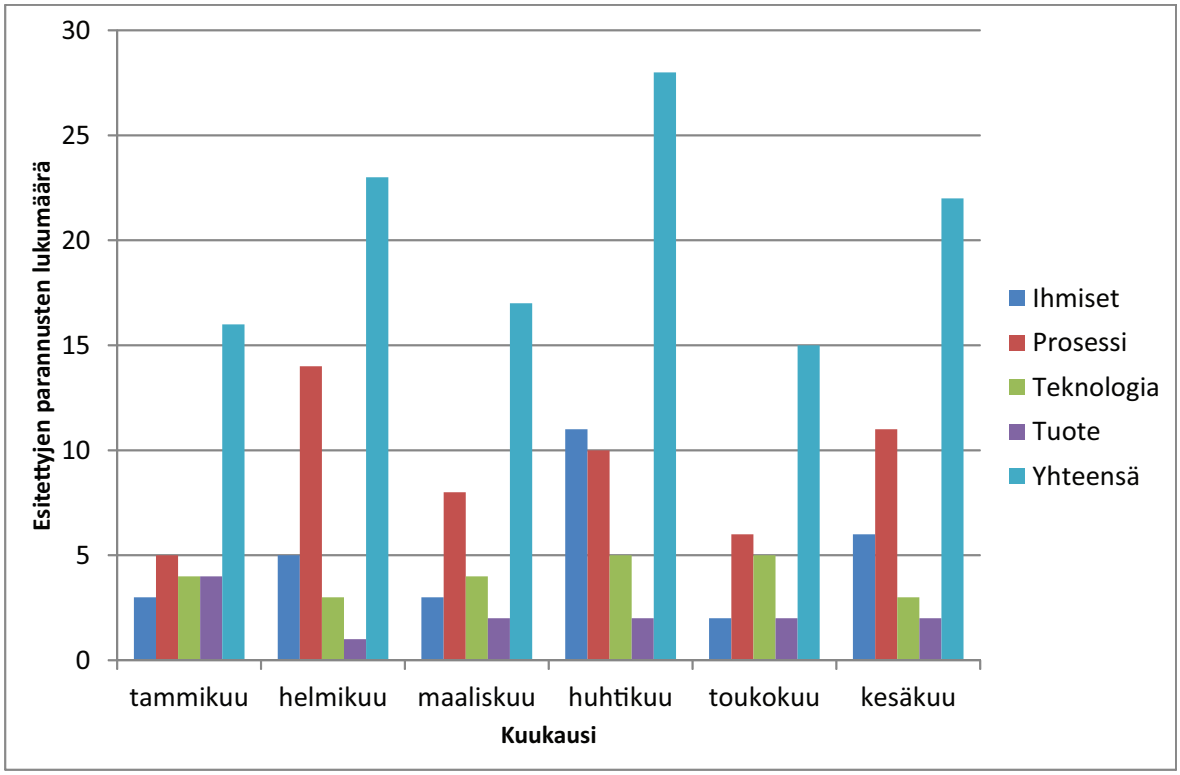
Kuvassa 7.1 nähdään kategorioittain jaoteltuina, missä asioissa tapausyrityksen tiimit ovat löytäneet eniten parannettavaa. Tulokset ovat peräisin tutkimuksen aikana pidettyjen iteraatioiden jälkipuintipalaverien pöytäkirjoista. Kaavion pystyakselilta ilmenee, kuinka monta kyseiseen kategoriaan kuuluvaa parannusta on lukumäärällisesti esitetty. Kategoriat löytyvät vaaka-akselilta. Kaaviosta havaitaan, että tiimit ovat pyrkineet parantamaan prosessia selkeästi eniten. Muut kategoriat, etenkin tuote, ovat jääneet vähemmälle huomiolle.



Kuva 7.1: Jälkipuintipalaverien ”parannettavaa”-aiheet yhteensä

Kuvassa 7.2 näkyy tulokset jaoteltuna kuukausittain. Eri kuukaudet vaikuttaisivat olevan jokseenkin toistensa kaltaisia lukuun ottamatta huhtikuuta, jossa ihmisiin liittyviin asioihin on löydetty eniten parannettavaa. Helmi- ja kesäkuussa prosessiin on esitetty selkeästi eniten parannuksia muihin kategorioihin verrattuna. Eri

kuukausina on ollut eri määrä jälkipuintipalavereita, joten sikäli kuukaudet eivät ole suoraan vertailukelpoisia toistensa kanssa. Kaavio antaa kuitenkin kuvaa siitä, missä suhteessa eri kategorioista on löytynyt parannettavaa missäkin kuussa muihin kategorioihin nähden.



Kuva 7.2: Jälkipuintipalaverien ”parannettavaa”-aiheet

Lähes ainoa, mitä kuvista 7.1 ja 7.2 voidaan varmasti tulkita, on se että prosessista on keskusteltu selkeästi eniten. Selkeää syytä tälle ei tutkimuksen aineiston perusteella voida sanoa. Joka tapauksessa, koska prosessit ovat olleet eniten esillä, oli se looginen valinta tarkempaa sisällönanalyysia ja -erittelyä varten. Prosessin lisäksi ihmiset-kategoria oli myös toiseksi suurimpana selkeä valinta. Esimerkiksi McConnel on todennut: ”Yhteen laskettuna henkilöstöasiat merkitsevät enemmän kuin prosessi, tuote ja teknologia yhdessä” [McC02, s.13].

7.1.1 Prosessi

Prosessit ovat Lean-ajattelussa merkittävässä roolissa. Tutkimalla jälkipuintipalaverien aineistoa tarkemmin prosesseihin liittyvien keskustelunaiheiden osalta, aiheille löytyivät seuraavat alakategoriat:

- **Asiakasriippuvuus:** Riippuvuus asiakkaasta tai asiakkaan prosesseista, mistä on seurannut usein odottamista tai ylimääräistä työtä.
- **Puutteita prosessin noudattamisessa:** Ei ole täysin noudatettu sovittuja prosesseja, mistä on seurannut haasteita, odottamista tai ylimääräistä työtä.
- **Tiedonpuute:** Puuttunut tarvittavaa (yrityksen sisäistä) tietoa, mikä on yleensä hidastanut tekemistä tai aiheuttanut odottamista.
- **Uudet ehdotukset:** Täysin uudet ehdotukset ohjelmistokehitysprosessin parantamiseksi. Näihin ei sinällään liity suoranaisia ongelmia, vaan ne ovat esille tuotuja uusia ideoita, joilla kehitysnopeutta tai -laatua pystyttäisiin mahdollisesti parantamaan.
- **Verifiointi:** Verifiointiin liittyvät haasteet tilanteissa, joissa sovittua prosessia on noudatettu.
- **Muut** Muut edellä mainittuihin kategorioihin kuulumattomat ongelmat.

Kun aineiston prosesseihin liittyvät aiheet lajitellaan sisällönerittelyä käyttäen yllä mainittuihin alakategorioihin, saadaan tulokseksi lukumäärät, jotka näkyvät taulukossa 7.1.

Taulukko 7.1: Prosessi-kategorian ”parannettavaa”-aiheet alakategorioittain jaoteltuina

Alakategoria	**Lukumäärä**
Uudet ehdotukset	13
Verifiointi	13
Puutteita prosessin noudattamisessa	9
Tiedonpuute	8
Asiakasriippuvuus	8
Muut	3

Uusia ehdotuksia ja verifiointiin liittyviä haasteita on ollut eniten. Aineisto on kuitenkin jakautunut varsin tasaisesti eri kategorioiden välille, joten mitään yksittäistä selkeää kehitysnopeutta rajoittavaa tekijää ei ole suoraan havaittavissa.

Verifiointiin liittyvät aiheet ovat olleet varsin erilaisia. Automaattiset hyväksymistestit ovat olleet esillä useampaan otteeseen, ja niissä ongelmana ovat lähes aina

olleet puutteet tai erilaisuudet testidatassa tuotantoon nähden. Myös testien automatisoinnin hankaluutta tietyissä tilanteissa on tuotu ilmi. Manuaalisiakin (ihmisten tekemiä) hyväksymistestejä on käsitelty muutamaan otteeseen, ja näissä ongelmana on ollut testien määrän kasvaminen liian suureksi, jolloin ne helposti menettävät merkityksensä. Lisäksi on useita muita yksittäisiä aiheita.

Prosessin noudattamatta jättämiset ovat olleet pieniä yksittäisiä asioita, jotka ovat yleensä vähentäneet tekemisen järjestelmällisyyttä ja mahdollisesti myös kehitysnopeutta tai kehityksen laatua. Esimerkkejä tällaisista asioista ovat iteraation burndown-kaavion puuttuminen, käännösnäytön puuttuminen tai epäselvyydet iteraation työlistassa.

Tiedon puute on tarkoittanut käytännössä dokumentoimattomia asioita tai niin sanottua hiljaista tietoa. Näistä on ollut seurauksena tekemisen hidastuminen tai odottaminen. Esimerkkejä tällaisista asioista ovat tietokannan taulujen dokumentoimattomuus tai ohjeiden puute ohjelmiston versionumeroinnin kasvattamisesta tietyssä tilanteessa.

Asiakasriippuvuus-alakategorian alla olevat asiat ovat olleet tilanteita, joissa on jouduttu odottamaan asiakasta tai joissa asiakkaan prosessit ovat hankaloittaneet tekemistä. Esimerkiksi on puuttunut tiettyjä dokumentteja tai on jouduttu odottamaan asiakkaan käyttöliittymäkatselmointia.

Koska aineisto on jakautunut varsin tasaisesti eri alakategorioiden kesken, on varsin hankala sanoa, mikä näistä alakategorioista on kehitysnopeutta eniten rajoittava tekijä. Todennäköisimmin se on suurin alakategoria eli verifiointi, koska se on tuotu useimmin esille, mutta varmasti tätä ei pysty aineiston pienen koon vuoksi sanomaan. Tunnistetut viisi alakategoriaa auttavat kuitenkin hahmottamaan, mille asioille on todettu löytyvän parantamisen mahdollisuuksia tutkimusjakson aikana.

7.1.2 Ihmiset

Ihmisiin liittyviä ”parannettavaa”-aiheita oli tuotu jälkipuintipalavereissa tutkimusjakson aikana toiseksi eniten esille. Yhtä kokeneiden ohjelmoijien tuottavuusero voi olla kymmenkertainen [McC02, s.12–13], joten kategoriaa ei sovi aliarvioida. Todennäköisesti sen merkitys kehitysnopeuteen on suurempi kuin prosessi-kategorian, sillä kuten tässä tutkielmassa aiemmin mainittua, tämän kategorian merkitys on McConnellin [McC02, s.12–13] mukaan suurempi kuin muiden kategorioiden yhteensä.

Ihmisiin liittyvistä jälkipuintipalaverien keskustelunaiheista oli tunnistettavissa seuraavat alakategoriat:

- **Osaaminen / koulutus:** Kokemattomuutta tai puutteita osaamisessa useimmiten uuden teknologian kanssa. Uusien teknologioiden oppiminen vie aikaa, mikä hidastaa tekemistä.
- **Projektin johtaminen:** Puutteita projektin johtamisessa, mikä aiheuttaa epätietoisuutta tai epävarmuutta tekemiseen.
- **Henkilöiden kuormitus:** Henkilöillä liian suuri tai pieni kuormitus. Esimerkiksi joillain henkilöillä liikaa kuormaa, mikä aiheuttaa viiveitä toisaalle.
- **Asiakaskommunikointi:** Puutteita asiakaskommunikoinnissa.
- **Inhimillinen erehdys:** Kehitystyössä tapahtunut inhimillinen virhe.
- **Motivaatio:** Esille tuodut motivaatiota heikentävät asiat.
- **Muut:** Muut edellä mainittuihin kategorioihin kuulumattomat ongelmat.

Taulukossa 7.2 on nähtävillä eri alakategorioihin liittyvien aiheiden lukumäärät. Kuten taulukosta havaitaan, on ihmiset-kategoriassa tunnistettavissa selkeä tekemisen nopeutta rajoittava tekijä: osaaminen / koulutus. Myös projektin johtaminen ja henkilöiden kuormitus on herättänyt keskustelua.

Taulukko 7.2: Ihmiset-kategorian ”parannettavaa”-aiheet alakategorioittain jaoteltuina

Alakategoria	**Lukumäärä**
Osaaminen / koulutus	11
Projektin johtaminen	6
Henkilöiden kuormitus	5
Asiakaskommunikointi	2
Inhimillinen erehdys	2
Motivaatio	2
Muut	2

Aineistoa tarkastelemalla kävi ilmi, että lähes kaikki osaaminen / koulutus -alakategorian esille tuodut ”parannettavaa” -aiheet liittyvät käyttöönotettuun uuteen teknologiaan tai kehittäjän mukaantuloon projektiin, jossa on käytössä hänelle uutta teknologiaa. Tämä voidaan tulkita tutkimusjakson ajalta selkeäksi ohjelmistokehityksen nopeutta hidastavaksi tekijäksi, koska uuden teknologian oppiminen

vie aina aikaa. Aineistosta ei käy suoraan ilmi, liittyykö tähän kehittäjien siirtymistä projektista toiseen, mikä voisi aiheuttaa tämänkaltaisia asioita, vai onko kyseessä puhtaasti uusien projektien alkaminen ja uusien työntekijöiden aloittaminen. Aineistosta ei myöskään ilmene, onko kesken projektien otettu uusia teknologioita käyttöön.

Projektin johtamiseen liittyviä asioita on tuotu esille toiseksi eniten. Näissä on melkein jokaisessa tapauksessa ollut kyse siitä, että tiimi on ollut epätietoinen tulevista tehtävistä tai projektin jatkosta. Kyse on siis ollut pääosin läpinäkyvyyden puutteesta projekteissa, mikä on aiheuttanut epätietoisuutta tekijöiden keskuudessa.

Henkilöiden kuormitukseen liittyvät asiat ovat yleensä olleet tilanteita, joissa ollaan oltu riippuvaisia jostain tietystä henkilöstä, jolla on ollut liian suuri kuormitus, ja tästä on seurannut ylimääräistä odottamista. Esimerkiksi kirjoitettu koodi on jäänyt odottamaan katselmointia turhan pitkäksi ajaksi.

Muihin alakategorioihin sisältyy vain yksittäisiä asioita, joten ne jätetään tässä tutkielmassa käsittelemättä. Todennäköisesti suurimmat kehitysnopeutta rajoittavat tekijät eivät sisälly kyseisiin alakategorioihin.

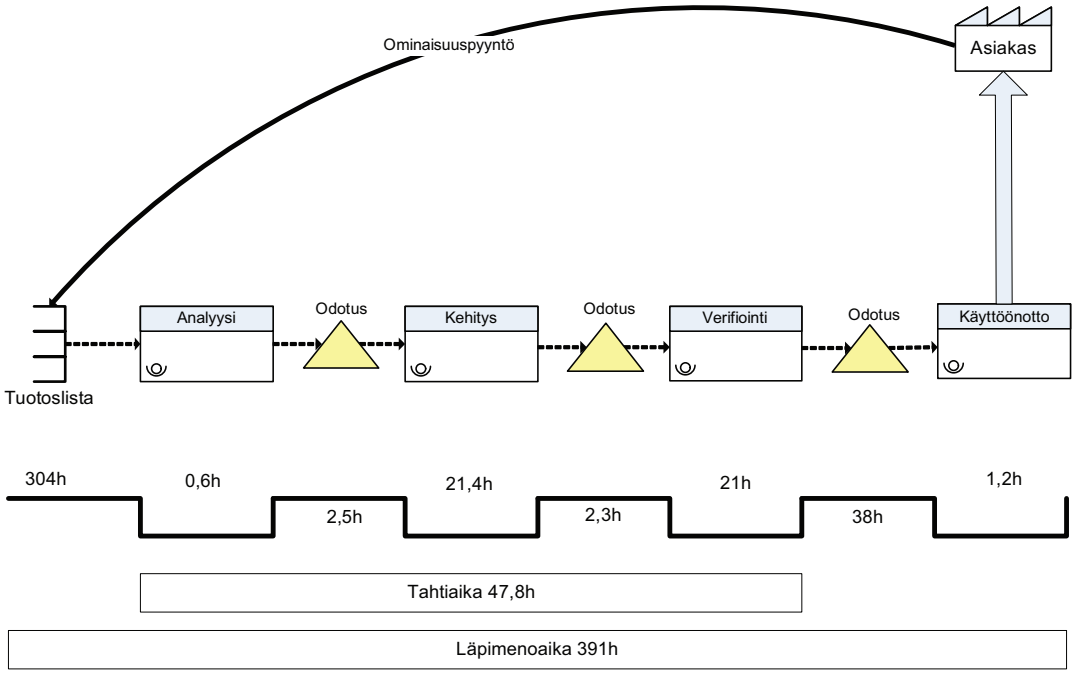
7.2 Tutkimustulokset vaatimusten mittausdatan perusteella

Vaatimusten mittausdataa oli yhden tiimin osalta olemassa jo noin puolen vuoden ajalta ennen varsinaisen tutkimuksen aloittamista. Tutkimuksen aikana kyseistä dataa kerättiin kolmen tiimin osalta. Vuoden 2010 tulokset toimivat vertailukohtana vuoden 2011 tuloksiin nähden, mikä auttaa tulkitsemaan vuoden 2011 tuloksia.

7.2.1 Loppuvuosi 2010

Kuvasta 7.3 käy ilmi vaatimusten mittausdatan tulokset arvovirtakartassa toukojoulukuulta 2010. Tämä mittausdata on kerätty yhden tiimin osalta, joka otti tapausyrityksessä ensimmäisenä käyttöön Lean-ohjelmistokehitysmallin ennen varsinaista tutkimuksen aloittamista, ja sitä käytetään lähinnä vertailukohtana varsinaiseen tutkimuksen aikana kerättyyn dataan.

Kuvan alareunassa olevat kellonajat kuvaavat työtuntien (ei absoluuttisten tuntien eikä miestyötuntien) määrää. Yksi työpäivä on kestoltaan 7,5 tuntia ja viikon seitsemästä päivästä viisi on työpäiviä. Eri työvaiheiden kestot on laskettu kaavalla $t = AVERAGE(A) * (7,5/24) * (5/7)$, missä A on kyseisen työvaiheen todellisten kestojen joukko. Kestot ovat tunteina, ja ne on laskettu jokaiselle vaatimukselle kaa-



Kuva 7.3: Arvovirtakartta loppuvuonna 2010

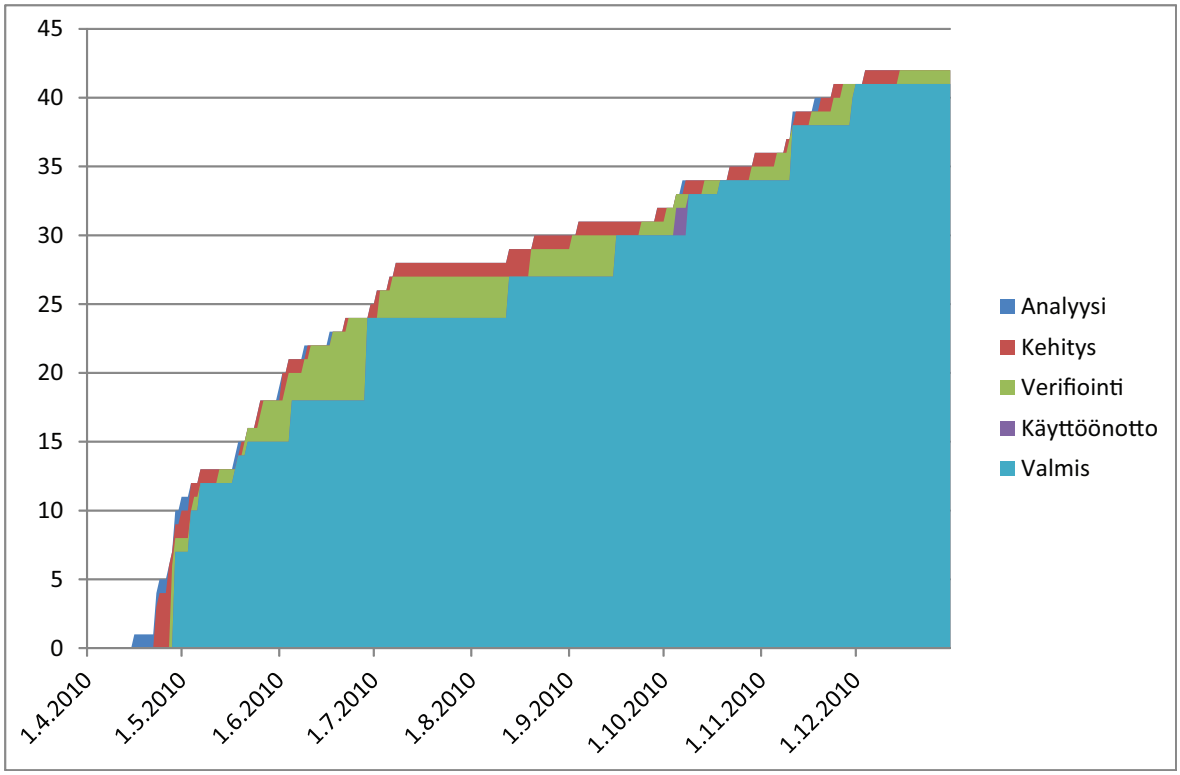
valla $t2 - t1$, missä $t2$ on kyseisen vaatimuksen työstön lopetusaika ja $t1$ on työstön aloitusaika. Ensimmäisestä kaavasta nähdään suoraan, että arvot eivät ole täysin tarkkoja, mutta ne ovat keskenään vertailukelpoisia.

Vuonna 2010 kerätystä datasta havaitaan, että odotusaika ennen käyttöönottoa (38h) on suurempi kuin muihin työvaiheisiin käytetty aika. Tästä voisi ajatella, että kyseisessä kohdassa prosessia on selkeä rajoite. Tälle odotusajalle on tapausyrityksessä looginen selitys: toteutettuja vaatimuksia käyttöönotettiin kyseisissä projekteissa tietyin aikavälein. Lean-ajattelun mukaan tässä on selkeää hukkaa. Koska tapausyrityksessä kyse on käytänteestä, jossa asiakas saa uudet ominaisuudet ennalta sovittuna päivämääränä, ei tätä voida suoraan pitää rajoitteena.

Käyttöönoton odotuksen jälkeen seuraavaksi suurimmat ajat ovat kehityksessä ja verifioinnissa. Ohjelmistotuotannossa on luonnollista, että kehitykseen käytetään paljon aikaa verrattuna muihin työvaiheisiin. Verifiointiin käytetty aika sen sijaan vaikuttaa huomattavan suurelta. Koska muihin työvaiheisiin käytetyt ajat ovat suhteellisen pieniä, voisi loppuvuoden 2010 arvovirtakarttaa tulkita siten, että suurin kehitysnopeutta hidastava tekijä on verifioinnissa. Sitä nopeuttamalla olisi mahdollisuus saada tahtiaikaa pienemmäksi.

Kuvan 7.4 kaaviossa on nähtävissä päivittäin jaoteltuna kussakin työvaiheessa olevien vaatimusten lukumäärä loppuvuonna 2010. Tulokset ovat peräisin samasta aineistosta kuin loppuvuoden 2010 arvovirtakarttakin (kuva 7.3). Vaatimuksen on

katsottu olevan työvaiheessa, kun se on ollut valmis edellisen työvaiheen osalta, eli kun se odottaa prosessointia tai on prosessoitavana kyseisessä työvaiheessa.



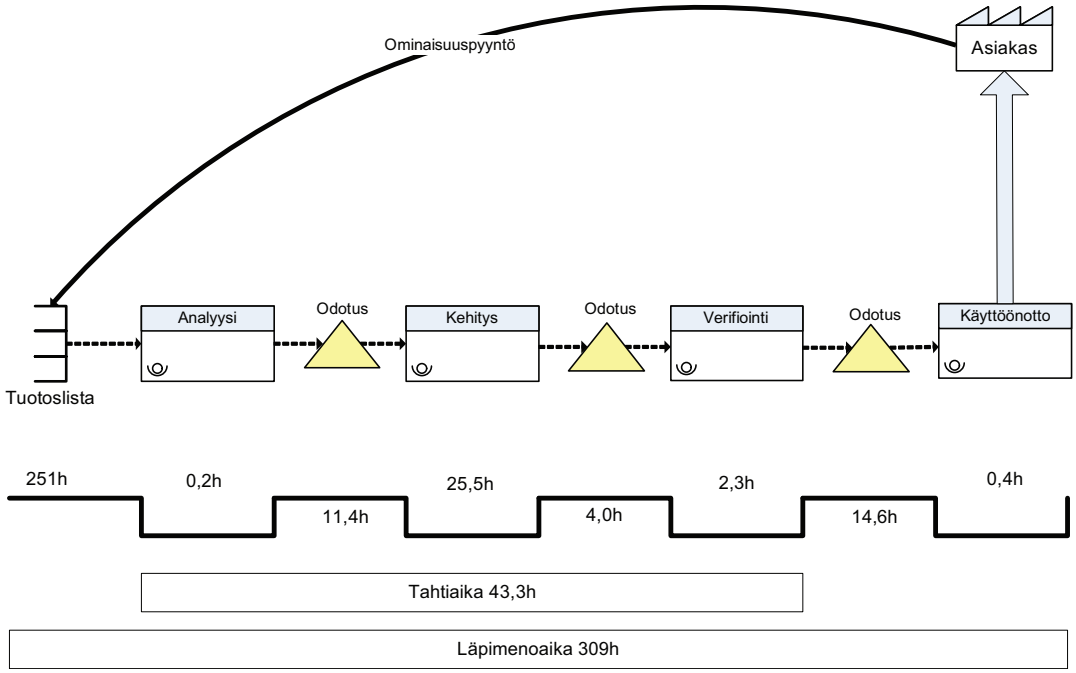
Kuva 7.4: Eri työvaihessa olevat vaatimukset loppuvuonna 2010

Kuvasta 7.4 nähdään, että mikään työvaihe ei ole toiminut missään vaiheessa hälyttävän suurena pullonkaulana. Kesä–syyskuussa on havaittavissa, että verifiointi-työvaiheeseen on kasautunut välillä vaatimuksia ja joskus on vienyt kauan ennen kuin kertyneet vaatimukset on saatu verifioitua. Tämä vahvistaa käsitystä, että verifiointi-työvaihe on ollut ”järjestelmän” nopeutta rajoittava tekijä, kuten kuvan 7.3 arvovirtakartan perusteella jo arvioitiin. Vaatimusten kasautumiselle verifiointi-työvaiheeseen voi olla useita syitä, mutta vuoden 2010 rajoittavat tekijät kuuluvat tämän tutkimuksen ulkopuolelle. Vuoden 2011 osalta on hyvä tutkia, löytyykö vastaavia tilanteita. Lean-ajattelussa tällaiset tilanteet tulkitaan puhtaaksi hukaksi.

7.2.2 Arvovirtakartta tammi-maaliskuu 2011

Vuoden 2011 ensimmäisen neljänneksen arvovirtakartasta (kuva 7.5) nähdään selkeitä eroja verrattaessa sitä vuoden 2010 arvovirtakarttaan (kuva 7.3). Arvot eivät ole suoraan vertailukelpoisia, koska vuoden 2011 aineisto on kerätty kolmelta eri tiimiltä ja erilaisista projekteista kuin vuoden 2010 aineisto, joka on kerätty vain yh-

deltä tiimiltä. Arvot ovat kuitenkin suuntaa antavia.



Kuva 7.5: Arvovirtakartta tammi-maaliskuussa 2011

Asiakkaan kannalta merkittävin tunnusluku, eli läpimenoaika, on ollut vuoden 2011 ensimmäisellä neljänneksellä 82 tuntia pienempi kuin loppuvuonna 2010. Tämä tarkoittaa käytännössä, että asiakas saa haluamansa ominaisuuden noin 11 työpäivää nopeammin. Tahtiajassa ei ole huomattavan suurta eroa, mutta eri työvaiheiden ajat ovat muuttuneet selkeästi.

Alkuvuonna 2011 vaatimukset ovat odottaneet tuotoslistassa 53 tuntia vähemmän kuin loppuvuonna 2010. Tässä on selkeää parannusta. Todennäköisesti tuotoslistojen kokoa on onnistuttu pienentämään, mistä seuraa automaattisesti lyhyemmät odotusajat. Tässä on jäänyt vielä kuitenkin selkeää parantamisen varaa.

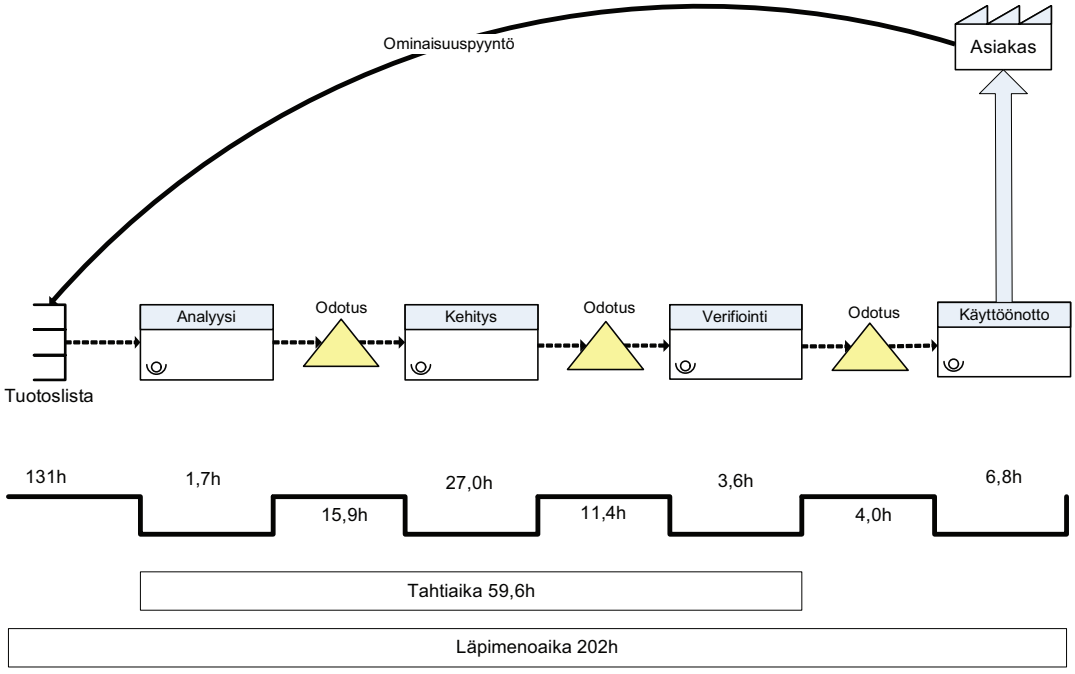
Analyysiin käytetyn ajan merkitys on kokonaisuuden kannalta vähäinen. Mielenkiintoisesti analyysin jälkeisen odotuksen aika on noussut huomattavasti 2,5 tunnista 11,4 tuntiin. Tämä on mitä todennäköisimmin seurausta siitä, että alkuvuoden 2011 aikana tapausyrityksessä luovuttiin niin sanotusta puhtaasta kanban-kehitysprosessista ja otettiin kaikille tiimeille iteraatiot käyttöön. Tuloksena on eräänlainen ”Scrumban”-prosessi, jossa kaikki iteraatioon sisällytetyt vaatimukset analysoidaan iteraation alussa, mutta vain osa otetaan heti työn alle. Tästä seuraa ylimääräistä odotusta analyysin ja kehityksen välille. Lean-ajattelun näkökulmasta tämä on hukkaa. Itse kehitys-työvaiheessa ei ole merkittäviä muutoksia. Kehitystyövaiheen jälkeinen odotus on prosentuaalisesti kasvanut merkittävästi (74%). Tämä voidaan

tulkita puhtaaksi hukaksi, mutta kokonaisuuden kannalta kyseisen neljän tunnin odotuksen merkitys on vähäinen. Vuoden 2010 arvovirtakartasta todettiin, että verifiointi oli tuolloin rajoittavana tekijänä. Tämä ei selkeästi ole ollut tilanne enää alkuvuonna 2011. Myös verifioinnin jälkeinen odotus on pienentynyt merkittävästi 21 tunnista 2,3 tuntiin. Tässä on todennäköisesti kyse erilaisista projekteista, koska ero on huomattavan suuri. Toiset projektit ovat helpommin verifioitavia kuin toiset.

Yhteenvetona alkuvuonna 2011 odotusajat ovat kokonaisuudessaan pienentyneet, mutta analyysin ja kehityksen jälkeiset odotusajat ovat kasvaneet, minkä takia tahtiaikaan ei ole tullut huomattavaa parannusta. Verifiointiin käytetty aika ja sen jälkeinen odotusaika ovat pudonneet merkittävästi. Rajoittaviksi tekijöiksi voisi alkuvuoden 2011 osalta tulkita kehityksen ja sen molemmin puolin olevat odotusajat.

7.2.3 Arvovirtakartta huhti-kesäkuu 2011

Vuoden 2011 toisen neljänneksen arvovirtakartassa (kuva 7.6) on edelleen tapahtunut selkeitä muutoksia ensimmäiseen neljännekseen verrattuna (kuva 7.5). Kyseiset arvovirtakartat ovat jokseenkin vertailukelpoisia ohjelmistokehittäjien pysyessä pääosin samoina. Myös osa projekteista oli samoja, mutta joitain projekteja loppui ja joitain uusia alkoi toisella neljänneksellä, mikä hieman hankaloittaa vertailua.



Kuva 7.6: Arvovirtakartta huhti-kesäkuussa 2011

Vuoden 2011 toisen neljänneksen arvovirtakartasta (kuva 7.6) havaitaan, että lä-

pimenoaika on pienentynyt lähes 35% ensimmäiseen neljännekseen verrattuna. Kuvasta myös nähdään, että tämä on seurausta siitä, että vaatimukset viettävät huomattavasti vähemmän aikaa tuotoslistassa ennen kuin ne otetaan tekoon, mikä tarkoittanee, että tuotoslistojen kokoa on edelleen onnistuttu pienentämään. Trendi on siis ollut sama, joka havaittiin 2010 loppuvuoden ja 2011 ensimmäisen neljänneksen välillä. Kaiken kaikkiaan läpimenoaika on pienentynyt loppuvuoden 2010 alkuperäisestä 391 tunnista 202 tuntiin, eli parannusta on ollut 189 tuntia. Tämä tarkoittaa 25 työpäivää, eli asiakas saa pyytämänsä ominaisuuden yli kuukauden nopeammin. Parannusta alkuperäiseen on ollut 48%.

Toisella neljänneksellä toteutetut vaatimukset ovat viettäneet tuotoslistassa keskimäärin 131 tuntia. Asiakas joutuu siis odottamaan noin 17 työpäivää, eli vähän yli kolme viikkoa, ennen kuin hänen pyytämänsä ominaisuus otetaan työn alle. Leanin mukaan tämä on edelleen hukkaa, mitä tulisi pyrkiä poistamaan.

Analyysiin käytetyn työn määrä on hieman kasvanut, mutta vaikutus kokonaisuuden kannalta on pieni. Lisäksi tämä on Lean-ajattelun mukaan lisäarvoa tuottavaa työtä, joten siihen ei tarvitse suhtautua yhtä kriittisesti kuin puhtaaksi hukaksi luettaviin työvaiheisiin.

Analyysin jälkeinen odotusaika on entisestään kasvanut 11,4 tunnista 15,9 tuntiin. Syyt tähän ovat todennäköisesti samat kuin edellä, eli työtä tehdään iteraatioissa, jolloin kaikki vaatimukset analysoidaan iteraation alussa, ja tämän jälkeen ne jäävät odottamaan kehitystä.

Kehitykseen käytetty aika on pysynyt käytännössä lähes muuttumattomana. Huomionarvoista on, että kehitykseen kuluva aika on tahtiajasta vain noin 45%, eli alle puolet. Ohjelmistokehityksen nopeus ei siis tapausyrityksessä riipu pelkästä ohjelmointinopeudesta.

Verifiointia edeltävä odotusaika on lähes kolminkertainen ensimmäiseen neljännekseen verrattuna. Tutkimusdataa tarkemmin tutkimalla oli havaittavissa, että mukana on kaksi vaatimusta, joissa kyseinen aika on ollut erittäin suuri, mikä selittää lukua. Kyseisen projektin projektipäällikön haastattelun perusteella odotusajat johtuivat riippuvuuksista tiettyyn sidosryhmään, minkä vuoksi verifiointia ei pystytty tekemään. Kyseessä oli siis odotus, mikä tulkitaan Leanissa hukaksi. Mikäli nämä kaksi poikkeamaa jätettäisiin huomiotta, olisi keskimääräinen odotusaika 0,8 tuntia. Tämä voisi kuitenkin olla liian optimistinen näkemys asiasta, koska vastaavia odotuksia voi esiintyä jatkossakin, ellei niitä tietoisesti pyritä välttämään.

Verifiointiin käytetty aika on hieman kasvanut ensimmäiseen neljännekseen nähden, mutta tahtiajan ja läpimenoajan osalta tälläkään ei ole suurta merkitystä.

Verifioinnin jälkeinen odotusaika taas on pudonnut merkittävästi. Tämä selit-

tyy projektien luonteella: työn alla olleista projekteista suuri osa on ollut sellaisia, missä uusien ominaisuuksien käyttöönotto on voitu tehdä välittömästi. Mikäli ominaisuudet otetaan käyttöön vasta iteraation lopussa, kasvaa luonnollisesti kyseinen odotusaika.

Käyttöönottoon käytetty aika on kasvanut merkittävästi. Sekä arvovirtakartan että tutkimusaineiston perusteella on pääteltävissä, että käyttöönotoissa on ollut haasteita. Käyttöönotto on Lean-ajattelun mukaan asiakkaalle lisäarvoa tuottamatonta hukkaa, koska siinä ei enää synny lisäarvoa. Tuotteeseen ei synny enää mitään uutta käyttöönoton aikana.

Tahtiaika on kasvanut toisella neljänneksellä 38%, mikä on huomattavan suuri määrä. Tämä on seurausta siitä, että kaikki yksittäiset ajat, joista tahtiaika koostuu, ovat kasvaneet.

Yhteenvetona toisella neljänneksellä läpimenoaikaa on edelleen pystytty pienentämään merkittäväksi, mutta tahtiaika on kasvanut ensimmäiseen neljännekseen nähden. Ohjelmistokehitysnopeutta rajoittaviksi tekijöiksi voisi yhä tulkita kehityksen ja sen molemmin puolin olevat odotusvaiheet. Lisähuomiona mainittakoon käyttöönottoon käytetyn ajan kasvun lisäävän tarpeettomasti prosessissa olevaa hukkaa.

7.2.4 Arvovirtakartta tammi-kesäkuu 2011

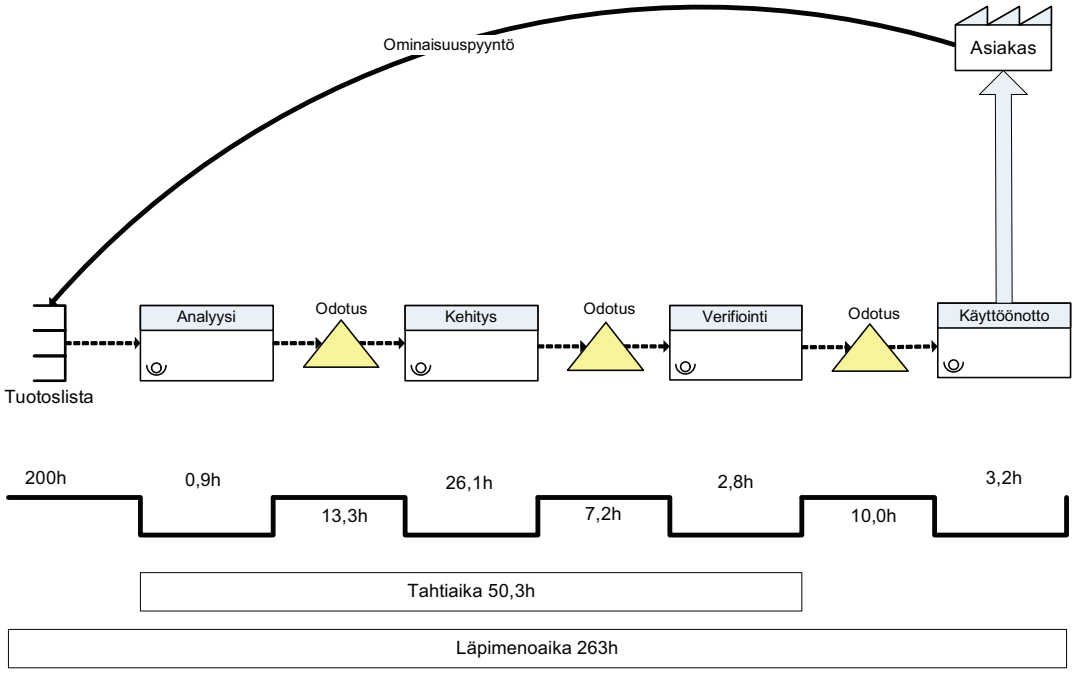
Kuvassa 7.7 on arvovirtakartta koko tutkimuksen ajalta, eli vuoden 2011 ensimmäiseltä puoliskolta. Siinä on siis yhdistetty aiemmissa luvuissa (7.2.2 ja 7.2.3) käsitellyt aineistot.

Puolen vuoden aikajaksolta piirretty arvovirtakartta antaa tapausyrityksen tilanteesta ehkä kaikista luotettavimman kuvan, koska aineistoa on yhteensäkin varsin rajallinen määrä. Tällöin yksittäiset poikkeamat aineistossa eivät vaikuta niin paljoa kuin aiempien lukujen arvovirtakartoissa, jotka ovat kolmen kuukauden aikajaksoilta.

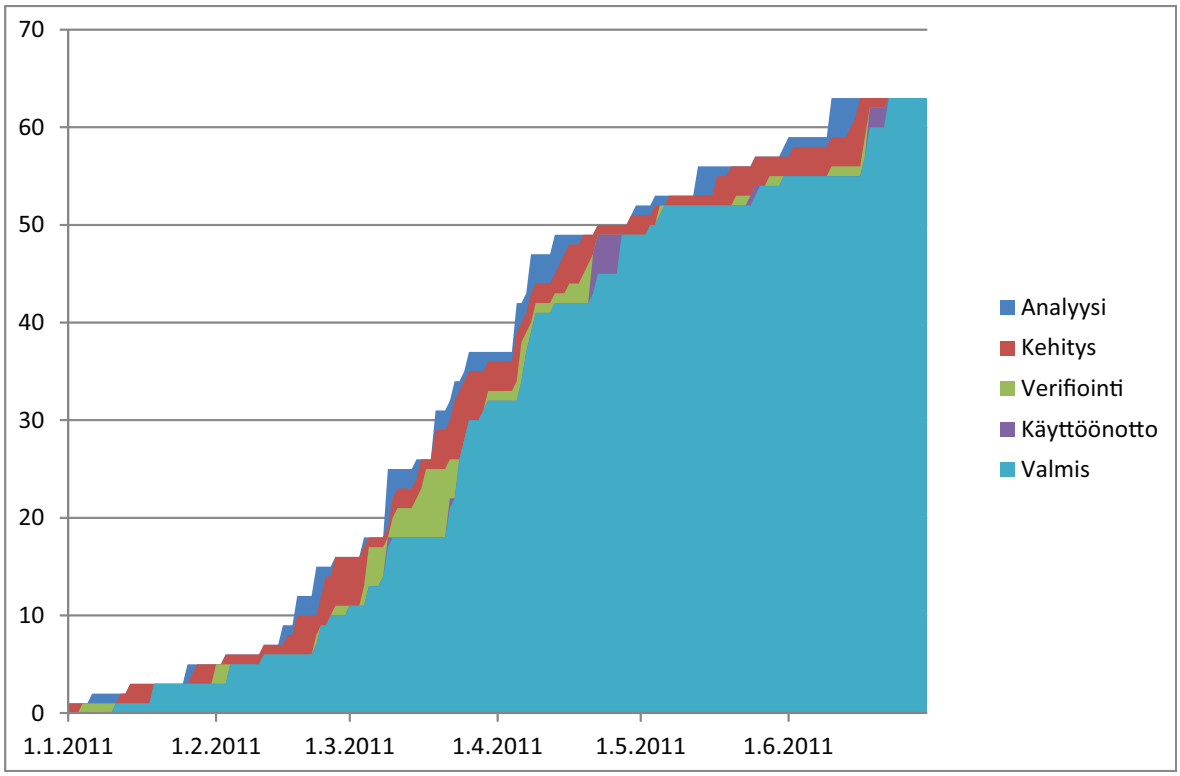
Puolen vuoden luvut ovat luonnollisesti ensimmäisen neljänneksen ja toisen neljänneksen lukujen välimaastossa. Kehitysnopeutta rajoittavat tekijät ovat prosessinäkökulmasta samat: kehitysvaihe ja sen ympärillä olevat odotusvaiheet.

7.2.5 Vaatimukset työvaiheittain jaoteltuina 2011

Kuvassa 7.8 nähdään eri työvaiheissa olevat vaatimukset tutkimuksen ajalta päivittäin jaoteltuina vastaavasti kuin kuvassa 7.4 nähtiin vuoden 2010 loppuvuoden osalta.



Kuva 7.7: Arvovirtakartta tammi-kesäkuussa 2011



Kuva 7.8: Eri työvaihessa olevat vaatimukset vuonna 2011

Vuoden 2011 ensimmäisestä puoliskostakaan ei ole havaittavissa mitään hälyttävää. Loppuvuoden 2010 vuokaaviosta havaitsimme, että siellä vaatimuksia oli kasautunut verifiointi-työvaiheeseen. Alkuvuonna 2011 näin ei ole tapahtunut maaliskuuta lukuun ottamatta. Maaliskuun lisäksi vuonna 2011 ei ole havaittavissa muita isompia ongelmakohtia. Mainittakoon, että aiemmin käsitellyissä arvovirtakartoissa iso osa maaliskuussa verifiointiin kasautuneista vaatimuksista ilmenee vasta huhti-kesäkuun arvovirtakartassa, koska vaatimukset on sisällytetty arvovirtakarttoihin käyttöönottopäivämäärien perusteella. Maaliskuun loppupuolella verifioiduista vaatimuksista iso osa on otettu käyttöön vasta huhtikuun alussa.

Mielenkiintoisena yksityiskohtana alkuvuoden 2011 kaaviosta havaitaan, kuinka käyttöönotetut iteraatiot aiheuttavat pientä vaatimusten kasautumista analyysityövaiheeseen. Vuoden 2010 kaaviossa tällaista ilmiötä ei ole näkyvissä. Käytössä oli tuolloin puhdas kanban-prosessi, jossa analyysi suoritettiin vaatimus kerrallaan, ja vaatimus siirtyi analyysistä suoraan kehitykseen.

Kaaviosta nähdään myös, että osassa työvaiheita ei ole välillä lainkaan vaatimuksia työn alla. Lean-ideologian mukaan optimaalisinta olisi luoda yksiosainen virtaus, jossa jokaisessa työvaiheessa on yksi kappale kerrallaan ja työvaiheiden välissä ei ole lainkaan välivarastoja, jolloin ei ole odottamistakaan.

TOC-teorian mukaan jokaisessa järjestelmässä on rajoite. Kuvaa 7.8 tulkitsemalla se on tässä ”järjestelmässä”, eli tapausyrityksen ohjelmistotuotannossa, ollut alkuvuonna 2011 kehitystyövaihe, koska kyseisessä työvaiheessa on lähes koko ajan ollut eniten vaatimuksia työn alla.

7.3 Yhteenveto

Tutkimustulokset käytiin läpi ja kunkin aineiston tulokset analysoitiin erikseen. Aineistoina olivat jälkipuintipalaverien pöytäkirjat tutkimuksen ajalta sekä vaatimuksista kerätty mittausdata. Mittausdataa oli käytössä vuoden 2011 ensimmäisen puoliskon lisäksi myös vuoden 2010 jälkimmäiseltä puoliskolta yhden tiimin osalta.

Jälkipuintipalaverien pöytäkirjojen ”parannettavaa”-aiheet jaoteltiin ihmiset, prosessi, teknologia ja tuote -kategorioihin. Aineiston perusteella prosessista on keskusteltu, ja siihen on tuotu parannusehdotuksia selkeästi eniten. Toiseksi eniten on noussut esille ihmiset-kategoriaan liittyvät asiat. Ihmiset-kategoria on näistä kahdesta todennäköisesti kehitysnopeuden kannalta kriittisempi.

Prosessiin liittyen eniten on tullut täysin uusia ehdotuksia ja myös verifiointia on käsitelty paljon. Lisäksi prosessin noudattamisessa on ollut puutteita, tarvittavaa tapausyrityksen sisäistä tietoa ei ole ollut aina saatavilla ja on esiintynyt riippuvuuk-

sia asiakkaasta. Seurauksena ongelmista on usein ollut tarpeeton odottaminen.

Ihmisiä käsitelleistä aiheista kävi ilmi, että suurimpana haasteena uusien teknologioiden opettelu on vienyt aikaa. Lisäksi projektit eivät ole olleet täysin läpinäkyviä, mikä on aiheuttanut epävarmuutta tiimeissä. Esille nousi myös, että tiettyjen henkilöiden ylikuormitus on aiheuttanut odottamista.

Arvovirtakartoista nähdään, että loppuvuoden 2010 ja alkuvuoden 2011 välillä läpimenoaika on pudonnut 391 tunnista 263 tuntiin eli noin 33 %. Tahtiaika taas on hieman kasvanut: 47,8 tunnista 50,3 tuntiin eli noin 5 %. Prosessin kannalta ”järjestelmän” nopeutta rajoittava työvaihe on kehitys sekä arvovirtakarttojen että kumulatiivisen vuokaavion perusteella. Tässä on tapahtunut muutosta loppuvuoteen 2010 nähden, jolloin rajoittavana tekijänä oli verifiointi. Arvovirtakarttojen perustella nähdään, että alkuvuonna 2011 osa ajasta on kulunut kehitys-työvaiheen molemmin puolin tapahtuvaan odotukseen, mitä voidaan kehityksen lisäksi pitää nopeutta rajoittavana tekijänä.

8 Pohdinta

Aiemmin käsitellyistä tutkimustuloksista saadaan johtopäätöksiä pohtimalla, mitä tulokset yhdessä tarkoittavat. Johtopäätösten perusteella laaditaan tapausyritykselle suosituksia sekä käydään läpi mahdollisia jatkotutkimustarpeita.

Tutkimuskysymys oli: *"Mitkä ovat merkittävimmät ohjelmistokehityksen nopeutta rajoittavat tekijät tutkittavassa yrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen?"*. Apukysymyksiä olivat *"Mitä Lean-ohjelmistokehitys on?"* ja *"Mitä Lean-ohjelmistokehitys tarkoittaa tutkittavassa yrityksessä?"*.

8.1 Johtopäätökset

Tahtiaika määrää järjestelmän nopeuden, eli sen, kuinka usein järjestelmästä saadaan keskimäärin uusi tuotos ulos. Tutkimustuloksissa vuoden 2011 ensimmäisen puoliskon tahtiaika on 50,3 tuntia, mikä tarkoittaa, että järjestelmästä (tapausyrityksen ohjelmistotuotanto) valmistuu uusi vaatimus noin seitsemän työpäivän välein.

Läpimenoaika on asiakkaalle siinä mielessä näkyvämpi kuin tahtiaika, että asiakkaan odotusaika pienenee läpimenoajan pienentyessä. Toisin sanoen pieni tahtiaika ei auta, mikäli läpimenoaika on kohtuuttoman suuri. Tapausyrityksessä läpimenoaika on 263 tuntia eli tahtiaika on 19 % läpimenoajasta. Tämä tarkoittaa, että tahtiajan lisäksi täytyy pienentää myös läpimenoaikaa. Lean-ajattelun mukaan tapausyrityksen arvovirran työvaiheista vain analyysi ja kehitys ovat lisäarvoa tuottavaa työtä. Näitä tehdään keskimäärin 27 tuntia per vaatimus, mikä on tahtiajasta noin 54 % ja läpimenoajasta noin 10 %. Voidaan siis sanoa, että järjestelmän tehokkuus on 10 %.

Läpimenoajan suhteen tuotoslista vaikuttaisi selkeästi toimivan rajoittavana tekijänä, mikä nähdään suoraan aiemman luvun arvovirtakartoista. Vaatimus odottaa tuotoslistassa keskimäärin 200 tuntia, mikä on 76 % läpimenoajasta.

Tahtiajan osalta rajoittava tekijä, kuten aiemmin todettiin, vaikuttaisi olevan kehitystyövaihe sekä sitä edeltävä ja sen jälkeinen odottelu. Tapausyrityksen jälkipuintipalavereissa on keskusteltu eniten prosesseista, mutta tahtiaika on pysynyt käytännössä samana tai jopa pudonnut vuoden toisella neljänneksellä. Toisaalta prosessikeskusteluissa pääpaino on ollut verifioinnissa, ja verifiointi ei ole enää pullonkaulana kuten se oli loppuvuonna 2010, joten tähän jälkipuintipalavereilla on voinut olla merkitystä. Tieto kehitysvaiheesta ja sitä ympäröivistä odotusvaiheista

rajoittavina tekijöinä ei yksinään anna paljoakaan informaatiota. Tuntuu luonnolliselta, että ohjelmistokehityksessä ohjelmointi rajoittaa ”järjestelmän” nopeutta, joten tähän aiheeseen on syytä pureutua hieman tarkemmin.

Ohjelmoinnin tekevät ihmiset, ja ihmisillä on suurin vaikutus ohjelmistokehityksen nopeuteen [McC02, s.12–13]. Tätä kategoriaa oli käsitelty paljon jälkipuintipalavereissa, ja eniten parannettavaa oli löytynyt seuraavista aiheista:

- osaaminen / koulutus (uudet teknologiat)
- projektin johtaminen (projektien riittämätön läpinäkyvyys)
- henkilöiden kuormitus

Voisi ajatella, että todennäköisesti juuri nämä asiat rajoittavat myös kehitystyövaiheen nopeutta. Osaamiseen liittyvät asiat liittyivät yleensä uuteen teknologiaan, mikä heijastuu suoraan kehitystyövaiheeseen. Myös tutkijan omat havainnot tukevat tätä päätelmää. Projektin johtaminen vaikuttaa kaikkiin työvaiheisiin, myös kehitykseen. Johtamisen osalta kyse oli turhan vähäisestä läpinäkyvyydestä projekteissa: kehitystiimi ei tiedä, kuinka projekti tulee etenemään, mikä voi myöhemmin näkyä esimerkiksi väärinä valintoina. Henkilöiden kuormituksesta taas oli seurannut odottelua toisaalla, josta ainakin osa on liittynyt suoraan kehitystyövaiheeseen.

Itse prosessista oli jälkipuintipalavereissa löydetty eniten parannettavaa seuraavista:

- verifiointi
- puutteita prosessin noudattamisessa
- tiedonpuute
- asiakasriippuvuus

Luvussa 7.2 todettiin, että verifiointi ei enää alkuvuoden 2011 osalta ollut pullonkaulana kuten se oli ollut loppuvuonna 2010. Todennäköisesti jälkipuintipalavereilla on ollut asiaan oma merkityksensä. Näin ollen verifiointi voidaan jättää ulkopuolelle kartoitettaessa kehitysnopeutta rajoittavia tekijöitä. Muut yllä listatut osa-alueet toimivat lähes varmasti kehitystyövaiheen nopeutta rajoittavina tekijöinä. Prosessin noudattamattomuus tarkoittaa yleensä joiltain osin liian vähäistä kuria tai liian vähäistä itsekuria, mutta se voi olla seurausta myös muista asioista. Leanissa myös työympäristön visualisoinnilla ja visuaalisella ohjauksella on iso merkitys prosessien noudattamisessa. Päällikön tulisi yhdellä silmäyksellä voida havaita,

noudatetaanko standardeja ja toimintaohjeita. Prosessin noudattamattomuuden syiden selvittäminen rajautuu kuitenkin tämän tutkimuksen ulkopuolelle, eikä se olisi mahdollista tutkimusaineiston perusteella. Tiedonpuute vaikuttaisi johtuvan muutamaa poikkeusta lukuun ottamatta puutteellisesta dokumentoinnista, mikä käy ilmi tutkimusaineistosta. Asiakasriippuvuudet taas ovat olleet luonteeltaan sellaisia, mitkä todennäköisesti pystyttäisiin välttämään aikaisemmalla tai suuremmalla asiakaskommunikaatiolla. Tämä päätelmä pohjautuu jälkipuintipalaverien pöytäkirjojen lisäksi tutkijan omiin havaintoihin tutkimuksen aikana.

Tutkimustulosten ja pohdiskelun johtopäätöksenä tapausyrityksestä on löydettävissä taulukossa 8.1 näkyvät ohjelmistotuotannon nopeutta todennäköisesti rajoittavat tekijät. Tekijät eivät ole tärkeysjärjestyksessä, eikä niiden todellista vaikuttavuutta kokonaisuuteen ole arvioitu. Tekijät on kuvattu taulukon oikeanpuoleisessa sarakkeessa.

Taulukossa ensimmäisenä on tuotoslistan koko. Tämä on käytännössä lähes sama asia kuin ohjelmiston vaatimusten lukumäärä. McConnelin mukaan ominaisuusluetteloon pätee 80/20 sääntö [McC02]. Tämä tarkoittaa, että jos tuotoslista on joustava, voidaan tuotteesta toteuttaa se 80 prosenttia, joka vie 20 prosenttia ajasta. Loput 20 prosenttia voidaan poistaa listalta ja lisätä ne siihen myöhemmin uudelleen, mikäli ne katsotaan yhä tarpeelliseksi. Näin toimimalla vältetään vaatimusten liioittelua *(engl. feature creep)*, joka on yksi klassisista ohjelmistoprojektien virheistä [McC02] [CSB08]. Liian suuri vaatimusten määrä on riski muissakin kuin Lean-ohjelmistoprojekteissa. Ylimääräiset ominaisuudet voidaan lukea puhtaaksi hukaksi [PP03, s.6].

Seuraava nopeutta rajoittava tekijä ovat uudet teknologiat. Myös uusiin teknologioihin liittyy klassisia virheitä: hopealuotisyndrooma sekä uusien välineiden tai menetelmien antamien säästöjen yliarviointi [McC02] [CSB08]. Hopealuotisyndroomalla tarkoitetaan, että uuden teknologian odotetaan olevan ratkaisu aikatauluongelmiin. Uusilla menetelmillä taas saavutetaan yleensä korkeintaan 10 prosentin tuottavuuden lisäys [McC02]. Sen lisäksi, että uudet teknologiat vaikuttaisivat olevan usein esiintyvä haaste ohjelmistokehityksessä, ovat ne myös Leanissa yleisesti tunnistettu asia. Eräs Likerin määrittelemistä Leanin periaatteista on ”käytä vain luotettavaa, läpikotaisin testattua teknologiaa” [Lik10, s.6].

Eräs Scrumin lähtökohdista on, että se on suunniteltu lisäämään ohjelmistoprojektien läpinäkyvyyttä [SVBP07]. Tapausyrityksen käytänteet perustuvat hyvin pitkälle Scrumiin. Niinpä voidaan pitää pienenä yllätyksenä, että läpinäkyvyyden puute on eräs kehitysnopeutta rajoittavista tekijöistä. Läpinäkyvyyden puute ei esiinny McConnellin klassisten virheiden listalla [McC02], joten se ei lukeutune ohjelmis-

Taulukko 8.1: Ohjelmistokehityksen nopeutta rajoittavia tekijöitä tapausyrityksessä

Mahdollinen rajoittava tekijä tapausyrityksessä	**Kuvaus**
Tuotoslistan koko	Leanin kannalta tuotoslista on varasto, joka kasvattaa tarpeettomasti läpimenoaikaa. Luonnollisesti vaatimuksia on hyvä olla listassa riittävästi, jotta pystytään suunnittelemaan asioita tulevaisuuteen, mutta silti listan koko tulisi pitää mahdollisimman pienenä.
Uudet teknologiat	Uusien teknologioiden oppiminen ja omaksuminen vie aikaa henkilöiltä, mikä näkyy hitaampana ohjelmistokehityksenä. Uusien teknologioiden tuominen ja oikeat teknologiavalinnat ovat tärkeitä onnistumisen kannalta, mutta on myös säilytettävä tasapaino vanhan ja uuden välillä.
Projektien riittämätön läpinäkyvyys	Epävarmuus projektien etenemisestä voi heijastua epävarmuutena tai väärinä päätöksinä myös tekemiseen.
Tiettyjen henkilöiden ylikuormitus	Ylikuormitus aiheuttaa odottamista toisaalla, jolloin seurauksena koko järjestelmän tuottavuus voi jäädä pienemmäksi kuin kuormittamalla henkilöitä vähemmän. Ylikuormitus on Lean-ajattelussa hukkaa.
Puutteita prosessien noudattamisessa	Prosessien noudattamatta jättäminen aiheuttaa sekavuutta tekemiseen ja näkyy siten hitaampana tekemisenä. Noudattamattomuus voi johtua esimerkiksi kurin tai itsekurin puutteesta, prosessien dokumentoinnin puutteesta, tai että prosesseista ei tiedetä, eli niistä ei ole annettu koulutusta.
Dokumentointi	Yrityksen sisäinen tiedonpuute tarkoittanee, että asioita ei ole dokumentoitu kaikilta tarpeellisilta osin tai dokumentaatio ei ole saatavilla. Toisaalta ketteriä menetelmiä käytettäessä pyritään välttämään ylimääräisen dokumentaation tuottamista, joten oikea tasapaino on tärkeää dokumentoinnin osalta.
Asiakas-kommunikaatio	Asiakasriippuvuudet johtuvat todennäköisesti usein liian vähäisestä tai liian myöhäisestä kommunikaatiosta asiakkaan kanssa.

toalan yleisimpien haasteiden joukkoon.

Tiettyjen henkilöiden ylikuormitus muodostaa helposti projektille pullonkaulan. Systematicilla havaittiin, että projektien alussa projektipäälliköillä oli suuri työkuorma, mistä seurasi viiveitä projektien aloituksessa [JP11]. Tämä johtui muun muassa siitä, että yleensä projektipäälliköt samaan aikaan viimeistelivät vielä edellistä projektia, kun aloittivat jo uutta. Systematicilla myös havaittiin, että mikäli tuotteen omistajalle ei resursoida riittävästi aikaa tehdä kyseiseen rooliin liittyviä tehtäviä, kärsii sprinttien suorituskyky huomattavasti [JP11]. Luvussa 5.2 havaittiin, että heijunka, eli tuotannon ja aikataulujen tasapainottaminen, ei ole käytössä tapausyrityksessä. Tämä voi omalta osaltaan näkyä työkuormien epätasaisena jakautumisena eri henkilöiden kesken. Heijungan käyttöönotto vaikuttaisi tasapainottavan työmääriä eri henkilöiden välillä [KTAD07].

Puutteita prosessien noudattamisessa on yksi havaituista kehitysnopeutta rajoittavista tekijöistä. Nopea kehitys on mahdotonta ilman, että tekee laadukasta työtä, ja laadukkaan työn tekemiseksi tarvitaan paljon kuria [PP03, s.190]. Poppendieckien mukaan he ovat havainneet kurin puutteen olevan yleinen ongelma pienissä ja nopeasti kasvavissa yrityksissä, joissa ei ole keskitytty laatuun. Prosessien noudattamatta jättäminen voi olla merkki kurin puutteesta. Klassisten virheiden listalta löytyy esimerkiksi ”laadunvalvonnasta tinkiminen” [McC02] [CSB08], joka voidaan mieltää tähän kategoriaan kuuluvaksi. Mikäli projektin laadunvalvonnasta tingitään aikataulupaineen alla, maksaa tämä todennäköisesti moninkertaisen työmäärän myöhemmin [McC02, s.45].

Tapausyrityksessä dokumentoinnin osalta rajoittavana tekijänä on ollut nimenomaan sen puute. Joissain tapauksissa on ollut päinvastoin. Esimerkiksi Capital One Financial Services -nimisessä suuryrityksessä löydettiin Lean-käyttöönotossa redundanttia dokumentointia prosesseissa [PK06]. Kanban-prosessimallissa dokumentaatiota ei taas lähtökohtaisesti synny lainkaan ilman asiakastarvetta [IPF+11]. On kuitenkin hyvien käytänteiden mukaista dokumentoida vähintään järjestelmän tärkeimmät suunnitteluratkaisut sekä arkkitehtuuri [IPF+11]. Helsingin Yliopistossa tehdyssä tutkimuksessa havaittiin, että Kanbania käyttänyt tiimi kirjoitti sisäisesti tarvitsemansa dokumentaation, mutta vältti ylimääräisen dokumentaation kirjoittamiseen liittyvän hukan syntymisen [IPF+11]. Saattaa olla, että tietty määrä sisäistä dokumentaatiota tarvitaan tiedon sujuvaan leviämiseen.

Eräs tunnistetuista rajoittavista tekijöistä on asiakaskommunikaatio. Kehitystiimin jäsenten vähäinen asiakaskommunikaatio on löydetty ongelmakohdaksi myös Capital One:lla vuonna 2006 [PK06]. Systematicilla taas havaittiin vuonna 2008, että heillä oli useissa projekteissa parantamisen varaa asiakaskommunikaatiossa liittyen

vaatimusten tarkennuksiin [JP11]. Systematicilla perimmäiset syyt liittyivät asiakkaan sitouttamiseen tiiviiseen yhteistyöhön ja tuotteen omistajan roolin selkeään ymmärtämiseen kaikissa projekteissa. Havainnot antavat viitteitä, että asiakaskommunikaatio saattaa olla yksi yleinen kehitysnopeutta rajoittava tekijä.

Luvussa 6.2 käytiin läpi Murauskaiteen ja Adomauskasin heidän Ericssonnille tekemässä tutkimuksessaan kehittämäänsä mallia, jossa oli tunnistettu Leanissa mahdollisesti esiintyviä rajoitteita tai pullonkauloja. Tätä tutkimusta ei tehty suoraan kyseisen mallin perusteella, koska ei haluttu rajata pois mahdollisesti mallin ulkopuolelle jääviä nopeutta rajaavia tekijöitä. Taulukossa 8.2 on vertailtu tässä tutkimuksessa löytyneitä rajoitteita Murauskaiteen ja Adomauskasin malliin. Mallista on poimittu taulukkoon Lean-perusperiaate sekä Lean-pullonkaula, joka vastaa kyseistä löytynyttä mahdollista rajoitetta. Taulukosta havaitaan, että Murauskaiteen ja Adomauskasin mallista löytyy varsin kattavasti vastineet kaikille löytyneille rajoitteille. Dokumentoinnille ei ole suoraa vastinetta, mutta näkisin sen kuuluvan ”Luo tietoa” Lean-perusperiaatteen alle. Todennäköisesti lähes samat tutkimustulokset olisi siis saavutettu kyseistä malliakin käyttämällä, koska mikään löytynyt rajoite ei jää täysin mallin ulkopuolelle.

Taulukko 8.2: Rajoitteiden vertailu Murauskaiteen ja Adomauskasin [MA08] malliin

Mahdollinen rajoite tapausyrityksessä	**Lean-perusperiaate**	**Lean-pullonkaula**
Tuotoslistan koko	Poista hukka	Ylimääräiset ominaisuudet
Uudet teknologiat	Kunnioita ihmisiä	Puutteita osaamisessa
Projektien riittämätön läpinäkyvyys	Kunnioita ihmisiä	Johtajuuden puute
Tiettyjen henkilöiden ylikuormitus	Toimita nopeasti	Ylikuormitus
Puutteita prosessien noudattamisessa	Rakenna laatu ohjelman sisään	Kurin puute
Dokumentointi	Luo tietoa	-
Asiakaskommunikaatio	Luo tietoa	Ei nopeaa palautetta

8.2 Tutkimuksen luotettavuus

Tutkimuksen osalta on hyvä tarkastella kriittisesti sen luotettavuutta. Tämän tutkimuksen osalta tarkastellaan ensin tutkimuksen toteutusta yleisesti. Tämän jälkeen käydään läpi tutkimusaineistoa. Lopuksi arvioidaan tutkijan omistussuhteen vaikutusta tutkimustuloksiin.

Luotettavuuden arvioinnissa keskeisiä käsitteitä ovat perinteisesti olleet etenkin määrällisessä tutkimuksessa reliabiliteetti ja validiteetti. Laadullisen tutkimuksen luotettavuutta ei voida kuitenkaan arvioida täysin samalla tavalla kuin määrällisen. Validiteetissa on kyse siitä, onko tutkimus pätevä, eli onko se perusteellisesti tehty ja ovatko saadut tulokset sekä tehdyt johtopäätökset oikeita. Reliabiliteetillä tarkoitetaan luotettavuutta. Määrällisessä tutkimuksessa esimerkiksi mittausprosessin reliabiliteetti on mittaustapahtuman ominaisuus erotuksena mittarin ominaisuudesta. Laadullisessa tutkimuksessa voidaan arvioida esimerkiksi metodin reliaabeliuutta [SKP06]. Reliabiliteetin käyttämisen mahdollisuudesta laadullisen tutkimuksen yhteydessä on ristiriitaisia näkemyksiä [SKP06], mutta luotettavuuden parantamisen keinot ovat kuitenkin pääpiirteissään selkeitä: esimerkiksi tekstejä analysoitaessa tulisi tehdä selkeitä kategorisointeja ja koodauksia [SKP06].

Tässä tutkielmassa käytiin läpi Lean-ohjelmistokehitystä ja kehitysnopeutta rajoittavia tekijöitä tapausyrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Tutkimus toteutettiin tapaustutkimuksena, ja siitä saatiin mielenkiintoista tietoa. Tutkimus koski tapaustutkimuksena tapausyritystä, joten saadut tutkimustulokset ja johtopäätökset eivät ole suoraan yleistettävissä muihin yrityksiin tai organisaatioihin. Käytettyä tutkimustapaa ja -prosessia sen sijaan voidaan soveltaa myös muihin tapauksiin, jolloin saadaan tuloksia, joita voidaan hyödyntää kyseisissä tapauksissa.

Tutkimusaineistona käytettiin jälkipuintipalaverien pöytäkirjoja sekä vaatimusten toteuttamisesta kerättyä mittausdataa. Lisäksi aineistoja tuki tutkijan tutkimuspäiväkirja, joka sisälsi lähinnä tutkimuksen aikana tehtyjä havaintoja. Jälkipuintipalaverien pöytäkirjoista löytyneitä ”parannettavaa”-kohtia oli aineistossa yhteensä 121 kappaletta. Toteutetuista vaatimuksista oli tutkimuksen ajalta mittaustietoa 63 vaatimuksen osalta sekä ennen tutkimusta kerättyä dataa 42 vaatimuksen osalta. Aineistoa oli käytössä siis varsin rajallisesti, mikä aiheuttaa epävarmuutta tutkimustuloksiin. Tuloksia ei voida aineiston niukkuuden takia pitää täysin luotettavina, mutta ne antavat kyllä suuntaa ohjelmistokehityksen nopeutta rajoittavista tekijöistä tapausyrityksessä. Aineistoa oli käytössä kahdesta hieman eri näkökulmasta, mikä parantaa tutkimuksen luotettavuutta. Aineistoa pyrittiin myös katego-

risoimaan ja jaottelemaan mahdollisimman selkeästi tutkimuksen luotettavuuden parantamiseksi. Eräs ongelma aineistossa ja sitä kautta tutkimuksessa oli, että se ei ottanut suoraan kantaa syntyneiden ohjelmistovirheiden määrään. Laatuongelmat, eli virheiden korjaus, voisi luonnollisesti olla kehitysnopeutta rajoittava tekijä, mutta tähän ei kyseistä aineistoa käyttäen ollut mahdollisuutta ottaa kantaa, joten se jäi selvittämättä.

Aineiston perusteella lasketut eri työvaiheiden kestot laskettiin yksinkertaistetulla kaavalla, mikä aiheuttaa tuloksiin pientä heittoa. Isolla aineistolla tämä ei olisi ongelma, koska yksittäisten heittojen vaikutus keskiarvoihin olisi pieni. Tämän tutkimuksen aineistolla on kuitenkin mahdollista, että esimerkiksi viikonlopun yli toteutusvaiheessa ollut vaatimus on näennäisesti kasvattanut toteutusvaiheen mitattua kestoa.

Tutkija on itse osaomistajana tapausyrityksessä, mikä mahdollisesti heikentää tutkimuksen luotettavuutta. Tutkimus pyrittiin tekemään mahdollisimman objektiivisesti. Täydellinen objektiivisuus ei kuitenkaan ole mahdollista. Kenenkään ei ole mahdollista irrottautua itsestään ja sulkea täysin pois omaa ajatteluaan [SKP06]. Täten ei voida kokonaan pois sulkea mahdollisuutta, että tutkijan päivätyö tapausyrityksen palveluksessa olisi vaikuttanut tutkimustulosten tulkintaan. Toisaalta tutkijalla oli käytössään enemmän tietoa kuin ehkä ulkopuolisella tutkijalla olisi ollut mahdollista saada, mikä saattoi osaltaan parantaa tutkimuksen luotettavuutta. Tutkijalla on myös pitkä kokemus tapausyrityksestä, mikä saattoi toisaalta helpottaa tutkimustulosten tulkitsemista, mutta toisaalta aiheuttaa subjektiivisuutta johtopäätöksiin. Todennäköisesti tuloksia voidaan kuitenkin pitää siinä määrin luotettavina, että tapausyritys pystyy hyödyntämään niitä Lean-ohjelmistokehityksen ja ohjelmistokehitysnopeuden parantamisessa.

8.3 Suosituksia ja jatkotutkimustarpeita

Tutkimustulosten ja johtopäätösten perusteella laaditaan suosituksia, jotka liittyvät löydettyjen kehitysnopeutta rajoittavien tekijöiden poistamiseen. Tällä pitäisi TOC-teorian mukaan olla järjestelmän tuottavuutta parantava vaikutus. Tässä tapauksessa ”järjestelmä” on tapausyrityksen ohjelmistotuotanto. Suositusten lisäksi tapausyritykselle esitetään joitain jatkotutkimusmahdollisuuksia sekä muutamia ideoita, joita tapausyritys voisi mahdollisesti haluta kokeilla.

Luvussa 8.1 esiteltiin johtopäätöksinä tekijöitä, jotka mahdollisesti rajoittavat tapausyrityksessä ohjelmistokehitysnopeutta. Taulukossa 8.3 on esitetty suosituksia, joilla näitä rajoittavia tekijöitä voisi olla mahdollista karsia tai niiden vaikutusta pie-

nentää. Suositukset on laadittu tutkimuksen aineiston, tutkimustulosten ja johtopäätösten perusteella. Lisäksi suositusten laatimisessa on hyödynnetty tutkijan tekemiä omia havaintoja sekä tutkijan omakohtaista kokemusta tapausyrityksen toiminnasta. Suositukset ovat pikemminkin ehdotuksia kokeiluiksi, joilla voisi olla positiivinen vaikutus tapausyrityksen ohjelmistokehitysnopeuteen, kuin valmiita ratkaisuita rajoitteiden poistamiseksi.

Taulukossa 8.3 esitetyistä suosituksista tuotoslistan koon pienentämisellä voisi olla merkittävä vaikutus läpimenoaikoihin. Taulukossa esitettyjen keinojen lisäksi on huomioitava että myös tahtiajan pienentäminen pienentää yleensä tuotoslistan kokoa, koska tällöin vaatimuksia pystytään toteuttamaan nopeammin. Toisin sanoen taulukon muilla suosituksilla on todennäköisesti epäsuora positiivinen vaikutus myös vaatimuslistan kokoon, koska ne mahdollisesti pienentävät tahtiaikaa.

Tutkimustulosten tarkastelussa luvussa 7.1 havaittiin, että jälkipuinneissa ei keskustella juurikaan tuotteesta tai tuoda siihen liittyviä parannusehdotuksia, mikä on mielenkiintoinen asia. Tuotteeseen liittyviksi asioiksi lasketaan kaikki tuotteen ominaisuuksista ohjelmiston teknisiin yksityiskohtiin liittyviin asioihin asti. Tässä voisi olla tapausyritykselle eräs selvitettävän arvoinen asia olisiko tuotekeskustelua lisäämällä mahdollisuus saavuttaa suurempi kehitysnopeus tuotteen elinkaaren aikajaksolla. Tuotekeskustelun puuttuminen saattaa jo sinällään olla kehitysnopeutta rajoittava tekijä.

Luvussa 7.2.5 havaittiin, että osassa työvaiheita ei ole välillä lainkaan vaatimuksia työn alla. Eräs kokeilun mahdollisuus olisi pyrkiä pitämään kaikki työvaiheet siten kuormitettuina, että niissä olisi ainakin yksi vaatimus työstettävänä. Tämä tutkimus ei anna suoraa vastausta, olisiko tästä apua kehitysnopeuteen, mutta Lean-ajattelun mukaan tulisi pyrkiä luomaan yksiosainen virtaus, jossa artikkelit eivät jää odottamaan puskureihin. Mikäli yksi työvaihe vie huomattavasti muita pidempään, on luonnollista, että välillä muilla työvaiheilla on tyhjäkäyntiä. Eräs apukeinona tähän voisi olla työvaiheiden keston tasapainotus.

Mahdollisia jatkotutkimuksia varten voisi olla hyvä jatkaa saman tiedon keräämistä, jota käytettiin aineistona tässä tutkimuksessa. Tämä mahdollistaisi myöhemmin tarkemman analyysin ja vertailun esimerkiksi työvaiheisiin kuluneiden aikojen suhteen. Eräs jatkotutkimuksen aihe voisi olla tutkia tässä tutkimuksessa esitettyjen suositusten vaikutuksia tapausyrityksen ohjelmistokehitysnopeuteen, mikäli tapausyritys päättää toteuttaa suosituksissa esitettyjä asioita. Tämä tutkimus ei myöskään ottanut millään tapaa kantaa syntyneiden ohjelmistovirheiden määrään, jotka luonnollisesti saattavat rajoittaa kehitysnopeutta. Tätäkin asiaa voisi olla hyvä selvittää, ja tutkia sen vaikutuksia.

Taulukko 8.3: Suosituksia rajoitteiden poistamiseksi

Mahdollinen rajoittava tekijä tapausyrityksessä	**Suositus**
Tuotoslistan koko	Ylimääräiset ja vanhentuneet vaatimukset tulisi poistaa listalta sekä välttää liiallista vaatimusten keräämistä listalle etukäteen, mikäli niitä ei voida realistisesti toteuttaa kohtuullisen aikajakson sisällä.
Uudet teknologiat	Ennen uusien teknologien käyttöönottoa tulisi harkita lisääkö teknologia tuottavuutta riittävästi, jotta sen opetteluun kannattaa käyttää aikaa. Voi myös miettiä onko eri teknologioita käytössä liikaa ja tulisiko niiden määrää vähentää jolloin opeteltavaakin olisi vähemmän. Lisäksi välttämällä henkilöiden siirtoa eri teknologioiden parista muiden teknologioiden pariin, vältetään ylimääräisen opettelun ja koulutuksen tarvetta.
Projektien riittämätön läpinäkyvyys	Tulisi miettiä keinoja, joilla projektien läpinäkyvyyttä ja tulevaisuudensuunnitelmia saadaan parannettua.
Tiettyjen henkilöiden ylikuormitus	Lean-ideologian mukaisesti on parempi pitää niin henkilöiden kuin koneidenkin kuormitus liian alhaisena kuin liian korkeana. Ylikuormitusta tulisi siis välttää ja kuormitus tulisi jakaa tasaisesti eri henkilöiden kesken.
Puutteita prosessien noudattamisessa	Olisi hyvä selvittää mistä prosessien noudattamatta jättämiset ovat johtuneet ja reagoida tämän mukaan. Apukeinoja voisivat olla esimerkiksi koulutukset prosesseista tai tarkastuslistat, joiden mukaan toimitaan.
Dokumentointi	Tulisi löytää oikea tasapaino dokumentoinnin ja dokumentoimatta jättämisen välillä. Kaikkea ei varmasti tarvitse dokumentoida, mutta tärkeimmät asiat olisi hyvä olla ylhäällä.
Asiakaskommunikaatio	Pitäisi etsiä keinoja, joilla asiakasriippuvuuksia voidaan välttää. Olisiko esimerkiksi mahdollista käydä etukäteen läpi mistä asioista riippuvuuksia voi muodostua ja keskustella näistä asiakkaan kanssa?

8.4 Yhteenveto

Eri aineistoista saatujen tutkimustulosten perusteella pohdittiin, mitä nämä tulokset yhdessä tarkoittavat. Johtopäätöksinä sekä läpimenoaikaa että tahtiaikaa tulisi pyrkiä pienentämään. Läpimenoajan osalta rajoittavana tekijänä vaikuttaisi olevan tuotoslista ja tahtiajan osalta kehitystyövaihe. Koska kehitystyövaihe ei yksinään kerro paljoakaan, käytiin tätä vaihetta tarkemmin läpi, ja johtopäätöksinä löydettiin kuusi muuta rajoittavaa tekijää: uudet teknologiat, projektien riittämätön läpinäkyvyys, tiettyjen henkilöiden ylikuormitus, puutteita prosessien noudattamisessa, dokumentointi sekä asiakaskommunikaatio. Nämä löytyneet rajoitteet löytyvät dokumentointia lukuun ottamatta myös Murauskaiteen ja Adomauskasin [MA08] mallista.

Tutkimuksen luotettavuuden osalta muun muassa aineiston vähyys sekä tutkijan päivätyö ja omistajuus tutkittavassa yrityksessä voivat vaikuttaa tutkimustuloksiin. Tutkimustulokset ovat kuitenkin todennäköisesti riittävän luotettavia, jotta tapausyritys pystyy hyödyntämään niitä jatkossa.

Tunnistettujen ohjelmistokehityksen nopeutta rajoittavien tekijöiden vaikutusta voidaan mahdollisesti vähentää taulukossa 8.3 esitettyjä suosituksia käyttämällä. Suositusten vaikutusta ei pystytä kuitenkaan suoraan osoittamaan.

Jatkotutkimusmahdollisuuksina olisi esimerkiksi suosituksissa olevien keinojen vaikutusten tutkiminen tapausyrityksen ohjelmistokehitysnopeuteen, sekä ohjelmiston laadun tai laatuongelmien vaikutuksen selvittäminen tapausyrityksen ohjelmistokehitysnopeuteen.

9 Yhteenveto

Lean on ajattelutapa ja filosofia, joka on peräisin autojen valmistuksesta pohjautuen Lean-tuotantoon ja Toyotan tuotantojärjestelmään. Lean-tuotantoa voidaan pitää teollisuuden historian kolmantena tuotantojärjestelmänä käsityöläisyyden ja massatuotannon jälkeen.

Leanin ydinajatuksena on maksimoida asiakkaalle tuotettu lisäarvo minimoimalla hukka. Leaniin voidaan ajatella olevan kaksi eri lähestymistapaa: japanilainen ja länsimaalainen. Näistä länsimaalainen on japanilaista prosessikeskeisempi. Sen perusperiaatteet ovat arvo, arvovirta, virtaus, imu ja täydellisyys. Japanilaisen näkökulman perusperiaatteet ovat ongelmanratkaisu, ihmiset ja yhteistyökumppanit, prosessi sekä filosofia. Tavallisesti Lean-ajattelusta puhuttaessa tarkoitetaan länsimaalaista lähestymistapaa.

Lean-ohjelmistokehitys on Leanin soveltamista ohjelmistokehitykseen. Se ei tarjoa valmista konseptia tai ohjelmistokehitysprosessia, vaan pikemminkin ajattelutavan, jonka avulla voidaan pikkuhiljaa kehittää omaan ympäristöön parhaiten sopivat toimintatavat ja parantaa niitä jatkuvasti. Lean-ohjelmistokehityksessä hyödynnetään olemassa olevia käytänteitä ja teknologioita, jotka ovat Lean-ajattelun mukaisia ja tukevat sekä ihmisiä että prosesseja. Lean-ohjelmistokehitys on asiakaskeskeistä ja keskittyy tuottamaan asiakkaalle mahdollisimman paljon arvoa poistamalla prosesseista hukkaa, eli asiakkaalle lisäarvoa tuottamattomia asioita. Lean-ohjelmistokehityksen suurimpina vaikuttajina voidaan pitää Mary ja Tom Poppendieckiä. He ovat tunnistaneet seitsemän Lean-ohjelmistokehityksen perusperiaatetta: poista hukka, rakenna laatu ohjelman sisään, luo tietoa, lykkää sitoutumista, toimita nopeasti, kunnioita ihmisiä ja optimoi kokonaisuus.

Tapausyritys Sysdrone Oy:n osalta tutkittiin ohjelmistokehityksen nopeutta rajoittavia tekijöitä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen. Tutkimus toteutettiin tapaustutkimuksena. Tapausyritys on jyväskyläläinen ohjelmistotalo, joka valmistaa asiakaskohtaisesti räätälöityjä sovelluksia erityisesti terveysteknologian alalle. Tutkimuksen aineistoa kerättiin tammikuusta 2011 kesäkuuhun 2011. Aineistona toimivat ohjelmistokehitystiimien jälkipuintipalaverien pöytäkirjat sekä toteutetuista vaatimuksista kerätty mittausdata eri työvaiheisiin käytetyistä ajoista. Aineisto analysoitiin käyttäen sisällönanalyysia ja sisällönerittelyä. Tutkimuskysymyksenä oli *"Mitkä ovat merkittävimmät ohjelmistokehityksen nopeutta rajoit-*

tavat tekijät tutkittavassa yrityksessä puoli vuotta Lean-ohjelmistokehityksen käyttöönoton jälkeen?".

Tutkimustulokset osoittavat, että tapausyrityksessä vaatimusten läpimenoaika on pienentynyt merkittävästi puolen vuoden aikana, mutta tahtiaika on pysynyt suurin piirtein samana. Jälkipuintipalaverien pöytäkirjojen perusteella prosesseihin ja ihmisiin liittyviin asioihin on esitetty tutkimusjakson aikana eniten parannusehdotuksia.

Johtopäätöksinä kehitysnopeutta rajoittavia tekijöitä ovat läpimenoajan osalta tuotoslista ja tahtiajan osalta kehitystyövaihe. Kehitystyövaiheen rajoitteina toimii kuusi tekijää: uudet teknologiat, projektien riittämätön läpinäkyvyys, tiettyjen henkilöiden ylikuormitus, puutteet prosessien noudattamisessa, dokumentointi sekä asiakaskommunikaatio.

Tutkimustulosten luotettavuuden osalta tutkijan päivätyö ja omistajuus tapausyrityksessä voivat vaikuttaa tulosten objektiivisuuteen. Lisäksi aineiston vähyys heikentää tulosten luotettavuutta. Tutkimustulokset ovat kuitenkin riittävän luotettavia, jotta tapausyritys pystyy hyödyntämään niitä.

Tapausyritykselle esitettiin suosituksia tunnistettujen kehitysnopeutta rajoittavien tekijöiden vaikutusten vähentämiseksi. Suositusten mahdollista vaikuttavuutta kehitysnopeuteen ei arvioitu. Vaikutusten tutkimisessa olisi eräs jatkotutkimuksen mahdollisuus, mikäli tapausyritys päättää toteuttaa suosituksissa esitettyjä asioita.

Lähteet

[ASK10] Antanovich Andrei, Sheyko Anastasia ja Katumba Brian. Bottlenecks in the development life cycle of a feature - a case study conducted at ericsson ab. Master's thesis, University of Gothenburg, Department of Applied Information Technology, 2010.

[CSB08] Construx Software Builders Inc. Software development's classic mistakes 2008. Saatavilla WWW-muodossa <URL: `http://www.construx.com/File.ashx?cid=2796`>, 7 2008. Viitattu 8.10.2011.

[DL99] DeMarco Tom ja Lister Timothy. *Peopleware: productive projects and teams*. Dorset House Publishing, New York, Yhdysvallat, 1999.

[FC06] Flinchbaugh Jamie ja Carlino Andy. *The Hitchhiker's Guide to Lean - Lessons from the Road*. Society of Manufacturing Engineers, Yhdysvallat, 2006.

[Gre02] Grenning James. Planning poker or how to avoid analysis paralysis while release planning. Saatavilla WWW-muodossa <URL: `http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf`>, 2002. Viitattu 20.7.2011.

[HJS09] Hibbs Curt, Jewett Steve ja Sullivan Mike. *The Art of Lean Software Development*. O´Reilly Media, Sebastopol, Yhdysvallat, 2009.

[Hol07] Holweg Matthias, The genealogy of lean production, *Journal of Operations Management*, 25/2007, sivut 420–437.

[IPF+11] Ikonen Marko, Pirinen Elena, Fagerholm Fabian, Kettunen Petri ja Abrahamsson Pekka, On the impact of kanban on software project work - an empirical case study investigation, *Proceedings of the 16th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2011)*, 2011, sivut 305–314.

[JJ04] Järvinen Pertti ja Järvinen Annikki. *Tutkimustyön metodeista*. Opinpajan kirja, Tampere, Suomi, 2004.

[JP11] Jakobsen Carsten Ruseng ja Poppendieck Tom, Lean as a scrum troubleshooter, *AGILE Conference (AGILE) 2011*, 2011, sivut 168–174.

[Kni] Kniberg Henrik. Kanban kick-start example. Saatavilla WWW-muodossa <URL: `http://www.crisp.se/kanban/kanban-example.pdf`>. 18.11.2009.

[Kra88] Krafcik John F., Triumph of the lean production system, *Sloan Management Review*, 30/1988, sivut 41–52.

[KTAD07] Koichi Furugaki, Tooru Takagi, Akinori Sakata ja Daisuke Okayama, Innovation in software development process by introducing toyota production system, *Fujitsu scientific and technical journal*, 43/2007, sivut 139–150.

[Lea] Lean Enterprise Institute. What is lean? Saatavilla www-muodossa <URL: `http://www.lean.org/WhatsLean/Index.cfm`>. Viitattu 20.7.2009.

[Lik04] Liker Jeffrey K. *The Toyota Way - 14 Management Principles from the World's Greatest Manufacturer*. McGraw-Hill, New York, Yhdysvallat, 2004.

[Lik10] Liker Jeffrey K. *Toyotan tapaan*. Readme.fi, Helsinki, Suomi, 2010.

[LZE04] Lindstrom Lowell, Zannier Carmen ja Erdogmus Hakan. *Extreme Programming and Agile Methods - XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, ... (Lecture Notes in Computer Science).* Springer, Berliini, Saksa, 2004.

[MA08] Murauskaite Asta ja Adomauskas Vaidas. Bottlenecks in agile software development identified using theory of constraints (toc) principles. Master's thesis, IT University of Göteborg, Department of Applied Information Technology or Department of Computer Science, 2008.

[McC02] McConnell Steve. *Ohjelmistotuotannon hallinta*. Edita Publishing, Helsinki, Suomi, 2002.

[ML06] Morgan James M. ja Liker Jeffrey K. *The Toyota Product Development System - Integrating People, Process and Technology*. Productivity Press, New York, Yhdysvallat, 2006.

[MS05] Middleton Peter ja Sutton James. *Lean Software Strategies - Proven Techniques for Managers and Developers*. Productivity Press, New York, Yhdysvallat, 2005.

[PK06] Parnell-Klabo Emma, Introducing lean principles with agile practices at a fortune 500 company, *Agile Conference*, 2006, sivut 232–242.

[PP03] Poppendieck Mary ja Poppendieck Tom. *Lean Software Development*. Addison-Wesley, Crawfordswille, Indiana, Yhdysvallat, 2003.

[PP08] Poppendieck Mary ja Poppendieck Tom. *Implementing Lean Software Development*. Addison-Wesley, Stoughton, Massachusetts, Yhdysvallat, 2008.

[Pur10] Purojärvi Joni. Lääkinnällisen laitteen ohjelmistokehitys täydennetyllä scrum-mallilla. Pro-gradu tutkielma, Jyväskylän yliopisto, tietotekniikan laitos, 2010.

[SKP06] Saaranen-Kauppinen Anita ja Puusniekka Anna. Kvalimotv - menetelmäopetuksen tietovaranto. Saatavilla WWW-muodossa <URL: `http://www.fsd.uta.fi/menetelmaopetus/`>, 2006. Viitattu 30.12.2010.

[SVBP07] Sutherland Jeff, Viktorov Anton, Blount Jack ja Puntikov Nikolai, Distributed scrum: Agile project management with outsourced development teams, *40th Annual Hawaii International Conference on System Sciences*, Jan. 2007/2007, sivut 274a – 274a.

[SW07] Sayer Natalie J. ja Williams Bruce. *Lean for Dummies*. Wiley Publishing, Indianapolis, Indiana, Yhdysvallat, 2007.

[Toya] Toyota Industries Corporation. History - toyota industries corporation. Saatavilla WWW-muodossa <URL: `http://www.toyota-industries.com/corporateinfo/history/`>. Viitattu 20.7.2009.

[Toyb] Toyota Motor Corporation. Toyota: Company > history of toyota > 1867-1939. Saatavilla WWW-muodossa <URL: `http://www2.toyota.co.jp/en/history/1867.html`>. Viitattu 20.12.2009.

[TS02] Tuomi Jouni ja Sarajärvi Anneli. *Laadullinen tutkimus ja sisällönanalyysi*. Tammi, Helsinki, Suomi, 2002.

[Wan11] Wang Xiaofeng, The combination of agile and lean in software development: An experience report analysis, *AGILE Conference (AGILE) 2011*, 2011, sivut 1–9.

[Wik] Wikipedia. Sakichi Toyoda. Saatavilla WWW-muodossa <URL: `http://en.wikipedia.org/wiki/Sakichi_Toyoda`>. Viitattu 19.7.2011.

[WJ03] Womack James ja Jones Daniel. *Lean Thinking*. Simon & Schuster, Lontoo, Iso-Britannia, 2003.

[WJR07] Womack James, Jones Daniel ja Roos Daniel. *The Machine that Changed the World: The Story of Lean Production – Toyota's Secret Weapon in the Global Car Wars That Is Now Revolutionizing World Industry*. Free Press, New York, Yhdysvallat, 2007.