hyväksymispäivä arvosana

arvostelija

Kanban-ohjelmistokehitysmenetelmän vaikutuksista arvon muodostukseen ja ohjelmistokehittäjän työhön

Elena Pirinen

Helsinki 2.6.2010

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta – Fakultet –Faculty	Laitos – Institution – Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä – Författare – Author

Elena Pirinen

Työn nimi – Arbetets titel – Title

Kanban-ohjelmistokehitysmenetelmän vaikutuksista arvon muodostukseen ja ohjelmistokehittäjän työhön.

Oppiaine – Läroämne – Subject

Tietojenkäsittelytiede

Työn laji – Arbetets art – Level	Aika – Datum – Month and year	Sivumäärä – Sidoantal – Number of pages
Pro gradu -tutkielma	2.6.2010	99

Tiivistelmä – Referat – Abstract

Ohjelmistokehitysprosessi on joukko toimintoja, jotka luovat asiakkaan tarpeesta ohjelmistotuotteen. Ohjelmistokehityksen menetelmät ja työkalut sisältyvät ohjelmistoprosessiin. Myös ihmiset ovat osa ohjelmistoprosessia sillä ihmisten tiedot, taidot ja motivaatio vaikuttavat prosessin onnistumiseen. Jokainen ohjelmistoprosessi sisältää määrittely-, kehitys-, testaus- sekä evoluutiovaiheen. Kanban on ohjelmistokehityksen menetelmä, jossa näistä vaiheista muodostetaan työnkulkuja, jotka visualisoidaan seinätaululle.

Ohjelmistokehitysprosessin eri vaiheilla voi olla oppimisarvoa, mutta asiakkaalle tuotteella on arvoa vasta, kun se on valmis ja alkaa tuottaa asiakkaalle jotakin. Myös toimittamisen nopeudella on suuri arvo asiakkaalle. Tutkimuksen mukaan Kanban auttaa nopeuttamaan toimitusta ja virtaviivaistamaan ohjelmistoprosessia. Seinätaulun sarakkeissa olevat rajoitteet estävät tukoksia muodostumasta ja pienet tehtävät kulkevat nopeasti ohjelmistoprosessin läpi valmiiksi piirteiksi ohjelmistoon. Nopeuteen auttaa myös Kanban-taulun visuaalisuus. Tiimin johtaja näkee jatkuvasti seinätaululta onko joku toimettomana tai alkaako jonnekin muodostua tukoksia, jotka pitää selvittää. Kanbanin työnkulusta voidaan tehdä analyysi, jossa erotellaan arvoa tuottavat ja arvoa tuottamattomat työvaiheet, jotka pyritään poistamaan prosessista.

Ohjelmistoprosessi vaikuttaa ohjelmistokehittäjän työssä dokumentoinnin määrään, ongelmanratkaisun periaatteisiin, prosessin ja menetelmän visualisointiin, kokonaisuuden hallintaan, kommunikoinnin määrään, prosessin ja menetelmän ymmärtämiseen ja omaksumiseen, palautteen määrään, asioiden hyväksyttämisprosessiin ja työtehtävien valintaan. Ohjelmistokehittäjät kokevat Kanbanilla olevan näihin huomattavia vaikutuksia. Kanban-menetelmää käytettäessä dokumentaatiota on vain se, mitä asiakas tarvitsee, ongelmat tulevat heti esiin ja ne ratkaistaan heti, Kanban-taulu on hyvin visuaalinen ja motivoiva, pyrkimys on kokonaisuuden hallintaan, kommunikointia on paljon ja se on vapaata, menetelmä on helppo ymmärtää ja omaksua, palaute on välitöntä, menetelmä ei sisällä monimutkaisia hyväksyttämisprosesseja ja työtehtävien valinta on itsenäistä.

ACM Computing Classification System (CCS): D.2 [Software Engineering]

Avainsanat – Nyckelord – Keywords

Ketterät ohjelmistometodit, Lean, Kanban, ohjelmistokehitys, ohjelmistokehitysprosessi

Säilytyspaikka – Förvaringställe – Where deposited

Kumpulan tiedekirjasto, sarjanumero C-

Muita tietoja – Övriga uppgifter – Additional information

Sisältö

1 Johdanto

Ohjelmistoprosessi tuottaa asiakkaan tarpeesta ohjelmistotuotteen [Hum89]. Kaikki ohjelmistoprosessit sisältävät ohjelmiston määrittely-, kehitys-, testaus- sekä evoluutiovaiheen [Ost97]. Myös ihmiset ovat osa ohjelmistoprosessia [Hum89]. Ohjelmistokehittäjän työnä on luoda asiakkaan tarpeista ohjelmisto [Gra89]. Työssä edetään ohjelmistoprosessin eri vaiheiden kautta kunnes ohjelmisto täyttää asiakkaan tarpeen. Ohjelmistokehitys on luonteeltaan hyvin muuttuvaa [Roy70, Gra89] ja monimuotoista [Ost97].

Japanissa alettiin viime vuosisadalla käyttää autonvalmistuksessa uutta imuohjaukseen perustuvaa Kanban-menetelmää, jonka avulla tuotantoketjut saatiin tehokkaammiksi, kuin perinteisissä työntöpohjaisissa systeemeissä [WoJ90]. Kanbanin käytön tuloksena osia valmistettiin oikea määrä, oikeaa osaa, oikeaan aikaan, pääomaa ei ollut juuri sidottuna työn alla oleviin puolivalmisteisiin ja valmistuksen hallinto oli yksinkertaista. [Yam82]. Kanban-menetelmä on osa japanilaisten kehittämää Lean-filosofiaa, jonka periaatteisiin kuuluvat imuohjauksen ohella myös mm. henkilöstön arvostamisen sekä jatkuvan kehittämisen periaatteet [LiK04].

Poppendieck ja Poppendieck [PoP03], Ladas [Lad08], Kniberg [Kni09] ja Shinkle [Shi09] väittävät, että Kanban-menetelmä soveltuu myös ohjelmistokehitykseen. Tämän tutkielman tavoitteena on tutkia Kanban-menetelmän käyttöä ohjelmistokehityksessä erityisesti arvon muodostuksen ja toisaalta ohjelmistokehittäjän työn kannalta. Tässä luvussa selvitetään tutkimuksen taustaa, määritetään tutkimuskysymykset ja tutkimusmenetelmät sekä esitellään tutkielman rakenne.

1.1 Motivaatio

Ettl [Ett95] väittää, että Kanban-menetelmä noudattaa Lean-tuotannon periaatteita. Lean-tuotantoa on tutkittu kirjallisuudessa melko paljon. Womack ja kumppanit [WoJ90] esittelivät japanilaisten Toyotalla kehittelemän tuotantotalouden toimintatavan kirjassaan The Machine That Changed the World. Tämän jälkeen Lean-tuotantoa on toteutettu lukuisissa yrityksissä, ja sitä on tutkittu lukuisissa tutkimustapauksissa. Roth ja Labedz [RoL06] osoittavat tutkimuksessaan, kuinka Lean-tuotantoa toteutettiin

menestyksekkäästi Rockwell Collins -yrityksessä, joka tuottaa mm. elektroniikkaa lentokoneisiin. Myös Hartwellin ja Rothin [HaR06] tutkimuksen löydökset tukevat Rothin ja Lebedzin väitteitä. Heidän tutkimustapauksensa oli Ariens, joka on lumilinkoja ja pientraktoreita valmistava yritys Yhdysvalloissa. Hartwell ja Roth [HaR06] väittävät, että Lean-tuotantoon siirtyminen oli Ariensilla hyvin kannattavaa. Liker [Lik04] esittää kirjassaan, että Lean-tuotannon periaatteet on laajennettava tuotantosysteemistä koko organisaation kattavaksi Lean-filosofiaksi, jotta yritys saa Leanin edut parhaiten käyttöönsä.

Kanban on alun perin tuotantotalouden imuohjaukseen perustuva menetelmä, ja siitä on runsaasti tutkimuksia kuten Lambrechtin ja Saegaertin [LaS90] tutkimus, jossa esitetään, minkä verran tavaraa voi olla tuotantoputkessa, jotta maksimoidaan läpimenomäärät. Siha [Sih94] esittää, että mikäli tuotantoputken loppuosaan varataan enemmän kapasiteettia, saadaan parannettua läpimenomääriä ja silti pidettyä tuotantoputkessa oleva määrä minimissään. Myös Ettl [Ett95] esittelee erilaisia Kanban suunnitteluperiaatteita ja toteaa, että mikäli kapasiteettia varataan enemmän tuotantolinjan loppupuolelle, saavutetaan paras mahdollinen läpimenomäärä pienimmällä mahdollisella linjalla olevien tuotteiden määrällä.

Lean-filosofian ja erityisesti Kanbanin soveltuvuudesta ohjelmistotuotantoon on vähemmän tutkimusta. Poppendieck ja Poppendieck [PoP03] kuvaavat kirjassaan lyhyesti Kanbanin käyttöä ohjelmistotuotannossa, mutta keskittyvät enemmän esittelemään Lean-filosofian näkökulmaa ohjelmistotuotantoon. Middleton [Mid01] esittelee kaksi menestyksekästä tapaustutkimusta Lean-filosofian periaatteiden käyttämisestä ohjelmistojen kehityksessä. Raman [Ram98] puolestaan päätyy johtopäätökseen, että Lean-filosofiaa voidaan sovittaa mihin tahansa prosessiin, myös ohjelmistokehitysprosessiin. Poppendieck [Pop07] esittää, että ketterien ohjelmistokehitysmallien suosio ja menestys voidaan suurelta osin selittää ymmärtämällä Lean-filosofian periaatteet. Shinkle [Shi09] esittelee Kanban-menetelmän käyttöönoton Software Engineering Professionals -ohjelmistokehitysyrityksessä. Hänen mukaansa se, miten hyvin Lean-filosofian periaatteet ymmärretään tiimissä, korreloi suoraan siihen kuinka Kanban-menetelmän käyttö onnistuu. Ladas [Lad08] kuvailee kirjassaan miten ketterää ohjelmistokehitysmenetelmää käytettäessä voidaan siihen

yhdistää sekä Lean-filosofian periaatteet että Kanban-menetelmä.

Tämä tutkielma lähestyy Lean-filosofiaa ja etenkin Kanbania ohjelmistokehityksen näkökulmasta.

1.2 Tutkimuskysymykset

Tutkimuksen tavoitteena on kerätä ja analysoida tämänhetkinen tietämys Kanban-menetelmästä ohjelmistokehityksessä, erityisesti ohjelmistokehittäjien työn sekä arvon muodostumisen kannalta, sekä empiirisesti tutkia voidaanko kirjallisuudesta löydettyjä väitteitä tukea. Kirjallisuustutkimuksessa muodostetaan teoreettinen pohja ohjelmistokehitykselle, Kanban-menetelmälle sekä arvon muodostumiselle ja muodostetaan yhteenveto tutkimustuloksista. Tulosten perusteella rakennetaan teoreettinen malli, jolla voidaan kuvata ohjelmistokehittäjän työtä. Empiirinen osuus suoritetaan havainnointitutkimuksena sekä puolistrukturoituna teemahaastatteluna yhden tapaustutkimuksen avulla. Tavoitteena on löytää yhtäläisyyksiä sekä eroavaisuuksia kirjallisuudesta löydettyjen väitteiden suhteen. Empiirinen tieto kerättiin Helsingin Yliopiston Tietojenkäsittelytieteen laitoksen Software Factory –projektista [Abr10].

Tutkimuksessa analysoidaan Kanban-menetelmää kahdesta näkökulmasta, arvon muodostumisen sekä ohjelmistokehittäjien työn kannalta.Tutkimuskysymykset ovat:

1. Mitkä ovat Kanban-menetelmän kvalitatiiviset vaikutukset ohjelmistokehittäjän työhön?
    - 1.1. Miten voidaan määritellä ohjelmistokehittäjän työ?
    - 1.2. Millaisia muutoksia ohjelmistokehittäjä omaan työhönsä kokee Kanban-menetelmällä olevan?
2. Kuinka Kanban-menetelmä vaikuttaa arvon muodostumiseen?
    - 2.1. Miten ohjelmistojärjestelmän arvo muodostuu?
    - 2.2. Miten ohjelmistokehityksessä arvo voidaan määritellä Kanban-menetelmää soveltaen?

1.3 Tutkimusmenetelmät

Tutkimuksessa suoritettava tiedon keräys koostuu kirjallisuusselvityksestä, jossa kootaan yhteen olemassa olevat tutkimusten löydökset, sekä empiirisestä tapaustutkimuksesta. Yleensä tutkimuksessa otetaan lähtökohdaksi aiempi teoria tai malli, sillä se nopeuttaa työtä suuresti [Rou00]. Tässä tutkimuksessa ei kuitenkaan ole käytettävissä kirjallisuudesta löydettävää mallia, sillä tutkimuksen kohde on verrattain uusi. Uutta kartoittavassa tutkimuksessa tehtävänä on tulkita aineistoa uudesta näkökulmasta [Rou00]. Tässä tapauksessa näkökulma on Kanban ohjelmistokehitysmenetelmänä. Kirjallisuuden perusteella muodostetaan tutkijan oma tutkimusmalli, jota tämän jälkeen empiirisesti validoidaan.

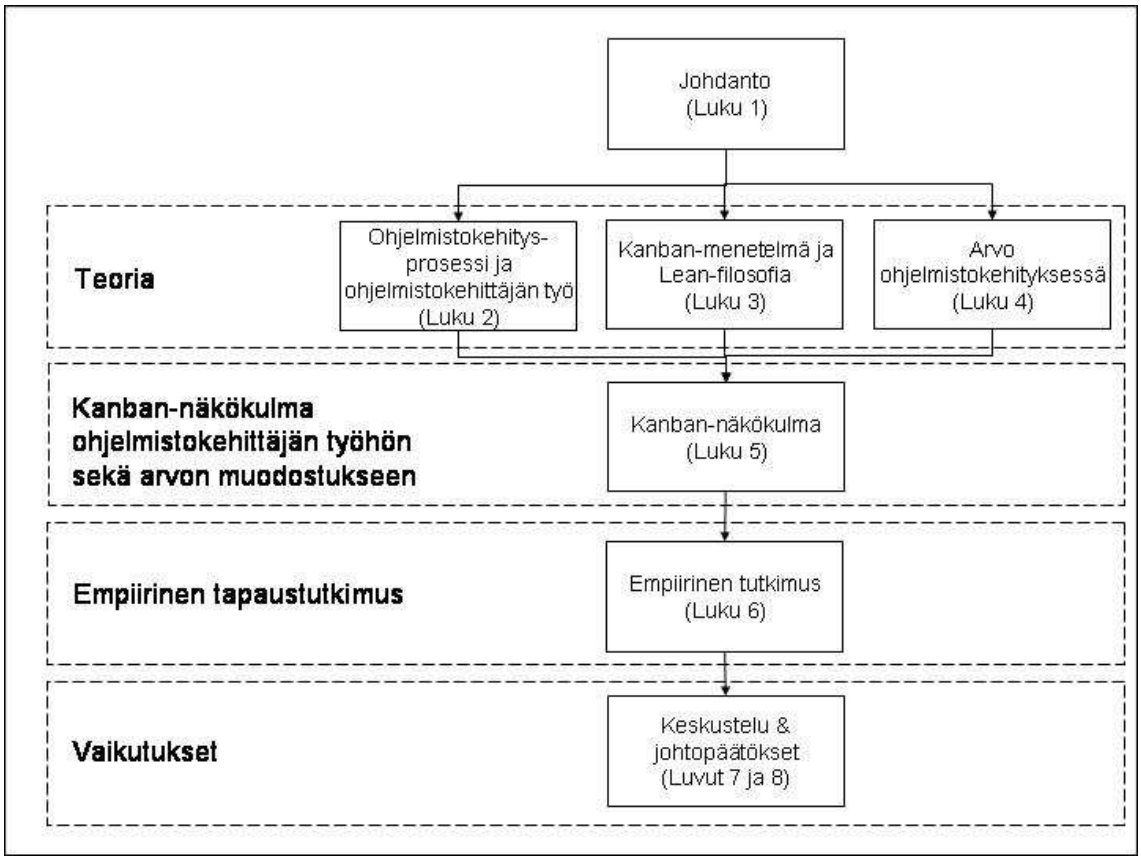
Tämän tutkimuksen empiirinen tutkimusmenetelmä on tapaustutkimus. Tapaustutkimus soveltuu poikittaistutkimukseen, jossa ei ole aikaperspektiiviä [Rou00]. Lähestymistapana käytetään menetelmätriangulaatiota. Siinä samassa tutkimuskohteessa käytetään eri menetelmiä [HiH01]. Tässä tutkimuksessa suoritetaan havainnointitutkimus ilman osallistumista sekä lopuksi haastattelututkimus. Haastattelututkimus suoritetaan puolistrukturoitua haastattelumentelmää käyttäen teemahaastatteluna [HiH01]. Teemahaastattelu soveltuu erityisesti tapaustutkimukseen, jolloin haastateltavien joukko on pieni [Rou00]. Teemahaastattelu on keskustelua, jolla on etukäteen päätetty tarkoitus. Se muistuttaa jokapäiväistä keskustelua, jota tutkija ohjailee siten, että se ei kokonaan erkane tutkimusongelmasta. Tutkija esittää avoimia kysymyksiä, eli sellaisia joihin ei ole valmiita vastausvaihtoehtoja. Haastateltava kertoo oman näkemyksensä asiasta. [Rou00].

Kerätty tieto analysoidaan tutkimuksen lopuksi teoreettisen mallin validoimiseksi ja tutkimuskysymyksiin vastaamiseksi. Laadullisen aineiston analyysissä voidaan nähdä kaksi vaihetta: havaintojen pelkistäminen ja tulosten tulkinta [Rou00]. Analysoinnin sekä tulosten esittelyn jälkeen tässä tutkielmassa ehdotetaan lopuksi tulevaisuuden käyttökohteita tutkimustuloksille.

1.4 Tutkielman rakenne

Työ on jaettu neljään osaan. Kuvassa 1 esitetään työn jaottelu johdantoon, teoriaosuuteen, Kanban-näkökulman käsittelyyn teorian perusteella, empiirisen

tapaustutkimuksen osuuteen ja vaikutusten analysointiin.



Kuva 1: Tutkielman rakenne.

Jatkossa käytetään virtaviivaisesta tuotannosta sekä filosofiasta sen englanninkielistä termiä Lean, sillä vakiintunutta suomenkielistä termiä ei vielä ole. Englanninkielisestä termistä waste näkee myös suomennoksia turha sekä hukka, tässä tutkielmassa käytetään termiä jäte. Asiakas nähdään tässä tutkimuksessa ohjelmiston tai tietojärjestelmän ostajaorganisaation edustajana, järjestelmän tilaajana. Ohjelmistokehittäjä puolestaan on tässä tutkimuksessa järjestelmän tekijä; ohjelmistokehittäjä voi toimia vaatimusten kerääjänä, järjestelmän määrittelijänä, suunnittelijana sekä toteuttajana. Hän voi tässä tutkimuksessa toimia missä tahansa näistä rooleista tai niiden yhdistelmistä.

2 Ohjelmistokehittäjän työ

Tämän luvun tarkoituksena on rakentaa mallia ohjelmistokehittäjän työn kuvaamiseksi. Luvun aluksi kuvataan ohjelmistoprosessia. Tämän jälkeen esitellään

ohjelmistokehityksen prosessimalleista kolme: vesiputousmalli, iteratiivinen ohjelmistokehitysmalli sekä ketterä ohjelmistokehitysmalli. Kaksi ketterän ohjelmistokehityksen menetelmää kuvataan tarkemmin, Extreme Programming sekä Scrum. Ohjelmistokehityksen prosessimallien käytännöistä pyritään löytämään ohjelmistokehittäjän työhön liittyviä erityispiirteitä, joista tutkijan oma, ohjelmistokehittäjän työtä kuvaava teoreettinen malli, koostuu. Lopuksi esitetään yhteenveto.

2.1 Ohjelmistoprosessin määrittely

Ohjelmistoprosessi (engl. software process) on joukko toimintoja jotka luovat asiakkaan tarpeesta ohjelmistotuotteen [Hum89]. Ohjelmistokehittäjät suorittavat nämä toiminnot. Ohjelmistoprosesseissa suoritetaan ainakin seuraavat toiminnot [Ost97]:

- ohjelmiston määrittely (engl. software spesification),
- ohjelmistokehitys (engl. software development),
- ohjelmiston validointi ja verifiointi (engl. software validation and verification) ja
- ohjelmiston evoluutio (engl. software evolution).

Määrittelyssä suunnitellaan ja mietitään ohjelmiston haluttu toiminta sekä sen rajoitteet. Kehitysvaiheessa tehdään ohjelmisto, joka vastaa määrittelyitään. Validointi tarkoittaa sen varmistamista, että ohjelmisto tekee sitä, mitä asiakas haluaa. Verifioinnissa puolestaan tarkastetaan, vastaako tehtävä ohjelmisto sille tehtyä määrittelyä, toteutuvatko kirjatut vaatimukset ja onko tuote vaatimusten mukainen. Ohjelmisto käy läpi kehitysvaiheita, evoluutiota, jotta asiakkaan tarpeet voidaan saavuttaa. Näitä eri neljää toimintoa voidaan erilaisissa ohjelmistoprosesseissa käyttää eri aikaan ja erimittaisina, mutta ne toistuvat jokaisessa ohjelmistoprosessissa [Ost97].

Humphrey [Hum89] väittää, että ohjelmistokehityksen menetelmät ja työkalut sisältyvät ohjelmistoprosessiin. Ohjelmistokehityksen menetelmät muodostavat käytännön ohjeet ohjelmiston rakentamiselle. Menetelmät käsittävät ohjelmistokehityksen tehtäviä kuten määrittely, suunnittelu ja toteutus, mutta myös yleisiä periaatteita kuten mallinnusperiaatteet, sekä mahdollisesti myös määrättyjä työkaluja. Humphreyn

[Hum89] mukaan myös henkilöt ovat osa ohjelmistoprosessia; ihmisten tiedot, taidot sekä motivaatio on otettava huomioon, mikäli ohjelmistoprosessia halutaan kehittää. Ohjelmistoprosessi on siis menetelmien, ihmisten ja työkalujen muodostama kokonaisuus, jolla on tietty päämäärä: luoda asiakkaan tarpeesta ohjelmisto.

Humphery [Hum89] esittää, että jotta ohjelmistoprosessi olisi tehokas, olisi sen oltava hallittavissa. Kun ohjelmistokehittäjiltä kysytään, mitkä aiheuttavat heidän työssään eniten ongelmia, vastaus on teknologian sijaan: huonosti määritelty ohjelmistoprosessi, epäjohdonmukainen toteutus ja huono ohjelmistoprosessin hallinta. Graham [Gra89] väittää samaa. Hänen mukaansa ohjelmistojen prosessimallit ovat kehittyneet, jotta ohjelmistoprosessia voitaisiin ohjata, hallita ja kontrolloida. Myös Komi-Sirviö [Kom04] tukee väitettä ja osoittaa, että hyvin hallittu ohjelmistoprosessi on nykyisten ohjelmistoyritysten strategista ydinosaamista. Hän esittää, että ohjelmistoprosessia pyritään kehittämään, sillä se vaikuttaa suoraan lopputuloksen, ohjelmiston, laatuun. Hän ehdottaa, että prosessin hyvällä hallinnoinnilla voidaan myös lisätä ennustettavuutta muuttuvassa ohjelmistokehityksen kentässä.

Osterweil [Ost87] esittää, että ohjelmistoprosessimalli kuvaa jonkin jonkin spesifisen ohjelmistoprosessin tai osan siitä. Hänen mukaansa prosessimalli on yksinkertaistettu esitys ohjelmistoprosessista ja kuvaa prosessia tietystä perspektiivistä. Myös Komi-Sirvion [Kom04] mukaan prosessimalli ohjaa ohjelmistokehitystä kuvaamalla ohjelmistoprosessin.

Osterweil [Ost87] väittää, että prosessimallin on oltava niin tarkka ja täsmällinen kuin mahdollista, jotta ohjelmistokehitys olisi tehokkainta mahdollista. Poppendieck ja Poppendieck [PoP03] sekä Kniberg [Kni09] ovat eri mieltä. Heidän mukaansa prosessimallin ei ole tarpeen kuvata prosessia monimutkaisesti ja tarkasti, vaan yksinkertainen prosessimalli riittää ainakin ohjelmistokehittäjien tarpeisiin. McConnell ja Tripp [McT05] väittävät, että mikäli prosessimalli on kovin monimutkainen, tekevät ohjelmistokehittäjät työtään omalla tavallaan mallista riippumatta.

Prosessimalli sisältää ohjelmistoprosessin lisäksi prosessiin kuuluvat menetelmät ja työkalut [CoW98]. Humphrey [Hum89] väittää, että prosessimallin menetelmän testaamiseksi riittää yksinkertainen keino: jos menetelmä palvelee yksittäisen ohjelmistokehittäjän tarpeita, se palvelee myös tiimiä ja hyvinkin suuria projekteja.

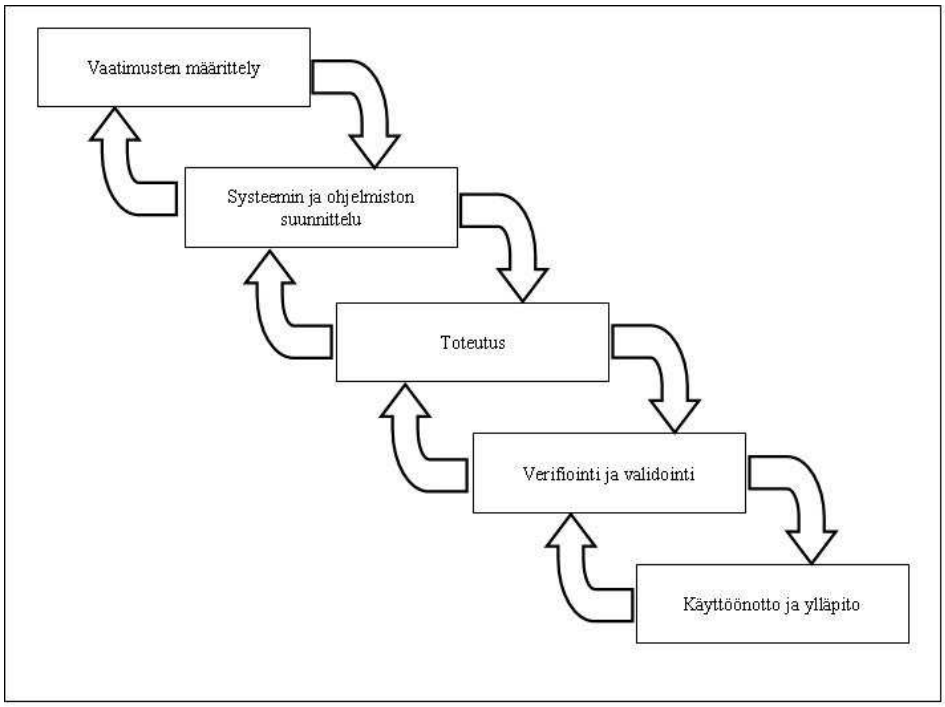
Tässä tutkielmassa menetelmää katsotaan yksittäisen ohjelmistokehittäjän työn näkökulmasta, mutta mikäli Humphreyn väite pitää paikkansa, saattavat tutkimustulokset vaikutuksista ohjelmistokehittäjän työhön hyödyttää myös laajemmassa perspektiivissä.

2.2 Ohjelmistokehityksen prosessimallit

Luvussa esitellään kolme ohjelmistokehityksen prosessimallia tarkemmin: vesiputousmalli, iteratiivinen prosessimalli sekä ketterä prosessimalli. Tämän jälkeen kuvaillaan ketteriä menetelmiä, niiden evoluutiota, ja esitellään yksityiskohtaisemmin kaksi menetelmää: Scrum sekä Extreme Programming. Lopuksi vertaillaan esiteltyjä prosessimalleja.

2.2.1 Vesiputousmalli

Perinteisesti ohjelmistokehityksessä on käytetty prosessimallina vesiputousmallia, jossa ohjelmistoprosessin toiminnot nähdään erillisinä toisiaan seuraavina vaiheina [Kom04]. Royce [Roy70] esittää vesiputousmallin periaatteen. Kuvassa 2 esitetään, kuinka ohjelmistoprosessi sisältää seuraavat vaiheet: vaatimusten määrittely, ohjelmiston suunnittelu, toteutus, testaus, käyttöönotto ja ylläpito. Vaiheesta toiseen edetään vesiputousmaisesti siten, että jokainen vaihe tulee saada päätökseen ennen seuraavan aloittamista.



Kuva 2: Vesiputousmallin eri vaiheet Roycen [Roy70] mukaan.

Ensimmäisessä vaiheessa projekti aloitetaan ja järjestelmän vaatimukset kerätään. Vaatimukset jäädytetään mahdollisimman aikaisessa vaiheessa, jotta päästään ohjelmiston suunnittelussa eteenpäin [Gra89]. Järjestelmän ja ohjelmiston suunnitteluvaihe sisältää projektin aikataulutuksen ja järjestelmän määrittelyn sekä suunnittelun. Toteutusvaiheessa järjestelmä ohjelmoidaan. Jokainen vaihe tuottaa dokumentaatiota, jotka hyväksytetään ennen seuraavan vaiheen aloittamista [Kom04]. Mikäli ongelmia ilmenee, ne säästetään korjattaviksi myöhemmin, niille tehdään tilapäinen kiertotie (engl. workaround) tai niitä ei huomioida lainkaan [Som00]. Prosessin lopputuloksena asiakas saa kerralla kokonaisen järjestelmän [Gra89]. Hyvin usein asiakkaan vastaus on lopputuotteen vihdoin saatuaan: ”Tätä minä en halunnut, vaikka pyysinkin juuri tällaista” [Gra89].

Ohjelmistoprosessia ei kuitenkaan voida viedä läpi täysin lineaarisesti, vaan jonkin asteista iteraatiota ilmenee aina [Roy70], kuten kuvassa 2 esitetään. Royce [Roy70] ehdottaa, että ideaalitapauksessa iteraatiota olisi ainoastaan kahden peräkkäisen vaiheen välillä. Käytännössä jokaisesta vaiheesta voidaan joutua muuttamaan mitä tahansa edellä ollutta vaihetta [Roy70]. Tämä on vesiputousmallissa sekä kallista että aikaa

vievää [BaT75, Gra89].

Basili ja Turner [BaT75] väittävät, että vesiputousmallin käyttäminen vaatii, että sekä ongelmakenttä että rakennettava ohjelmistokokonaisuus ymmärretään hyvin eivätkä ne ole täysin uusia projektin henkilöille. Graham [Gra89] on samaa mieltä ja väittää, että vesiputousmalli saattaa sopia pienten ja lyhytkestoisten ohjelmistojen, joiden muuttumattomat vaatimukset tunnetaan hyvin etukäteen, kehitysprosessiksi. Myös Larman [Lar07] tukee Basilin ja Turnerin väitettä ja esittää, että perinteinen ohjelmistonkehityksen prosessimalli, vesiputousmalli, sopii hyvin sellaisten tuotteiden tuottamiseen, jotka ovat suhteellisen samanlaisia, tuotantoprosessi ei juuri muutu, vaikka tuote hieman uudistuukin ja prosessissa on aina samanlaisina toistuvia osia tuotteesta toiseen. Tällaista on esimerkiksi puhelinten valmistus. Tietokoneohjelmistot sopivat kuitenkin hänen mukaansa harvoin massavalmistuksen tuotteeksi, sillä niitä luodessa tehdään yleensä täysin uutta tuotetta, jolloin prosessi on tapauskohtainen.

Brooken [Bro87] mukaan mikään yksittäinen tekniikka tai tapa tehdä asioita ei kerralla ratkaise asioita ohjelmistokehityksessä. Ohjelmistokehitys on luonteeltaan niin monimutkaista ja siihen liittyvä ympäristö niin muuttuvaa, että ei ole olemassa pikaratkaisua, jolla kaikki ohjelmistokehityksen ongelmat saataisiin poistettua. Ohjelmistokehityksessä erityisesti määrittely, suunnittelu sekä testaus ovat aikaa vieviä toimintoja ja tulevat aina olemaan sellaisia. Samaa väittää myös Osterweil [Ost97]. Hänen mukaansa ohjelmistokehityksessä yksittäinen prosessimalli on harvoin riittävä, vaan tarvitaan useiden eri mallien yhdistelyä eikä sekään aina riitä ratkaisemaan ohjelmistokehityksen monimuotoisuuden aiheuttamia ongelmia.

2.2.2 Iteratiivinen prosessimalli

Larman [Lar07] esittää, että iteratiivisen ohjelmistokehityksen tarkoituksena on jalostaa tuotetta vähitellen, osina, kunnes asiakkaan tarvitsema järjestelmä on valmis. Yleisesti ottaen määrittely, suunnittelu ja testaus eivät ole niin erillisiä, kuin vesiputousmallissa esitetään [Lar07]. Esimerkiksi suunnittelutyön laajuutta on hankala ennakoida ennen kuin on toteuttamalla varmistettu suunnittelun toimivuus [Lar07]. Sekä ohjelmistokehittäjän että asiakkaan tietämys asiasta lisääntyy prosessin aikana, jolloin

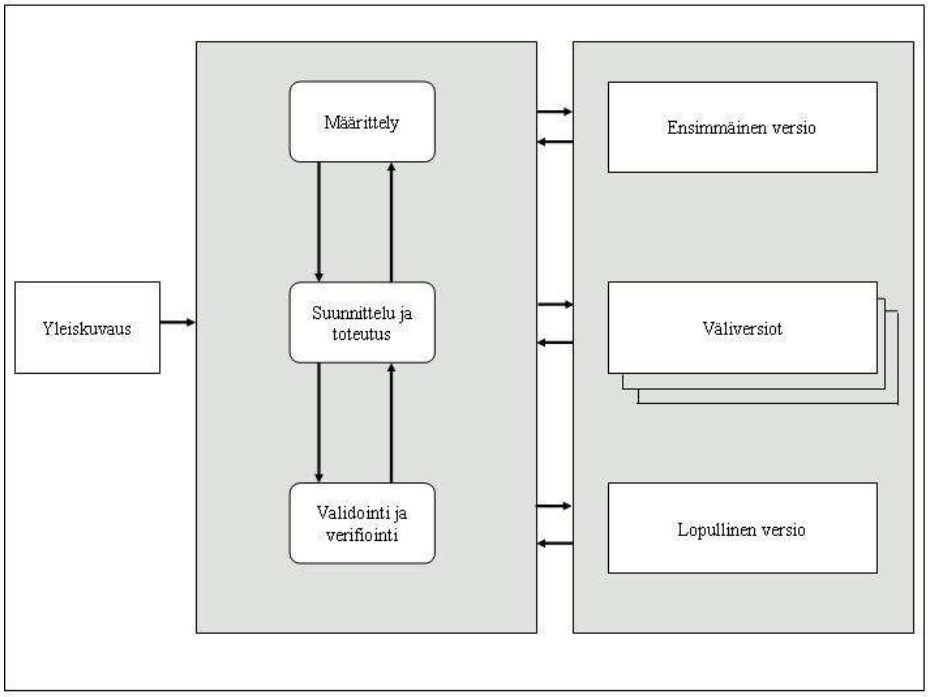
olisi hyödyllistä voida palata edellisiin vaiheisiin uuden tietämyksen kera [Gra89].

Iteratiiviinen ohjelmistokehitysprosessimalli perustuu useisiin peräkkäisiin iteraatiokierroksiin, joista jokainen on oma miniprojektinsa koostuen vaatimusten analysoinnista, suunnittelusta, ohjelmoinnista ja validoinnista sekä verifioinnista. Jokaisen kierroksen päämääränä on osaltaan valmiin tuotteen julkistaminen [Lar07]. Yleensä kullekin iteraatiolle määritellään aikarajat, jotka eivät saa muuttua (engl. timeboxing) [Lar07].

Basil ja Turner [BaT75] esittivät jo vuonna 1975, että iteratiivisia ohjelmistokehitysprosesseja on kahdenlaisia:

- sellaisia, joissa ensimmäinen versio on ratkaisu koko ongelmaan, mutta on vain lopullisen ratkaisun luuranko, jonka päälle rakennetaan lisäosia kokonaisuuden aikaansaamiseksi ja
- sellaisia, joissa asiakkaan vaatimusten kokonaisjoukosta valitaan kuhunkin iteraatioon osajoukko jonkin valmiin osakokonaisuuden tuottamiseksi.

Kuvassa 3 esitetään iteratiivinen ohjelmistokehityprosessi Basilin ja Turnerin [BaT75] mukaan. Ensin tehdään järjestelmän yleiskuvaus, jossa määritellään järjestelmän vaatimukset. Yleiskuvauksen jälkeen tehdään määrittely, kehitystyö ja validointi sekä verifiointi kussakin iteraatiossa. Jo ensimmäinen versio on ratkaisu koko ongelmaan. Sitä täydennetään ja parannetaan iteraatioilla. Olemassa olevaa ohjelmiston toteutusta joudutaan myös uudelleenorganisoimaan, jotta lopullisen järjestelmän toteutus olisi laadukasta. Iteraatiot muodostavat tuotettavan ohjelmiston eri versioita, kunnes lopullinen asiakkaan tarpeet täyttävä ohjelmisto on valmis.



Kuva 3: Iteratiivisen ohjelmistokehityksen malli Basilin ja Turnerin [Bat75] mukaan.

Graham [Gra89] väittää, että iteratiivisessa ohjelmistokehityksessä on se etu, että asiakkaan ei tarvitse tietää kaikkia vaatimuksia ennakkoon ja yhdellä kertaa vaan hän voi tarkentaa niitä sitä mukaa, kun osia ohjelmistotuotteesta valmistuu. Asiakas saa huomattavasti nopeammin toimivia osakokonaisuuksia kuin vesiputousmallissa. Muutos voidaan kääntää epätoivotusta asiasta eduksi, asiakkaan parhaaksi. Larman [Lar07] on samaa mieltä ja esittää, että on hyvä, että asiakkaan kommentteja saadaan jo prosessin aikaisessa vaiheessa. Iteratiivisen kehityksen etuna on se, että asiakas voi tarkentaa vaatimuksiaan prosessin aikana.

Iteratiivisessa ohjelmistokehityksessä on haasteensa. Milloin iteraatioiden tekeminen pitäisi lopettaa ja todeta, että ohjelmisto on valmis? Basili ja Turner [BaT75] esittävät, että kokonaisuus pidetään hallinnassa siten, että ohjelmistoprojektissa pidetään yllä tehtävien asioiden luetteloa, joka sisältää kaikki ne asiat, jotka tarvitaan lopullisen tuotteen aikaansaamiseksi. Poppendieck ja Poppendieck [PoP03] ovat eri mieltä. Heidän mielestään iteratiivisessa kehityksessä valmiin ohjelmistotuotteen määrittäminen on todellinen ohjelmistokehityksen ongelma. Kun palautetta tulee jatkuvasti, tehdään uusia iteraatioita ja muodostuu loputon syklien sarja.

Iteratiivinen ohjelmistokehityksen prosessimalli ei sellaisenaan ole visuaalinen [Som00]. Ohjelmistokehittäjien sekä yritysjohtajien on hankalaa tietää, missä ohjelmistoprosessissa ollaan menossa, kun ohjelmistoa kehitetään jatkuvasti lyhyissä sykleissä. Jatkuva muutos myös haastaa tuotettavan järjestelmän rakennetta [BaT75]. Kun asiakkaan vaatimukset muuttuvat, joudutaan olemassa olevaa ohjelmakoodia kenties muuttamaan ja ohjelmiston rakenteesta tulee kokonaisuutena yhä sekavampi ja vaikeampi hallita [PoP03].

Dokumentaatiota iteratiivisessa ohjelmistokehityksessä on Basilin ja Turnerin [BaT75] mukaan pääasiassa tehtävien asioiden lista. Projektissa pidetään yllä tehtävien asioiden listaa, jolta kuhunkin iteraatioon valittavat tehtävät asiat valitaan. Kun jokaisesta iteraatiosta säilytetään oma tehtävien asioiden listansa, toimii se projektin päädokumentaationa.

Ohjelmistokehittäjät ja yritysjohtajat tarvitsisivat työkaluja, jotta prosessissa edistymistä voitaisiin mitata [Som00]. Basilin ja Turnerin [BaT75] tehtävien asioiden lista toimii mittarina, sillä siitä voidaan nähdä kuinka paljon asioita on vielä tekemättä sekä toisaalta, kuinka paljon on jo tehty. Poppendieck ja Poppendieck [PoP03] ovat samaa mieltä ja esittävät kirjassaan, että tehtävien asioiden listan avulla voidaan mitata projektissa edistymistä.

Yleisesti luullaan, että ketterät menetelmät toivat mukanaan iteratiivisen ohjelmistojenkehitysprosessin, mutta todellisuudessa iteratiivisia metodeita on käytetty jo kauan, 1960-luvulta lähtien Nykyisin käytettävät iteratiiviset menetelmät ovat kuitenkin laajempia ja niitä yhdistellään muihin prosessimalleihin tuotettavan ohjelmistojärjestelmän tarpeiden mukaan [Lar07].

2.2.3 Ketterä prosessimalli

Graham [Gra89] mainitsee seuraavaa: ohjelmiston kehittäminen vaatimusten perusteella on kuin kävelisi veden päällä - se on helpompaa, jos vesi on jäässä. On kuitenkin helpompaa jäädyttää lampi kuin valtameri. Ketterät menetelmät ovat syntyneet tähän tarpeeseen. Nyfjord [Nyf08] väittää, että ketterä ohjelmistokehityksen prosessimalli on syntynyt, koska jäykät, perinteiset ohjelmistojenkehitysmallit, kuten vesiputousmalli, eivät ole pystyneet vastaamaan jatkuvaan muutokseen ja epävarmuuden sietämiseen

ohjelmistoja kehitettäessä. Perinteisesti, kun aikataulut ja kustannukset on yritetty tietää varmasti etukäteen, on järjestelmän vaatimukset lyöty lukkoon mahdollisimman aikaisin eivätkä muutokset ole olleet tervetulleita [Gra89]. Kuitenkin, mikäli esimerkiksi vaatimukset pitää vahvistaa huomattavasti aikaisemmin kuin valmis tuote voidaan toimittaa, ovat liiketoimintaympäristö sekä ohjelmiston vaatimukset muuttuneet alkuperäisistä eikä valmistuva ohjelmisto ole välttämättä enää lainkaan tarpeellinen [Gra89]. Pitkän järjestelmäprosessin aikana muuttuvat niin ohjelmiston vaatimukset kuin asiakkaan asettama priorisointijärjestyskin [Pre05]. Myöskään etukäteen tehty aikataulu- ja kustannusarvio ei yrityksestä huolimatta ole osoittautunut oikeaksi [Gra89]. Etenkin nopeasti muuttuvissa Internet-pohjaisissa ohjelmistoissa sekä mobiililaitteiden ohjelmistokehityksessä on tarvetta valmistaa ohjelmistoja nopeasti sekä ketterästi [AbS04].

Ketterät ohjelmistokehitysmenetelmät ovat iteratiivisia metodeita, joissa projekti pilkotaan pieniin erikseen suoritettaviin osiin. Menetelmät korostavat muutoksen pitämistä hyvänä asiana, joustavuutta, sekä lisäksi kommunikoinnin ja palautteen merkitystä ohjelmistojenkehitysprosessissa. Ketterät metodit luottavat tiimityöskentelyyn ja asiantuntemukseen, ja ryhmien sisäinen roolijako hämärtyy ja tilalle tulee tiimivetoisuus ja osaamispääoman hallinta. [Lar07].

Ketteriä menetelmiä ei voi tarkasti määritellä, koska eri menetelmissä käytännöt vaihtelevat. Yleisiä periaatteita ketterässä ohjelmistokehitysprosessissa ovat kuitenkin Larmanin [Lar07] mukaan:

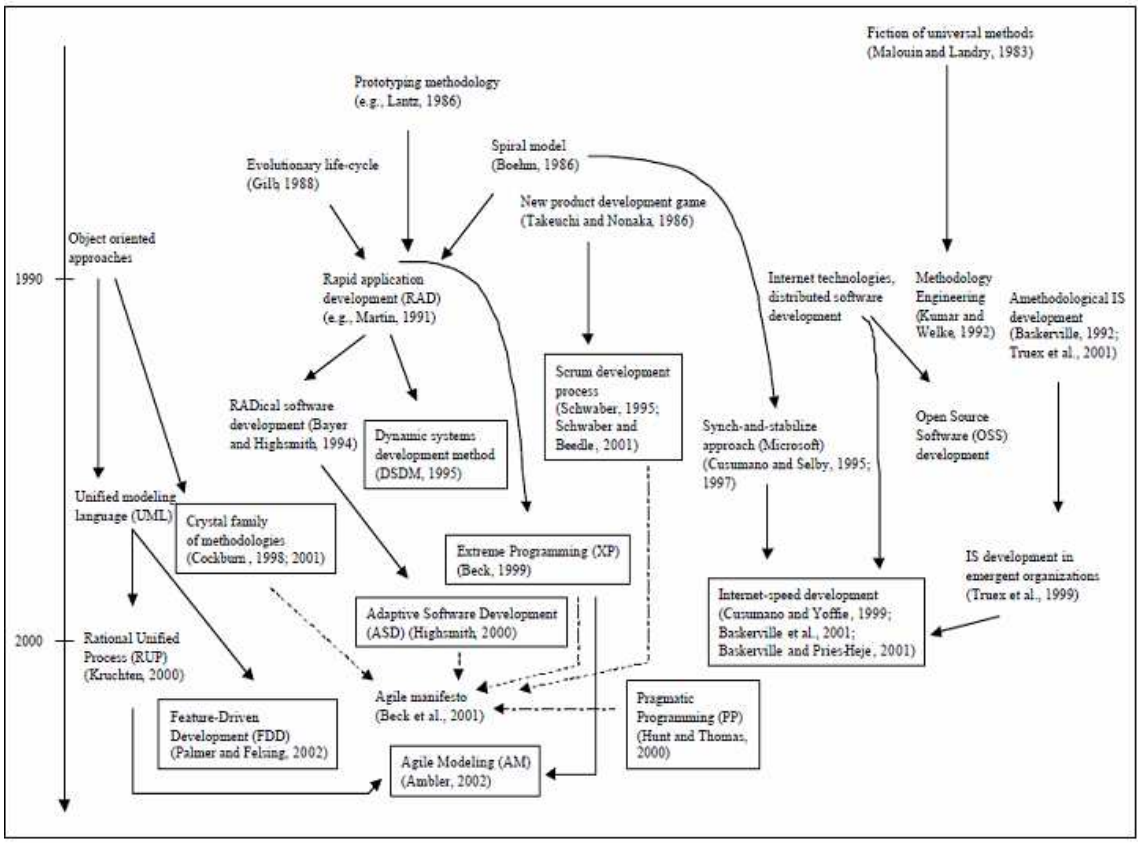
- ajankäytön jaksottaminen (engl. timeboxing),
- kehittyvä iteratiivinen kehitysprosessi,
- toimivan valmiin osatuotteen toimittaminen ja
- muita käytäntöjä, jotka tukevat nopeaa toimittamista ja varautumista jatkuvaan muutokseen.

Ketterien menetelmien periaatteen mukaan henkilöt sekä kommunikaatio ovat hyvin tärkeitä, prosessit ja välineet ovat toissijaisia [Lar07]. Tiimin yhteishenkeä tuetaan monin tavoin: työtilat ovat lähekkäin ja tiimi kokoontuu usein [AbS04]. Pyrkimys on toimivan ohjelmiston jatkuvaan ja nopeaan julkaisemiseen kaikenkattavan

dokumentaation sijaan. Ohjelmistokehittäjiä rohkaistaan kirjoittamaan yksinkertaista ja suoraviivaista ohjelmakoodia sekä välttämään hankalia teknisiä rakenteita kuitenkin käyttäen korkealaatuista tekniikkaa [AbS04]. Asiakas on nostettu tärkeään asemaan; yhteistyö ohjelmistokehittäjien ja asiakkaan välillä on tärkeämpää kuin muodolliset sopimusneuvottelut. Muutosta pidetään hyvänä asiana ja muutoksiin reagoidaan nopeasti. Muutokseen on varauduttu etukäteen esimerkiksi sopimuksissa, ja ohjelmistokehittäjillä on valtaa ja mahdollisuuksia tehdä muutoksia jopa prosessin aikana [Lar07, AbS04].

Asiakkaan muuttuvat vaatimukset tai kokonaan uusien vaatimusten lisääminen eivät ketterässä ohjelmistokehityksessä ole yhtä suuria ongelmia kuin vesiputousmallissa [AlM09]. Kun ketterässä mallissa asioita tehdään lyhyissä sykleissä, voidaan muuttuvia vaatimuksia käsitellä helpommin. Kaikki vaatimukset kirjataan ylös ja jokaisen syklin alussa ne priorisoidaan uudelleen, jotta löydetään kussakin syklissä toteutettavat asiat [Lar07].

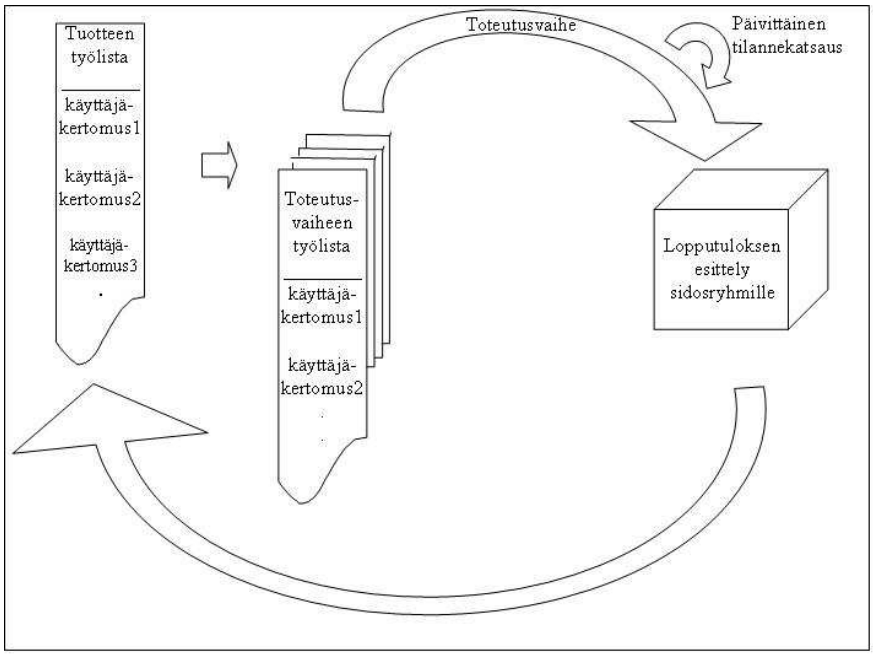
Abrahamsson ja kumppanit [AbW03] kokosivat vuonna 2003 yhteen olemassa olevat ketterät ohjelmistokehityksen menetelmät. Kuvassa 4, joka on suora lainaus heidän artikkelistaan, esitetään ketterät menetelmät aikajärjestyksessä sekä niiden väliset suhteet. Menetelmiä on yhdeksän: Scrum, Dynamic systems development method, Crystal family of methodologies, Extreme Programming, Internet-speed development, Adaptive Software Development, Pragmatic Programming, Feature-Driven Development ja Agile Modeling. Ensimmäiset ovat 80-luvun alusta ja viimeisimmät vuodelta 2002. Kuva 4 sisältää myös menetelmien kehitykseen vaikuttaneita taustatekijöitä ja metodologioita, mutta nämä ovat tämän tutkielman rajauksen ulkopuolella.



Kuva 4: Ketterien menetelmien evoluutio ja taustoja Abrahamssonin ja kumppanien [AbW03] artikkelista. Julkaistaan tekijän luvalla.

Tässä tutkielmassa perehdytään tarkemmin Scrum sekä Extreme Programming -menetelmiin. Nämä ovat yleisesti käytössä olevia menetelmiä [Lar07]. Scrum korostaa itseohjautuvien tiimien merkitystä, päivittäistä tiimin toiminnan mittaamista ja pyrkimystä joustavaan työskentelyyn kaavan mukaan etenemisen sijaan. Extreme Programming eli XP puolestaan painottaa yhteistyön merkitystä, nopeaa ohjelmiston toimitusta asiakkaalle ja hyvien ohjelmistokehityksen menettelytapojen käyttämistä. [Lar07].

Kuvassa 5 nähdään Scrum-menetelmän prosessi. Tämän Scrumin prosessia kuvaavan kappaleen päälähdeteoksena on käytetty Schwaberin ja Beedlen [ScB02] kirjaa Agile Software Development with Scrum.



Kuva 5: Scrum-prosessi Schwaberin ja Beedlen [ScB02] mukaan.

Scrum-menetelmässä ensimmäisessä vaiheessa kartoitetaan kokonaisuutta: kerätään käyttäjäkertomuksilla tai muilla tavoilla tietoa asiakkaan vaatimuksista, tietoa liiketoimintaympäristöstä sekä tehdään toteutettavan järjestelmäkokonaisuuden aikataulu- sekä kustannusarvio. Käyttäjäkertomukset ovat asiakkaan tekemiä kuvauksia tilaamansa ohjelmiston halutuista vaatimuksista. Niissä vaatimukset kuvataan sanallisesti; käyttäjäkertomukset voivat ilmaista järjestelmän piirteitä (engl. feature) sekä toiminnallisa ja ei-toiminnallisia vaatimuksia [Lar07].

Seuraavassa vaiheessa toteutettaviksi valituista käyttäjäkertomuksista muodostetaan toteutusvaiheen suunnttelupalaverissa toteutusvaiheen työlista yhteistyössä asiakkaan edustajan kanssa. Tämän jälkeen käyttäjäkertomukset määritellään, suunnitellaan, toteutetaan ja testataan järjestelmään. Kunkin toteutusvaiheen lopputuloksena on toimiva tuote. Prosessia iteroidaan, kunnes tuote on valmis.

Kukin toteutusvaihe kestää yleensä neljä viikkoa, ja sen aikana tehdään toteutusvaiheen työlistalla olevat asiat. Mitään muita asioita ei oteta tehtäväksi toteutusvaiheen kuluessa. Toteutus tehdään yhdessä tai useammassa tiimissä. Tiimit ovat itseohjautuvia eli niiden jäsenillä on päätösvalta töidensä suunnittelussa ja toteutuksessa. Tiimeissä vallitsee jaettu johtajuus, jolloin niissä ei ole erikseen valittua johtajaa. Tiimeissä on jäseninä eri

alojen huippuasiantuntijoita, jotka vastaavat oman alansa asioista prosessissa. Scrum-mestari (engl. Scrum master) valvoo tiimin toimintaa ja pyrkii tarjoamaan sille parhaat mahdolliset työskentelyolosuhteet. Toteutusvaiheen aikana pidetään päivittäin 15 minuutin tilannekatsaus, jossa esitetään tietyt Scrum-menetelmän mukaiset kysymykset:

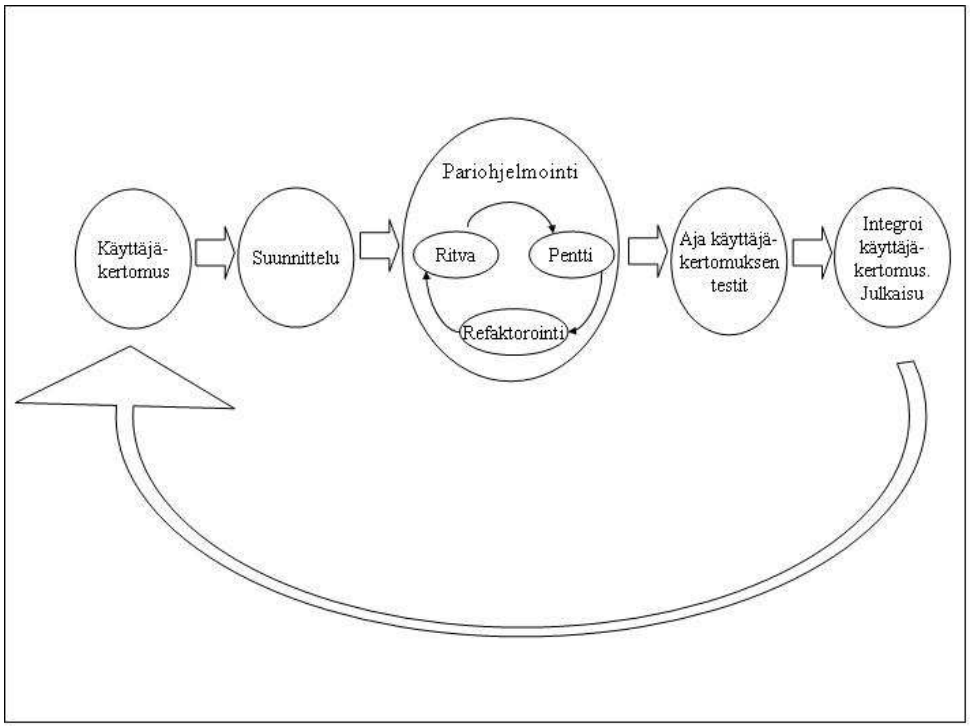
- mitä olet tehnyt viimeisimmän tapaamisen jälkeen,
- mitä aiot tehdä nyt, ennen seuraavaa tapaamista ja
- onko ilmennyt joitakin ongelmia, asioita, jotka ovat esteenä työn suorittamiselle.

Toteutusvaiheen jälkeen pidetään yhdessä asiakkaan ja muiden sidosryhmien kanssa toteutusvaiheen päätöspalaveri. Siinä esitellään vaiheen lopputulos, toimiva tuote, sidosryhmille. Mahdolliset korjaukset tuotteeseen viedään tuotteen työlistalle, joka elää ohjelmistokehitysprosessin aikana: tuotteen työlistalle voidaan viedä myös uusia asiakkaan vaatimuksia tarpeen mukaan. Kun tuotteen työlista on jälleen vakiintunut, aloitetaan seuraavan toteutusvaiheen suunnittelu suunnittelupalaverissa, ja prosessia jatketaan iteroiden kunnes tuote on valmis.

Beckin [Bec99b] kehittämä Extreme Programming on toinen yleisesti tunnettu ketterä ohjelmistokehitysmenetelmä [Lar07]. Seuraavat XP:tä kuvaavat kappaleet perustuvat Beckin [Bec99b] kirjaan ellei lähdettä ole erikseen ilmoitettu.

Kuvassa 6 esitetään XP:n prosessi, joka koostuu neljästä vaiheesta:

- käyttäjä kirjoittaa käyttäjäkertomukset ensimmäiseen julkaisuun,
- käyttäjäkertomusten priorisointi ja aikataulutus,
- iteraatiovaihe: määrittely, suunnittelu ja testaus pariohjelmointina useina pieninä iteraatioina,
- testaus ja järjestelmän suorituskyvyn tarkastaminen ennen julkaisua ja
- viimeinen vaihe, jossa järjestelmä täyttää asiakkaan tarpeen eikä käyttäjäkertomuksia enää synny.



Kuva 6: Extreme Programming –prosessi Beckin [Bec99b] mukaan.

Aluksi asiakas kirjoittaa käyttäjäkertomukset, jotka kuvaavat tuotettavan järjestelmän piirteitä ja toiminnallisuutta. Seuraavaksi asiakas priorisoi käyttäjäkertomukset. Tämän jälkeen asiakas ja XP-tiimi suunnittelevat yhdessä seuraavaan julkaisuun valittavat käyttäjäkertomukset. XP:n kokonaisprosessin aikana asiakas voi muuttaa käyttäjäkertomuksia tai poistaa niitä kokonaan listalta. Kun seuraavaan julkaisuun on valittu käyttäjäkertomukset, ne suunnitellaan toteuttaen yksinkertaisuuden periaatetta (engl. keep it simple). Yksinkertaisempi on aina parempi. Pariohjelmoinnissa työnkulku etenee suunnittelusta yksikkötestien kirjoittamiseen, ohjelmointiin, yksikkötestien suorittamiseen ja suunnittelun validointiin. XP:ssä siis ehdotetaan yksikkötestien kirjoittamista ennen ohjelmoinnin aloittamista. Refaktoroinnissa järjestelmää pyritään parantamaan siten, että toteutetun osakokonaisuuden käyttäytyminen ei muutu, mutta rakenne kohentuu. Pariohjelmoinnin jälkeen ajetaan käyttäjäkertomuksen hyväksymistestit, joissa kokonaisuuden toiminta varmistetaan, ja liitetään käyttäjäkertomukset osaksi ohjelmistoa.

XP:ssä nähdään, että kommunikaatio-ongelmat aiheuttavat suuren osan ohjelmistoprojektien vaikeuksista. Tämän vuoksi kommunikaatioon rohkaistaan

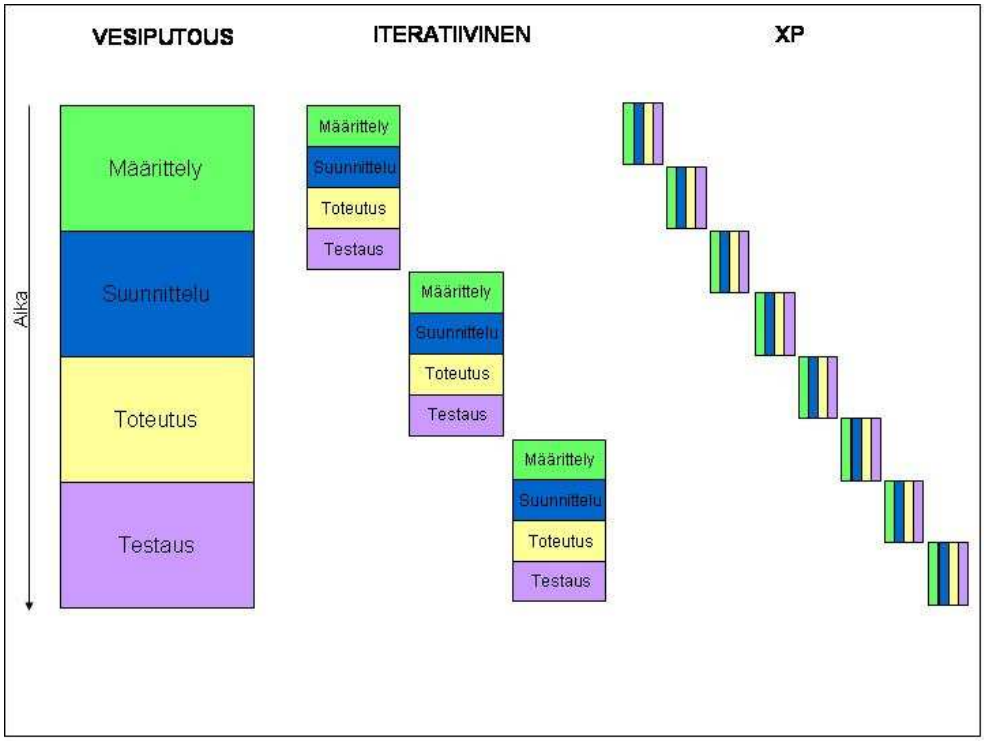
muunmuassa pariohjelmoinnilla sekä päivittäisillä tilannekatsauksilla. Lähtökohtana olisi asiakkaan vahva läsnäolo sekä kommunikointi asiakkaan kanssa läpi koko prosessin. Korkalan ja Abrahamssonin [KoA04] mukaan asiakkaan jatkuva läsnäolo kuitenkin harvoin toteutuu käytännössä. He väittävät, että asiakkaan läsnäolon puute on ongelma useimmissa ketterissä projekteissa, vaikka ketterien menetelmien perusperiaatteisiin kuuluu jatkuva kommunikointi asiakkaan kanssa.

XP:ssä pyritään tekemään niin mutkaton ohjelmisto, kuin mahdollista. Tehdään ainoastaan asiakkaan vaatimat asiat, ei ylimääräisiä piirteitä ohjelmistoon. Palaute lisää XP:n mukaan ohjelmiston laatua. Testauslähtöinen ohjelmistokehitys lisää aikaisen palautteen määrää, jolloin virheet löydetään aikaisemmin. Testausta tehdään usein, automatisoidusti ja mahdollisimman aikaisessa vaiheessa. Päivittäisissä palavereissa, joissa asiakaskin on mukana, on mahdollisuus antaa palautetta säännöllisesti. Toimiva ohjelmisto kehitetään mahdollisimman nopeasti. Tällöin muuttuviin tilanteisiin vastaaminen on helpompaa.

Abrahamsson ja kumppanit [AbW03] paljastavat, että ketterät menetelmät tarjoavat työkaluja ohjelmistokehitykseen, mutta suurin osa niistä ei juuri anna tukea organisaation johtajille. Heidän mukaansa ketterät menetelmät ovat lähinnä menettelytapoja, joissa annetaan suuntaviivoja ohjelmistokehittäjille. Johdon näkökulma, eksplisiittisesti kuvattuna, olisi kuitenkin hyvin tärkeä osa menetelmää. Ketterissä menetelmissä on perusperiaatteita kuten nopeiden iteraatioiden käyttö sekä osajärjestelmien julkaisut, joista voi olla enemmän haittaa kuin hyötyä, mikäli menetelmällä ei ole hallinnointia koskevia ohjeita. Menetelmät ovat myös erilaisia koskien esimerkiksi iteraatioiden pituutta tai päivittäisiä toimenpiteitä, kuten tiimipalavereja. Tämä voi aiheuttaa sekaannusta. Abrahamsson ja kumppanit [AbW03] väittävät, että projektinhallinnan näkökohdat ovat puuttellisia kaikissa olemassa olevissa ketterissä menetelmissä.

Kuvassa 7 esitetään yksinkertaistettuna edellä kuvatut ohjelmakehityksen prosessimallit Beck:n [Bec99a] mukaan. Vesiputousmallin, iteratiivisen ohjelmistokehityksen prosessimallin ja ketterän ohjelmistokehityksen prosessimallin, tässä XP:n, vertailussa nähdään, kuinka nämä prosessimallit eroavat toisistaan. Vesiputousmallissa ja iteratiivisessa prosessimallissa prosessin vaiheet: määrittely, suunnittelu, toteutus ja

testaus seuraavat toisiaan ajallisesti lineaarisessa järjestyksessä. Vesiputousmallissa tällaisia vaiheiden joukkoja on koko prosessissa vain yksi, iteratiivisessa prosessimallissa niitä on useita, mutta edelleen eri vaiheiden järjestys on lineaarinen. XP:ssä perinteiset prosessin vaiheet pyritään sekoittamaan ja niitä tehdään vähän kerrallaan, osin päällekkäin. Tämä sopii paremmin ohjelmistokehitykseen, sillä se ei ole lineaarinen, vaan jatkuvasti muuttuva prosessi [Bec99a].



Kuva 7: Prosessimallien vertailu Beckin [Bec99a] mukaan.

2.3 Ohjelmistokehittäjän työn erityispiirteet

Erilaisissa ohjelmistokehityksen prosessimalleissa tuotetaan eri määrä **dokumentaatiota**. Perinteinen vesiputousmalli synnyttää paljon dokumentaatiota; jokaisen ohjelmistokehitysprosessin vaiheen tuloksena on hyväksytettäviä dokumentteja [PoP03]. Ketterissä ohjelmistokehitysmalleissa rohkaistaan vähäisempään dokumentaatioon. Pyrkimys ei olekaan mahdollisimman runsaaseen ja kaikenkattavaan dokumentointiin vaan siihen, mitä kaikkea voisi jättää pois dokumenteista [AbS04]. Dokumentteja kuitenkin tuotetaan myös ketterissä malleissa: XP:ssä järjestelmän

valmistuttua viimeisessä vaiheessa on dokumentoinnin aika [AbS04]. Dokumentaationa voidaan pitää myös tehtävien asioiden luetteloa [ScB02].

Prosessimalli vaikuttaa huomattavasti **ongelmanratkaisun** periaatteisiin. Vesiputousmalli lähinnä rohkaisee kätkemään ongelmia. Kun edellisessä vaiheissa muodostuneita ongelmia tai virheitä tulee esiin, tehdään niille tilapäinen korjausratkaisu tai ne jopa jätetään kokonaan ottamatta huomioon [Som00]. Kun käytetään ketterää prosessimallia, ongelmat ja virheet tulevat nopeammin esiin ja niiden korjaukset viedään työlistalle ja tehdään priorisoinnin mukaisesti. Ohjelmistokehityksessä ongelmat sisältävät suuria epävarmuustekijöitä, riskejä sekä monimuotoisuutta, jolloin ongelmanratkaisun hajauttaminen on toimiva ratkaisu [Lad08]. Ohjelmistokehittäjät saattavat tuntea epäonnistuneensa työssään, mikäli ongelma tai virhe esiintyy [Sin89].

Ohjelmistokehitysprosessissa edistymisen **visualisoiminen** auttaa ohjelmistokehittäjää näkemään, missä ohjelmistokehityksessä ollaan menossa: mitä työtä on jo tehty ja kuinka paljon sitä on vielä jäljellä [PoP03]. Mikäli jokainen toiminto tuottaa selkeän, mitattavissa olevan tuloksen, voidaan prosessimallia pitää visuaalisena [Som00]. Vesiputousmallissa edistymistä voidaan seurata, sillä jo ennen kutakin vaihetta tiedetään, kuinka paljon työtä vaiheeseen sisältyy edellisen vaiheen tuottaman dokumentaation perusteella [Som00]. Ketterissä malleissa prosessin edistyminen ei ole nähtävissä oleva, sillä asiakkaan vaatimukset tarkentuvat jatkuvasti matkan varrella, eikä dokumentaatiota ole järkevää tehdä jokaisesta versiosta [Som00].

Ohjelmistokehitysprosessit ovat monimutkaisia ja vaativat inhimillistä harkintaa [Hum89]. **Ohjelmiston kokonaisuuden hallinta** on yksinkertaisempaa vesiputousmallissa kuin ketterissä malleissa, sillä kokonaisuus on tiedossa ensimmäisen vaiheen jälkeen, kun dokumentaatio on valmis [PoP03]. Tämän takia suuriin, yli 500 000 koodirivin järjestelmiin, Sommerville [Som00] suosittelee eri prosessimallien yhdistämistä esimerkiksi siten, että iteratiivisesti tehdään ensin pois heitettävä prototyyppi, jotta asiakas voi tarkistaa vaatimuksiaan. Sen jälkeen toteutetaan järjestelmän helppotajuisimmat osat vesiputousmallilla. Sellaiset osat, jotka vaativat runsaasti asiakkaan osallistumista, kuten käyttöliittymät, toteutetaan iteratiivisesti. Kokonaisuuden hallintaa on myös liiketoiminta-alueen ymmärrys [PoP03]. Tähän eivät kuitenkaan anna ohjeita sen enempää vesiputoismalli kuin ketterät mallitkaan.

Vesiputousmallissa ei juuri rohkaista **kommunikointiin**, vaan yhden vaiheen päätyttyä tuotetaan seuraavalle vaiheelle dokumentaatiot, jota toteuttavat henkilöt seuraavat [PoP03]. Eri vaihetta tekevät henkilöt eivät juuri kommunikoi keskenään, sillä seuraavasta vaiheesta ei palata edelliseen [PoP03]. Ketterissä malleissa sen sijaan rohkaistaan kommunikaatioon ja nähdään sen parantavan koko ohjelmistokokonaisuutta [Lar07]. Kommunikointi nähdään suorastaan kriittisenä asiana [AbS04]. Kuitenkin kommunikointi asiakkaan kanssa, ja etenkin se, ettei asiakas ole jatkuvasti paikalla, ovat ongelmana useimmissa ketterällä menetelmällä toteutetuissa ohjelmistoprojekteissa [KoA04]. Asiakkaan läsnäoloa vaadittaisiin julkaisutilanteen lisäksi myös muutosten hyväksymiseen sekä muutosten toteuttamisen suunnitteluun, korjausten priorisointiin ja seuraavaan iteraatioon sisällytettävien asioiden valintaan [KoA04].

On ratkaisevaa ohjelmistokehittäjän kannalta, kuinka hyvin prosessimallin voi **ymmärtää ja omaksua** [Som00]. Tarvitaanko paljon koulutusta, jotta työskentely on sujuvaa, vai onko prosessimalli intuitiivisesti ymmärrettävä. Saattaa olla, että organisaatiossa on käytössä lukuisia ohjelmistokehitysprosesseja, joita kukaan ei hallitse kunnolla. Perinteisesti ohjelmistokehityksessä on keskitytty prosessiin ja hyvä sekä kattava suunnittelu on ollut kaiken lähtökohtana [PoP03]. Tarkat prosessimäärittelyt ovat olleet tarpeen ja ihmisten sopeuduttava prosessiin [PoP03]. Ohjelmistokehittäjät ovat kuitenkin tuottaneet ohjelmistoja prosessimallista riippumatta, omalla tavallaan [McT05]. Ohjelmoijat ovat kirjoittaneet koodia, testanneet toimiiko se ja mikäli se ei vastaa tarkoitustaan, he ovat muuttaneet koodiaan kunnes se toimii [McT05]. On ollut valtava ero sen välillä, miten ohjelmistoa pitäisi prosessimallin mukaan tehdä, ja miten sitä todellisuudessa tehdään [McT05]. Ihmiset eivät ole omaksuneet prosessia. Ohjelmistokehitysprosessit kannattaa standardoida organisaatiossa, jotta niiden monimuotoisuutta olisi parempi hallita [Som00]. Tämä johtaa parempaan kommunikaatioon henkilöiden välillä ja pienempään koulutuksen tarpeeseen [Som00]. Ketterissä menetelmissä, kuten XP:ssä, ensimmäinen iteraatio on yleensä kaikkein tuottamattomin, se on lähinnä oppimisprosessi kaikille osapuolille, jos tiimille menetelmä on uusi [Abr03]. Tämän jälkeen tuottavuus nousee huomattavasti kaikilla osa-alueilla [Abr03].

Konteksti vaikuttaa aina siihen, miten prosessia toteutetaan [PoP03]. Liiketoiminta-

alueen tuntemus on tärkeää ja yhden liiketoiminta-alueen tietämys ei välttämättä päde muilla alueilla [PoP03]. Periaatteet ovat kuitenkin voimassa liiketoiminta-alueesta riippumatta [PoP03]. Ohjelmistokehitysmenetelmien abstrakteilla ja lukuisilla periaatteilla ei juuri ole hyötyä ohjelmistokehittäjien työn kannalta [AbW03]. Perusperiaatteiden pitäisi muodostaa vain menetelmän ydin ja niitä pitäisi täydentää konkreettisilla ohjeilla siitä, miten tietty työtehtävä tulisi hoitaa [AbW03]. Useat ketterät menetelmät eivät tarjoa konkreettisia ohjeita, ainoastaan abstrakteja periaatteita ohjelmistokehitystyöhön [AbW03]. XP tarjoaa konkreettisia ohjeita koko prosessin ajaksi, kun taas Scrum tarjoaa ohjeistusta ainoastaan vaatimus- ja määrittelyvaiheisiin, mutta ei ota kantaa toteutusvaiheeseen [AbW03]. Ohjelmistokehittäjät tarvitsisivatkin selkeitä ohjenuoria menetelmien toimivuudesta eri tilanteissa ja eri organisaatioissa abstraktien periaatteiden sijaan [AbW03].

Palautteen avulla ohjelmistokehittäjien on mahdollista kehittää itseään ja omaa työtään [PoP03]. Vesiputousmallissa palautetta ei kuitenkaan pidetä erityisen toivottavana, vaan palautteen katsotaan vaarantavan ennalta määrätyn suunnitelman toteutumisen [PoP03]. Palautetta on sallittua antaa vain kunkin vesiputousmallin vaiheen välissä [Som00]. Ketterissä malleissa palautetta pidetään hyvänä asiana ja sille pyritään antamaan mahdollisuuksia, esimerkiksi päivittäisissä tiimien kokoontumisissa tilannekatsaukseen [Lar07].

Asioiden **hyväksyttämisprosessi** on myös ohjelmistokehittäjän kannalta hyvin erilainen eri prosessimalleissa. Vesiputousmallissa jokaisen vaiheen jälkeen tuotettu dokumentaatio vaiheessa tehdyistä töistä hyväksytetään ennen seuraavaan vaiheeseen siirtymistä [Kom04]. Ketterissä malleissa hyväksyttämistä suoritetaan usein, esimerkiksi Extreme Programming –menetelmässä tehdään hyväksymistestaukset jokaisessa iteraatiossa [Bec00]. Extreme Programming –menetelmässä ja Scrumissa asiakas hyväksyy osaltaan valmiin osatuotteen julkaisutilaisuudessa, jossa lopputulos esitellään [Bec00, ScB02].

Työtehtävien valintaan vaikuttaminen riippuu prosessimallista. Määrää-ja-kontrolloi –johtamismenetelmä (engl command-and-control) ei ole tehokas, kun työntekijät ovat erittäin ammattitaitoisia ja liiketoiminta-alueen erityispiirteet tärkeitä lisäarvon tuottamiseksi [Lad08]. Ketterissä menetelmissä ohjelmistokehittäjillä on

Abrahamssonin ja kumppanien [AbS04] mukaan laaja-alainen tietämys sovellusalueesta, kun suunnitteluvoittoisissa menetelmissä, kuten vesiputousmallissa, ohjelmistokehittäjillä on työtehtäviään vastaavat tiedot ja tämän lisäksi mahdollisesti pääsy ulkopuoliseen tietoon, jos tarvetta ilmenee.

2.4 Yhteenveto

Ohjelmistokehittäjän työ on erilaista riippuen siitä, mitä ohjelmistokehitysprosessimallia käytetään. Kirjallisuudesta valittiin yhdeksän ohjelmistokehittäjän työn erityispiirrettä, joihin prosessimallilla on vaikutusta. Vaihtelua on kirjallisuuden perusteella dokumentaation tuottamisessa, myös ongelmanratkaisun periaatteet vaihtelevat prosessimallista riippuen. Prosessimalleissa on myös eroa siinä, kuinka visuaalinen malli on sekä siinä, kuinka hyvin ohjelmistokehittäjä voi nähdä kokonaisuuden, kuinka paljon mallin mukaan tapahtuu kommunikaatiota sekä kuinka ymmärrettävä prosessimalli on. Palautteeseen suhtaudutaan prosessimalleissa eri tavalla ja työtehtävien valintaprosessi on erilainen.

3 Kanban tuotantotaloudessa sekä ohjelmistokehityksessä

Edellisessä luvussa esiteltiin ohjelmistokehittäjän työn erityispiirteitä. Jotta voidaan tutkia, kuinka Kanban-menetelmä vaikuttaa ohjelmistokehittäjän työhön, perehdytään tässä luvussa tarkemmin Kanban-menetelmään.

Lean-filosofia on kokonaisuus, jonka osa Kanban-menetelmä on. Tämän vuoksi luvun alussa tutkitaan, mitä Lean tarkoittaa tuotantotaloudessa. Tämän jälkeen esitellään Lean-filosofia, jossa Lean-tuotannon periaatteet tuodaan tuotantoprosessista koko yritystä koskeviksi. Tämän jälkeen tutkitaan mitä Lean tarkoittaa ohjelmistokehitysprosessissa. Kanban-menetelmä esitellään samoin: ensin tuotantotalouden kannalta ja sitten ohjelmistokehityksessä. Lopuksi tehdään yhteenveto luvun pääkohdista.

3.1 Lean-tuotannon filosofia

3.1.1 Lean tuotantotaloudessa

Kajaste ja Liukko [KaL94] väittävät, että yritysten kilpailukyky riippuu merkittävästi niiden kyvystä tuottaa asiakkaille uusia tuotteita ja palveluja nopeasti, laadukkaasti, kustannustehokkaasti sekä siten, että erilaisia tuotevariaatioita voidaan tuottaa joustavasti. Menestyminen vaatii heidän mukaansa myös yrityksen henkilökunnalta kykyä kehittyä jatkuvasti ja keksiä uusia ratkaisuja työn toteuttamiseen.

Womack ja kumppanit [WoJ90] esittelevät kirjassaan länsimaalaisittain uuden ajattelutavan, jonka avulla yritys voi suoriutua paremmin markkinoilla. He kuvaavat, kuinka Lean-tuotantoa (engl. Lean Production) on käytetty autonvalmistuksessa Japanissa. Heidän mukaansa yritys voi parantaa kannattavuuttaan ja joustavuuttaan sekä vastata asiakkaiden tarpeisiin paremmin ottamalla käyttöön Lean-tuotannon. He väittävät, että keskittymällä vain asiakkaalle lisäarvoa tuottavaan toimintaan voidaan säästää merkittävästi kustannuksia ja aikaa. Womack ja Jones [WoJ94] osoittavat, että yritykset ovat jopa puolittaneet kehittämisestä, valmistamisesta ja jakelusta aiheutuvat kustannukset sekä myös niihin kuluvan ajan ottamalla käyttöön Lean-tuotannon.

Lean-tuotannon toimintatavat ovat [KaL94] :

- kaiken perustana on asiakkaalle tuleva arvo,
- huomio kiinnitetään kokonaisuuteen,
- kustannusrakennetta kevennetään jatkuvasti,
- tiedonkulku on suoraa ja avointa,
- omaa toimintaa kehitetään jatkuvasti,
- joustavat ja nopeat toimitusketjut ja
- henkilöresurssien järkevä yhdistäminen nykyaikaiseen tuotantotekniikkaan.

Asiakkaan tarpeita kartoitetaan systemaattisesti, ja nuo tarpeet ohjaavat koko yrityksen toimintaa. Jokaisella työntekijällä on oltava käsitys yrityksen asiakkaista ja heidän tarpeistaan. Asiakkaalle aikaansaatava arvo on toiminnan kehittämisen lähtökohta. Kysytään: ”Mitä asiakas haluaa tältä prosessilta?”. Vastaus määrittelee arvon (engl.

value). Arvo muodostuu usein laadusta, hinnasta ja ajasta, sekä myös toimitusvarmuudesta, kyvystä reagoida muutoksiin. Kaiken lähtökohtana ovat asiakkaan tarpeet, jotka vetävät tuotantoa.

Tavoitehinnoittelun (engl. target pricing) periaatteen mukaisesti yritys kartoittaa ensin asiakassegmentit ja määrittelee mitkä niistä ovat yrityksen kannalta kiinnostavia. Sitten määritetään tuotteelle hinta ja kartoitetaan mitä toiminnallisuutta kyseinen asiakassegmentti haluaa tuotteelta sillä hinnalla sekä arvioidaan kuinka paljon tuotetta tulee menemään kaupaksi. Tämän jälkeen tuotantoprosessi suunnitellaan näiden mukaan siten, että halutulla hinnalla saadaan toteutettua määritelty toiminnallisuus sekä tuotantomäärä. [WoJ94, CoC96]. Tuotannon nopeus ja määrä suhteutetaan asiakkaan kysynnän mukaan [Swa03]. Tällöin tuotteita tehdään vain silloin, kun niitä tarvitaan, siellä missä niitä tarvitaan sekä siinä määrin kuin tarvitaan.

Shinkle [Shi09] esittelee toisen näkökulman, jossa tuotannon määrä määritellään asiakkaan ja yrityksen välisessä sopimuksessa sen sijaan, että se suunniteltaisiin etukäteen halutun hinnan pohjalta. Myös tässä mallissa asiakkaan tarve kuitenkin siis määrää tuotannon määrän.

Koko yrityksen toimintaa tarkastellaan asiakkaalle lisäarvoa tuottavina prosesseina, joista löydetään työnkulkuja (engl. flow) asiakkaan tarpeesta toimitukseen asti. Huomio kiinnitetään kokonaisuuteen, jossa jokainen ketjun osa vastaa toiminnastaan seuraavalle vaiheelle. Työnkulku pyritään pitämään tasaisena ja sellaisena, että tehdään vain arvoa tuottavia asioita, lyhyissä intervalleissa. Jätettä on kaikki se, joka ei tuota lisäarvoa. Näin jätettä ovat esimerkiksi ylituotanto, odottaminen, turhat tai liialliset tavaroiden kuljetukset, ylimääräinen tai virheellinen tuotteen osien prosessointi, ylimääräinen varasto, tarpeeton liike sekä virheet.

Ylituotanto on perustavaa laatua oleva jäte, sillä se aiheuttaa useimmat muut [Lik04]. Tuottamalla missä tahansa prosessin vaiheessa enemmän kuin asiakas haluaa johtaa ylimääräisen varaston kertymiseen. Tämä puolestaan aiheuttaa sen, että jatkuvaan parantamiseen ei pyritä riittävästi. Koneita ei tarvitse huoltaa ennaltaehkäisevästi eikä laatuvirheistä välittää, kun viallinen osa voidaan vain heittää pois ja ottaa varastosta tilalle uusi. Mikäli varastoa pääsee kertymään, on viallisia osia kuitenkin voitu valmistaa varastoon jo useita ennen kuin virhe huomataan.

Liker [Lik04] ehdottaa kirjassaan vielä yhtä jätteen muotoa, nimittäin sitä, että työntekijöiden luovuutta ja innovatiivisuutta ei hyödynnetä riittävästi. Mikäli työntekijöitä rohkaistaan tuomaan esiin ideoitaan, voidaan Likerin mukaan voittaa aikaa ja löytää kehityskohteita työn suhteen.

Jäte pyritään poistamaan tuotantoprosessista, toimintoketjuista ja tuotteista. Perinteisesti johtamisen periaatteet ovat keskittyneet materiaalien ja tietojen käsittelyvaiheisiin, joita on pyritty tehostamaan esimerkiksi investoimalla tuotantokoneistoon. Muihin vaiheisiin ei ole juuri kiinnitetty huomiota. Saavutettu hyöty on kuitenkin osoittautunut pieneksi siihen verrattuna, että keskityttäisiin jätteen poistamiseen, erityisesti niihin vaiheisiin, jolloin materiaaleja ja tietoja ei käsitellä. Käsittelyajan lyhentäminen, jota saavutetaan jätettä poistamalla, johtaa parempaan laatuun ja pienempiin kustannuksiin sekä parantaa turvallisuutta ja työmoraalia.

Lean-tuotannon keskeisen periaatteen mukaan pyritään jatkuvasti kehittämään toimintaa. Jatkuva parantaminen vaatii koko henkilöstöltä pyrkimystä kehittää omaa työtänsä ja myös koko yrityksen prosesseja. Työtä tehdään tiimeissä, joissa kannustetaan aloitteellisuuteen työprosessien kehittämisessä. Tiimien jäseniä rohkaistaan ajattelemaan itse sekä kehittämään omaa työtään. Parantamisen on oltava säännönmukaista ja järjestelmällistä.

Lean-toiminnassa sekä tuotteet että työtehtävät pyritään standardoimaan. Standardoidun työn tekeminen on huomattavasti edullisempaa, mukavampaa ja helpompaa hallita. Standardoitu työ tarkoittaa myös työtaakan tasaamista, jolloin milloinkaan ei ole liikaa tai liian vähän töitä. Liker [Lik04] väittää, että kun työ on standardoitu, virheet tulevat esiin nopeammin. Tuotestandardoinnilla puolestaan yhdistetään asiakastarpeiden ja kustannustehokkuuden vaatimukset. Valtaosa asiakkaan tarpeesta pyritään täyttämään jo aikaisemmin valituilla ratkaisuilla. Kajaste ja Liukko [KaL94] esittävät, että tuotestandardointi vaatii perusteellista suunnittelua, mutta tuloksena saavutetaan mittavia parannuksia tuotteen laadussa, läpäisyajoissa, työmäärässä, ohjaustarpeessa ja varaston koossa.

Henkilöstö on Lean-toiminnassa nostettu avainasemaan. Ihmisiä pidetään yrityksen suurimpana voimavarana. Henkilöstöä kohdellaan hyvin, turvallisuus on tärkeätä, työergonomiaan kiinnitetään huomiota ja työpisteet pyritään saamaan viihtyisiksi.

Mielipiteiden vaihto on toivottavaa ja kanssakäymisessä pyritään avoimuuteen. Tiimien avulla koordinoidaan työtä, tiimin jäsenet motivoivat toisinaan ja oppivat toisiltaan. Dyer [Dye94] on tutkinut Toyotan ja Nissanin suhteita alihankkijoihin ja havainnut, että suora, henkilökohtainen kommunikointi on erittäin tuottavaa. Kommunikaatio on tärkeässä osassa Lean-toiminnassa, sillä rohkaisemalla henkilökohtaiseen keskusteluun sekä antamalla mahdollisuus ihmissuhteiden kehittymiselle pitkällä aikavälillä saadaan lisättyä tehokkuutta, nopeutettua tuotteen elinkaarta valmiiksi tuotteeksi, sekä luotettavampia tuotteita [PoP03].

Lean-tuotannon periaatteen mukaan paras käytännön asiantuntemus ja tietämys ongelmista on kullakin työpaikalla; ongelmat ratkaistaan tehokkaimmin niiden syntypaikalla. Kun ongelma tulee, ei pyritä etsimään syyllisiä, vaan varmistamaan etteivät virheet pääse toistumaan. Keskeistä on ongelmien havaitseminen mahdollisimman aikaisin ja virheiden korjaaminen nopeasti. Tuotteiden kulkeminen linjalla pyritään visualisoimaan, jotta ongelmat tulevat mahdollisimman nopeasti esiin. Tällaisesta sisäänrakennetusta laadusta Toyota Production System käyttää nimitystä jidoka. Laatu saavutetaan pysäyttämällä linjasto heti, kun ongelmia esiintyy, erottelemalla henkilöstön ja koneen tekemä työ, kontrolloimalla laatua jokaisessa työpisteessä ja ratkaisemalla jokainen ongelma perussyyhyn asti. Kun ongelma tulee, pyritään ensin selvittämään missä vika varsinaisesti on. Tämän jälkeen kysytään viisi kertaa ’Miksi?’ ongelman perussyyn löytämiseksi. Lopuksi korjataan ongelman pohjimmainen syy ja standardoidaan uusi toimintatapa. Pyritään pureutumaan virheiden perimmäiseen syyhyn ja sen jälkeen estämään virheiden syntyminen. Tavoitteena on nolla virhettä. Tietojärjestelmät ovat vain apuneuvo; ne eivät korvaa henkilökohtaisia keskusteluja eivätkä ihmisen panosta työpaikalleen. [Lik04].

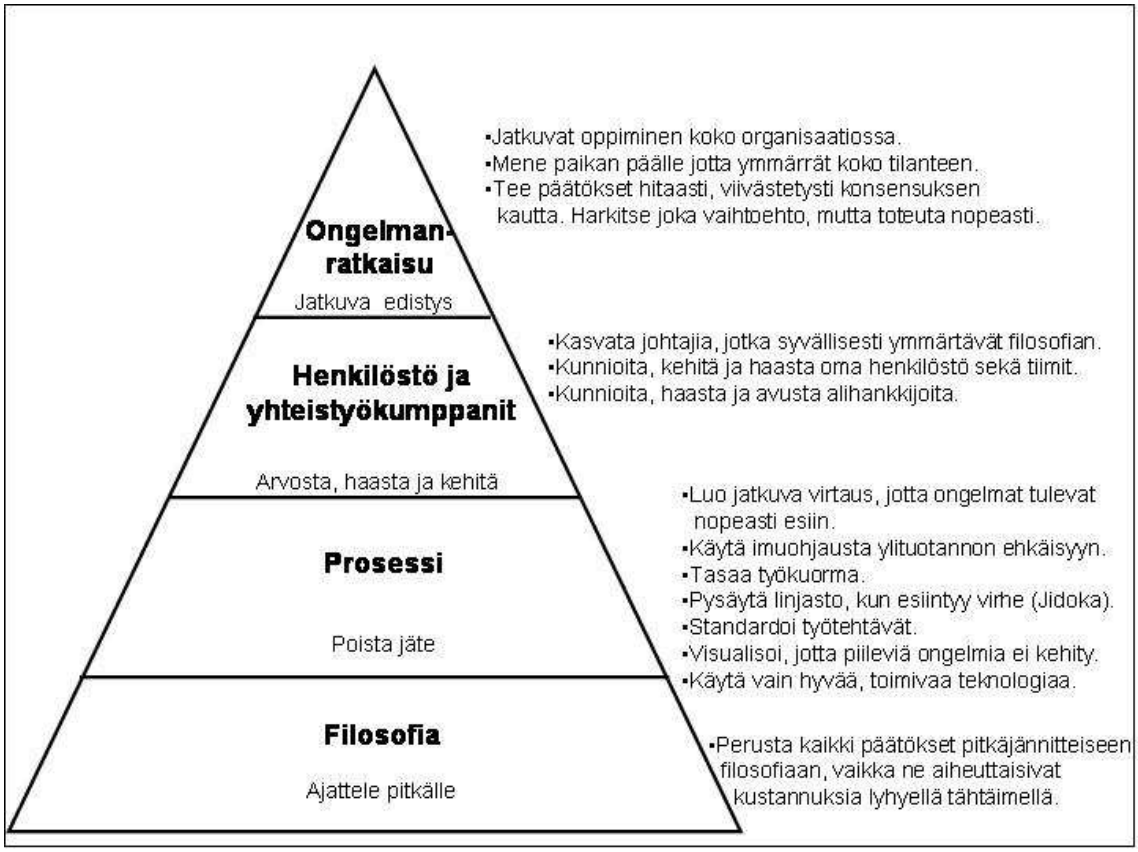
Henkilöstöä kannustetaan monitaitoisuuteen. On etu, että kukin osaa oman työnsä ohella muitakin tehtäviä. Näin varmistetaan toiminnan joustavuus poissaolojen tai henkilöstövaihdosten aikana. Pulmatilanteissa tai kuormitushuipun tilanteessa voidaan tilapäisesti lisätä henkilöitä pullonkaulakohtiin. Tällä tavalla luodaan pohjaa toiminnan yhteiselle kehittämiselle [KaL94]. Henkilöstöä myös koulutetaan jatkuvasti ja työntekijöille voidaan laatia henkilökohtaiset koulutussuunnitelmat.

Perinteisessä valmistavassa tuotantotaloudessa ei juurikaan kiinnitetä huomiota

joustavuuteen tai siihen, että asiakkaalla olisi valinnanvaraa. Valmistetaan suuria määriä samaa tuotetta ja keskitytään yksittäisten prosessien tehokkuuteen. Nämä prosessit, joita pyritään parantamaan, ovat lähinnä arvoa tuottavia prosesseja. Tehokkuutta saadaan aikaan esimerkiksi korvaamalla työntekijä koneella. Tuloksena saattaakin olla tuon yksittäisen prosessin huomattava parannus, mutta kokonaisuuteen toimilla on harvoin suurta merkitystä. Valmistavassa tuotantotaloudessa arvoa tuottavia prosesseja on yleensä suhteellisen vähän, joten niiden tehostaminen ei vielä paranna kokonaisuutta. Lean-tuotannon mukaisella jätteen poistamisella sen sijaan voidaan saavuttaa valtavia etuja, kuten tehokkaampaa ja joustavampaa toimintaa.

Sittemmin Womack ja Jones [WoJ94] esittävät, että pelkkä tuotantoprosessin Lean-tuotannon mukainen malli ei riitä, vaan myös koko yrityksen on sisäistettävä samanlainen ajattelu sekä toimintatavat johtoa myöten. Tuotteen elinkaaren prosessin (tilaus, vaatimukset, suunnittelu, toteutus, ylläpito) lisäksi yrityksessä on myös muita elementtejä, kuten infrastruktuuri (mm. tietotekniikka, henkilöstö, taloushallinto) sekä yrityksen johtamisprosessit (mm. strategian ja liiketoimintamallin määrittely, yhteistyökumppaneiden valinta), joiden tulee kaikkien toimia yhteistyössä sekä Lean-filosofian mukaisesti, jotta asiakkaan tarpeisiin pystytään vastaamaan nopeasti ja joustavasti [NiR04] [Mid01]. Lean tarjoaa siis filosofian koko yrityksen toiminnalle, antaa sille perustan.

Liker [Lik04] väittää, että Toyotan menestyksen takana ei ole ainoastaan Lean-tuotannon ottaminen käyttöön. Koko organisaation ja organisaatiokulttuurin on syvästi ymmärrettävä ja noudatettava Lean-tuotannon periaatteita noudattavaa Lean-filosofiaa. Jotta Lean-tuotanto saadaan kannattamaan ja Lean-filosofian noudattaminen onnistumaan koko yrityksessä, on työkaluja ja tekniikoita tärkeämpää, että yrityksen johto sitoutuu jatkuvasti ja pitkällä tähtäimellä investoimaan henkilöstöönsä sekä ylläpitämään yrityksessä jatkuvan parantamisen ilmapiiriä. Liker [Lik04] väittää, että on hyvin yleistä, että yrityksen johto haluaa ottaa käyttöön Lean-tuotannon tekniikoita, kuten Just-in-Time -periaatteen. Keskitytään vain prosessien parantamiseen ja jätteen eliminoimiseen. Jatkuvan parantamisen kulttuuri ja henkilöstön aito arvostaminen on kuitenkin olennainen osa menestymisen perustana olevaa pitkän tähtäimen Lean-filosofiaa.



Kuva 8: Lean-kokonaisuus Likerin [Lik04, s.13] mukaan.

Kuvassa 8 nähdään, kuinka Lean on kokonaisuus, joka nojaa pitkän tähtäimen Lean-filosofiaan [Lik04]. Kuvan 8 mukaisesti Lean-kokonaisuus muodostuu neljästä osa-alueesta: Lean-filosofia, prosessin kehittäminen ja jätteen poistaminen, ihmisten ottaminen huomioon tärkeänä osana organisaatiota ja jatkuva pyrkimys parempaan. Liker [Lik04] väittää, että usein, mikäli organisaatiossa päätetään ottaa käyttöön Lean-filosofian mukaisia toimia, keskitytään ainoastaan toiseksi alimpaan osioon, jätteen poistamiseen erilaisilla menetelmillä, esimerkiksi muuttamalla tuotanto toimimaan Just-in-Time -periaattella. Tällaiset toimet ovat kuitenkin pinnallisia eivätkä tuota hyvää lopputulosta. Olennaista on, että Lean-filosofia muuttaa koko yrityksen kulttuuria, jokapäiväisissä tehtävissä pyritään jatkuvasti parantamaan toimintaa ja ennen muuta koko yritys sisäistää Lean-filosofian ylintä johtoa myöten. Lean-filosofian johtamisperiaatteiden mukaan [KaL94]:

- ihmiset yhteistyössä tekevät yrityksen kannattavaksi,

- tavoitteet ovat selkeät ja tiedossa kaikilla työntekijöillä,
- organisaatio on hajautettu ja tulosvastuullisuus delegoitu,
- henkilöstö on monitaitoinen ja yritteliäs, palautetta annetaan sekä saadaan ja kannattavuus on asiakkaan ja koko organisaation henkilöstön yhteinen etu.

Kuvassa 8 nähdään, kuinka Lean-filosofia on koko toiminnan perusta. Päätökset tehdään pitkän tähtäimen Lean-filosofian mukaisesti, ei esimerkiksi lyhyen tähtäimen voittojen maksimoimiseksi. Koko organisaatiota ohjataan yhteiseen päämäärään, joka on yksittäisiä voittoja suurempi. Työntekijät ymmärtävät osuutensa, oman panoksensa yrityksen historiassa ja jokainen henkilö pyrkii parantamaan toimintaa koko ajan. Arvon tuottaminen asiakkaalle sekä yhteiskunnalle on kaiken lähtökohta. Jokaista yrityksen toimintoa arvioidaan tätä vasten, tuottaako se arvoa asiakkaalle tai muulle taholle.

Kuvassa 8 esitetään myös, kuinka työn standardointi ja sitä kautta tehostaminen ulotetaan Lean-filosofiassa tuotantoprosessin ulkopuolelle koskemaan koko organisaatiota. Esimerkiksi Toyotalla koneenrakennus on niin pitkälle standardoitu, että insinööri voi mennä Toyotan tehtaaseen missä päin maailmaan tahansa ja nähdä kutakuinkin samanlaiset rakennusprosessit. Myös suunnitteluprosessit on standardoitu. Standardit eivät Lean-filosofian mukaan ole pysyviä, vaan toimintaa pyritään kehittämään koko ajan, ja mikäli parempi toimintamalli löytyy, standardia ollaan valmiita muuttamaan.

Lean-filosofian periaatteen mukaisesti päätökset tehdään mahdollisimman myöhäisessä vaiheessa pitäen useita mahdollisuuksia avoimina, jotta asiasta saadaan kerättyä niin paljon tietoa kuin mahdollista ennen päätöksen tekemistä. Perinteisesti suunnittelussa on painotettu lopullisia päätöksiä jo analysointivaiheessa ja ne ovat olleet pohjana myöhemmin tuleville aktiviteeteille. [PoP03].

Ladas [Lad08] esittää, että Lean-filosofiaa menestyksekkäästi toteuttavan yrityksen toiminta saattaa ulkopuolisesta tarkkailijasta vaikuttaa epäluonnollisen toimivalta. Ihmiset, prosessit ja teknologia ovat paikalla silloin, kun niitä tarvitaan ja tekevät juuri sen mitä tarvitaan, ei lainkaan ylimääräistä. Kaikki on järjestyksessä ja säännöllistä, mutta samalla joustavaa ja muutokseen nopeasti vastaavaa. Kaikilla henkilöillä on syvä

tietämys prosessin toiminnasta sekä seikoista, jotka siihen toimintaan vaikuttavat. Tietämys on ulotettu kaikkialle. Säännöt pohjautuvat perusperiaatteisiin, jotka kaikki ymmärtävät. Koko henkilöstö ymmärtää kaiken toiminnan kokonaisuuden kannalta. Myös Poppendieck ja Poppendieck [PoP03] väittävät, että Lean-filosofian menetys perustuu syvään ymmärrykseen siitä, mitkä asiat tuottavat arvoa, miksi nopea virtaus on tärkeätä sekä miten saadaan työntekijöiden parhaiten käyttämään osaamistaan.

3.1.2 Lean-filosofia ohjelmistokehityksessä

Raman [Ram98] sekä Middleton [Mid01] väittävät, että Lean-filosofian periaatteet sopivat myös ohjelmistotuotantoon. Raman [Ram98] esittää, että ohjelmistotuotanto on hyvin erilaista verrattuna esineiden tai tavaroiden tuottamiseen, mutta Lean-filosofiaa voidaan noudattaa minkä tahansa prosessin tai aktiviteetin suhteen, myös ohjelmistotuotantoprosessin. Liker [Lik04] väittää, että Lean-filosofian toteuttaminen ohjelmistotuotannossa saattaa olla hieman hankalampaa, kuin valmistavassa tuotantotaloudessa, mutta se onnistuu kyllä. Poppendieck ja Poppendieck [PoP03] esittävät, että ohjelmistokehitys voi oppia Lean-filosofiasta paljon valmistavalta tuotantotaloudelta. Ohjelmistoja vaaditaan yhä nopeammin, yhä räätälöidympinä ympäristöönsä, mutta silti laadukkaina [Kom04]. Middletonin [Mid01] mukaan Lean-filosofian mukaisella ohjelmistonkehityksellä voidaan saavuttaa nopeasti korkeaa laatua sekä parantaa tuottavuutta.

Poppendieck ja Poppendieck [PoP03] väittävät, että palaute on olennaista ohjelmistotuotantoprosessin parantamiseksi ja virtaviivaistamiseksi. Vesiputousmallissa palautteen antamisen mahdollisuuksia on suhteellisen vähän ja palautteen antamisen paikat on tarkasti määritelty. Perinteisessä vesiputousmallissa oletetaan, että prosessi etenee ennalta arvattavasti junan lailla. Mitä enemmän palautetta on mahdollista antaa, sitä enemmän ennalta määritelty suunnitelma tai projekti on vaarassa muuttua. Palaute kuitenkin voi ehkäistä suurta osaa ohjelmistotuotannon ongelmista.

Lean-filosofian mukaisesti jätettä on kaikki se, mikä ei tuota lisäarvoa asiakkaalle [Lik04]. Vesiputousmallin perusaskeleet ovat suunnittelu ja toteutus [Roy70], eivätkä muut vaiheet vaikuta yhtä suoraan asiakkaalle toimitettavaan lopputulokseen [PoP03]. Poppendieck ja Poppendieck [PoP03] väittävät, että ohjelmistokehitysprosessia

voitaisiinkin tehostaa etsimällä etenkin ohjelmistokehityksen muista vaiheista: spesifikaatiovaiheesta, verifioinnista ja validoinnista sekä käyttöönotto- ja ylläpitovaiheesta sellaisia toimintoja, jotka eivät tuota arvoa asiakkaalle ja poistamalla nuo toiminnot.

Ohjelmistokehityksessä jätettä voivat olla Poppendieckin ja Poppendieckin [PoP03] mukaan:

- osittain tehty työ,
- ylimääräiset prosessit,
- ylimääräiset ominaisuudet tuotteessa,
- vuorottelu tehtävien välillä,
- odottaminen,
- liikkuminen ja
- toimintovirheet.

Osittain tehty työ sitoo resursseja, sillä osittainkin tehdyn työn eteen on tehty investointeja ja käytetty työpanosta. Mitä, jos työ ei koskaan valmistukaan? Tämän takia keskeneräinen työ on aina liiketoiminnallinen riski. Ei myös voida tietää mitä virheitä tuotteessa on ennen kuin se on valmis tai testattavissa. Keskeneräinen osakokonaisuus ei ehkä sovikaan kokonaisuuteen; virheitä tai ongelmia kokonaisuuden toiminnassa saattaa tulla sen jälkeen kun siihen on integroitu uusi osa. Vasta, kun ohjelmisto on oikeasti käytössä tuotannossa, voidaan tietää vastaako se liiketoiminnan tarpeeseen.

Ylimääräiset prosessit saattavat olla esimerkiksi dokumentteja, jotka eivät ole todella tarpeellisia. Tulisi miettiä, keitä varten paperityöt tehdään. Jos vastausta ei löydy, saattaa olla, että dokumentit eivät ole hyödyllisiä. Joskus yrityksen toimintatavat vaativat dokumentteja, joilla on vain vähän arvoa asiakkaalle tai yritykselle itselleen. Tällaiset dokumentit tulisi pitää mahdollisimman lyhyinä ja samalla laadukkaina.

Tuotteeseen saatetaan haluta rakentaa ylimääräisiä ominaisuuksia joko asiakkaan vaatimuksesta tai toteuttavan organisaation aloitteesta. Kuitenkin pitäisi miettiä, ovatko uudet piirteet riskeerauksen arvoisia, niihin menee toteutus- ja testausaikaa ja lisäksi ne saattavat rikkoa jo toimivia ominaisuuksia tuotteessa. Nämäkin ohjelmiston osat on

ylläpidettävä myös myöhemmin, ne lisäävät ohjelmakoodin monimutkaisuutta, ja ne on testattava muun koodin ohella aina, kun ohjelmaa muutetaan.

Kun tehtävien välillä vuorotellaan, aikaa kuluu asioiden palauttamiseen mieleen, kokonaistyömäärä kasvaa ja työn laatu voi huonontua. Ihminen ei pysty tekemään montaa työtä yhtä aikaa. Kokonaisuutena projektissa tulisi minimoida odotteluaika. Esimerkiksi ylimääräistä odottelua on projektin aloittamispäätöksen hyväksyttämisen odottaminen yrityksen johdolla, jos johtoryhmä kokoontuu kerran kolmessa kuukaudessa ja koko projekti seisoo sen aikaa. Odottaminen viivästyttää valmiin tuotteen toimittamista asiakkaalle, ja valmis tuote on arvoa asiakkaalle. Jos liiketoiminta-alue muuttuu voimakkaasti ja nopeasti, voi odottamisella olla asiakkaalle kriittisiä seurauksia.

Ylimääräinen liike tarkoittaa paitsi ihmisten, myös asioiden liikettä. Mikäli ohjelmoijat eivät työskentele samassa tilassa, saattaa ylimääräistä aikaa kulua kysymyksiin ja kommentteihin. Jos puolestaan ollaan asiakkaan tiloissa toteuttamassa, saadaan vastaukset asiakkaalta nopeasti. Toisaalta myös tuotokset liikkuvat ohjelmistoprojektissa. Ohjelmoijien valmiiksi saama työ toimitetaan testaajille, jotka puolestaan toimittavat testitulokset takaisin ohjelmoijille. Jokaisessa toimituksessa häviää tietoa, sillä asiasta kertynyttä ymmärrystä ei ole yksinkertaista siirtää eteenpäin. Artefaktien siirtäminen tiimiltä toiselle on mitä suurimmassa määrin jätettä ohjelmistoprojektissa.

Toimintovirheet tulisi minimoida, sillä viikkojen kuluttua havaittu pienikin virhe ohjelmistossa on kallis. Testaamista suoritetaan jatkuvasti, jolloin virheet pyritään löytämään ja korjaamaan heti. Valmis osatuote integroidaan nopeasti kokonaisuuteen, jotta kokonaisuuden toiminta voidaan testata. Tuote julkaistaan tuotantoon niin pian kuin mahdollista, jotta tuotantoympäristössä tulevat virheet saadaan esiin.

Perinteisessä massatuotannossa saatettiin tehdä vain tarpeeksi hyvää, mutta Lean-filosofiaan liittyy jatkuvan parantamisen ja täydellisyyteen pyrkimisen tavoite [Mid01]. Prosesseja pyritään jatkuvasti parantamaan siten, että kaikki jäte tunnistetaan ja poistetaan. Virheet pyritään ehkäisemään joka ohjelmistonkehityksen vaiheessa [Ram98]. Työnkulkua voidaan koordinoida, mutta silti ottaen huomioon ihmiset yksilöinä sekä myös tiimin jäseninä. Ihmisille annetaan mahdollisuus olla luovia ja

kehittää työnkulun askeleita parempaan suuntaan [Ram98].

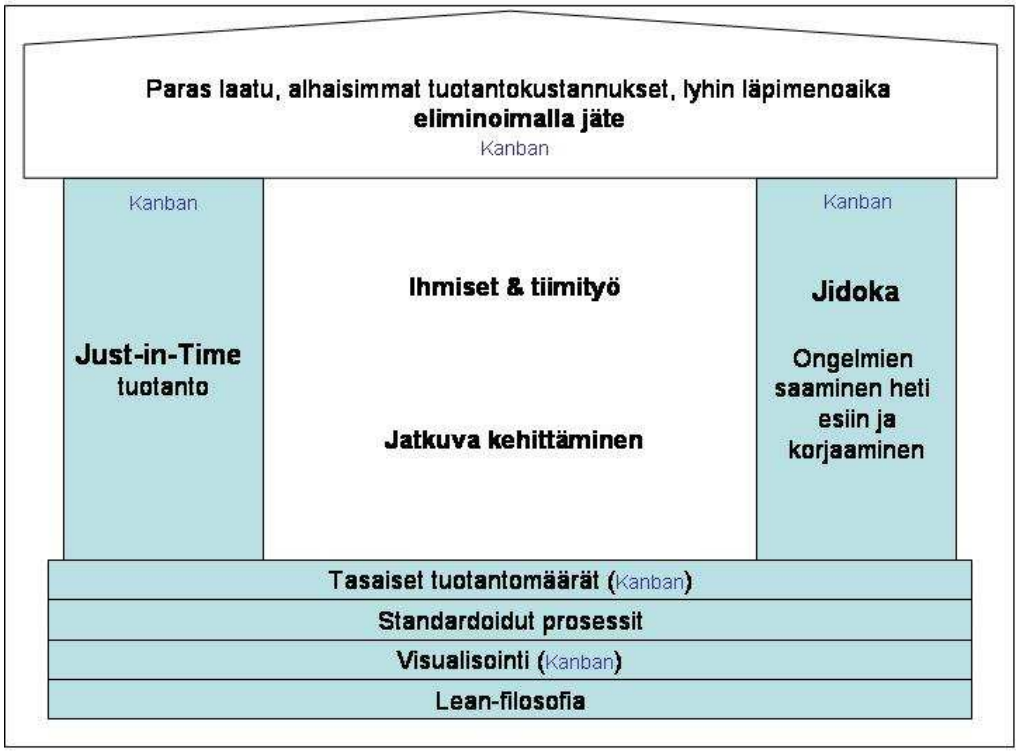
3.2 Kanban-menetelmä

3.2.1 Tuotantotalouden näkökulma Kanban-menetelmään

Kanban on alkujaan signalointisysteemi, jota käytetään imuohjaukseen perustuvassa Just-in-Time -tuotannossa. Sen mukaan tuotanto käynnistyy asiakaskysynnästä ja ikään kuin imee tuotteita läpi koko tehtaan tuotantoprosessin [Yam82, BeS98]. Kanban-menetelmä toteuttaa Lean-filosofiaa käytännössä [Cha08, BeS98]. Likerin [Lik04] mukaan Kanbanissa pyritään Lean-filosofiaa mukaillen:

- eliminoimaan jäte,
- alentamaan tuotantokustannuksia,
- jatkuvasti kontrolloimaan ja ylläpitämään laatua,
- muodostamaan työnkulusta jatkuva virtaus (engl. flow),
- rajoittamaan tuotantolinjalla olevan tavaran määrää,
- tekemään tuotteita vain, kun asiakas tarvitsee ja
- tunnustamaan sekä arvostamaan ihmisten asiantuntemusta.

Kuvassa 9 esitetään mitkä osat Lean-kokonaisuutta Kanban-tekniikka toteuttaa Likerin [Lik04] mukaan. Kanbanin avulla Toyotalla toteutetaan Just-in-Time –tuotanto, eliminoidaan jätettä, visualisoidaan työnkulut, tasataan tuotantokuormaa ja tuodaan ongelmat näkyville. Kuvassa Lean-kokonaisuus kuvataan talona Liker:n [Lik04] kirjan mukaan. Katossa ovat päämäärät: laatu, alhaiset kustannukset ja lyhyt läpimenoaika, joihin päästään jäte poistamalla. Seininä ovat Just-in-Time –tuotanto sekä ongelmien saaminen heti esiin ja korjaaminen. Kaiken perustana ovat Lean-filosofia, visualisointi, standardoidut prosessit sekä tasaiset tuotantomäärät. Talon keskiössä ovat ihmiset sekä jatkuva kehittäminen.



Kuva 9: Kanbanin sijoittuminen Lean-kokonaisuuteen Likerin [Lik04, s.33] mukaan.

Kanban on alun perin valmistavan tuotantotalouden menetelmä, jonka tarkoituksena on varmistaa nopea läpimenoaika autonvalmistuksen tuotantoprosessissa, tuotannon mukauttaminen vastaamaan kysyntää ja jatkuvan liikkeen, virtauksen, muodostaminen tuotantolinjalla ilman odotusaikoja osin rajoittamalla linjalla olevan tavaran määrää [Ett95] [Cha08]. Kanban-systeemiä kehitettäessä lähtökohtana oli supermarketeiden toiminta [Lik04]. Kanban -menetelmässä signaali ilmoittaa kysynnästä laukaistakseen tuotantoprosessin. Valmistusasema noutaa edelliseltä tarpeen mukaan sen sijaan, että edellinen asema toimittaisi ko. asemalle suunnitelman mukaan [Yam82].

Seuraava esimerkki ruokatavaroiden tilaamisesta on Likerin [Lik04] kirjasta. Oletetaan kaksi erilaista Internet-palvelua, joissa ruokatavaroita toimitetaan kotiin tilausten perusteella. Ensimmäisessä yrityksessä kertaluonteisen tilauksen yhteydessä ilmoitetaan kuinka paljon viikoittain tavaroita halutaan, ja tavaroita toimitetaan tuon tilauksen mukaan joka viikko. Toimitus luvataan joka viikko, mutta tarkkaa toimituspäivää ei ilmoiteta. Tilaaja ei voi olla varma tulevatko tavarat maanantaina vai perjantaina, joten jääkaapissa on oltava ruokatavaroita varmuuden vuoksi koko viikoksi. Mikäli toimitus

kuitenkin onkin jo maanantaina, on jääkaappi täynnä, ja täytyisi ostaa uusi kakkosjääkaappi, jonne saadaan tavarat varastoitua väliaikaisesti. Jos asiakas loman ajaksi unohtaa peruuttaa viikkotoimituksen, on lomalta palattua kuistilla odottamassa vanhenevia elintarvikkeita. Tämä on esimerkki perinteisestä työntöpohjaisesta systeemistä. Valmistavassa tuotantotaloudessa alihankkijat valmistavat tavaroita jälleenmyyjälle varastoon ilman, että kukaan tietää onko niille välitöntä tarvetta kysyntää.

Toisessa Internet-palvelussa asiakkaalle toimitetaan asiakaspääte, joka on langattomassa yhteydessä tavarantoimittajaan. Laitteessa on painike jokaiselle asiakkaan ilmoittamalle ruokatavaralle. Kun asiakas avaa maidon tai juustopaketin, hän painaa samalla kyseistä nappulaa. Seuraavana päivänä hänelle toimitetaan uusi pakkaus maitoa tai juustoa. Näin jääkaapissa on ainoastaan avattu paketti sekä uusi valmiina odottamassa. Jos kuitenkin asiakkaalla on tulossa juhlat ja hän tietää tarvitsevansa pian enemmän maitoa ja juustoa, hän voi lähettää lisätilauksen internetissä tai puhelimitse, ja tuotteet ovat seuraavana päivänä kuistilla odottamassa. Kun tavarantoimittaja saa tilauksen suuremmasta maitoerästä, se puolestaan ilmoittaa välittömästi meijeriin, joka lähettää toimittajalle suuremman maitomäärän seuraavana päivänä. Tämä on esimerkki imuohjauksesta. Tavaroita tulee vain silloin, kun niitä tarvitaan. Alihankkijat valmistavat tavaroita jälleenmyyjälle vain sen mukaan, mitä asiakas oikeasti tarvitsee. Imuohjauksen tuloksena on [Yam82]:

- oikea määrä oikeaa osaa oikeaan aikaan,
- työn alla olevissa tuotteissa on sidoksissa mahdollisimman vähän pääomaa ja
- valmistuksen yksinkertainen hallinto.

Alkujaan Kanban tarkoittaa fyysisesti korttia, johon on lisätty informaatiota tuotteesta. Kortit voidaan laittaa seinätaululle, johon tuotanto on jaettu useiksi askeleiksi, ja seuraamalla korttien liikkumista ja määrää seinätaululla voidaan seurata tuotteen liikkumista tuotantolinjalla. Liian vähäinen korttien määrä jossakin kohdassa tuotantolinjaa signaloi tarpeesta tuottaa sinne lisää [DuF95] [BeS98]. Mikäli jossakin kohtaa on puolestaan liikaa kortteja, on siellä mahdollinen pullonkaula, ja sinne on

lisättävä resursseja [Ett95]. Kanban voi myös olla fyysisen kortin sijaan elektroninen signaali, joka lähetetään haluttuun paikkaan lisäosien saamiseksi [McR89].

Perusajatuksena on se, että valmistus on organisoitava ja suunniteltava siten, että sitä on helppo valvoa ja ohjata [Yam82]. Tämä merkitsee, että valmistus on oltava visuaalisesti valvottavissa [Yam82]. Toyotalla tuotannossa käytössä on visuaalinen valvontalaite, joka varoittaa työntekijöitä ongelmista linjastolla [Lik04]. Esimerkiksi koneeseen saattaa olla kytketty valokatkaisija, joka ilmoittaa ongelmasta, tai koneesta saattaa kuulua tietty ääni, kun jotakin epänormaalia tapahtuu.

Ottamalla Kanban käyttöön pyritään pääsemään eroon perinteisten työntöpohjaisten systeemien (engl. push) ongelmista, kuten yllättävään muutokseen reagoinnin hankaluudesta ja suurista varastomääristä [BeS98]. Kanban-valmistuksen pyrkimyksenä on valmistuskustannusten alentaminen eliminoimalla kaikenlainen jäte. Jätteeksi käsitetään kaikki toiminta, joka ei lisää tuotteen jalostusarvoa [Yam82].

Kanban-menetelmässä prosessi pysähtyy automaattisesti, kun jotakin epänormaalia tapahtuu, esimerkiksi, jos kone rikkoutuu. Prosessin automaattisella pysähtymisellä saavutetaan etuja. Koneenhoitajan ei tarvitse keskittyä pelkästään yhteen prosessiin, vaan hän voi hoitaa useampia prosesseja rinnakkain. Lisäksi valmistusjärjestelmän ongelmat saadaan esiin ratkaistaviksi sen sijaan, että ne esimerkiksi peittyisivät taroittaviin materiaalin puskurivarastoihin. [Yam82]

Prosessissa on otettava huomioon mahdolliset huoltokatkokset ja koneiden rikkoutumiset siten, että pullonkaulaksi muodostuville prosesseille liitetään enemmän kapasiteettia, ja tuotantolinjoja yhdistämällä systeemi ei ole niin riippuvainen yksittäisistä koneista [Ett95]. Kun tuotantolinjoja yhdistetään, pitää myös määritellä, missä järjestyksessä tuotteita prosessoidaan [DuF95].

Pyrkimyksenä on myös tasoittaa valmistus vähentämällä maksimaalista kapasiteetintarvetta [Yam82]. Tuotantoprosessissa pyritään pitämään tasainen rytmi siten, että tavaraa on varastossa mahdollisimman vähän ja tavarat liikkuvat tasaisena virtana. Siirtymät toiminnosta toiseen pyritään hoitamaan nopeasti ja joustavasti [WiD02]. Tasainen rytmi voidaan saavuttaa rajoittamalla linjalla liikkuvien tavaroiden määrää siten, että mihinkään ei muodostu tukoksia. Toisaalta mikään linjaston osa ei saa joutua odottelemaan tulevia tavaroita.

3.2.2 Kanban-menetelmä ohjelmistokehityksessä

Shinkle [Shi09] väittää, että Kanban-menetelmän avulla Lean-filosofian periaatteet voidaan tehokkaasti tuoda ohjelmistokehitysprosessiin. Kanbanin avulla ohjelmistokehitystiimin on helpompi omaksua kulttuurinmuutos, joka Lean-filosofiaan liittyy. Lean-filosofian mukaisesti tehdään vain sellaista, mitä asiakas haluaa. Ei tehdä enempää ohjelmiston suunnittelua, kuin mitä pystytään ohjelmoimaan eikä enempää ohjelmakoodia, kuin mitä pystytään testaamaan. Ladaksen [Lad08] mukaan tämän hallinnoimiseen Kanban-menetelmä tuo ratkaisun [Lad08].

Poppendieck ja Poppendieck väittävät, että Kanbanin noudattama imuohjaus nopeuttaa ohjelmiston toimitusta. Kun asiakkaan tarpeet määrittelevät jokaisen työvaiheen ja kaikki lähtee asiakkaan tarpeesta, on jokaisella ohjelmistoprosessissa selvää mitä seuraavaksi vaaditaan. Seuraava vaihe ikään kuin vetää tuloksia edelliseltä vaiheelta [Lad08]. Kun valmistunut työ on vedetty yhdeltä vaiheelta, esimerkiksi suunnitelusta, eteenpäin ohjelmoitavaksi, aloittaa suunnitteluvaihe uuden työn tekemisen [Lad08].

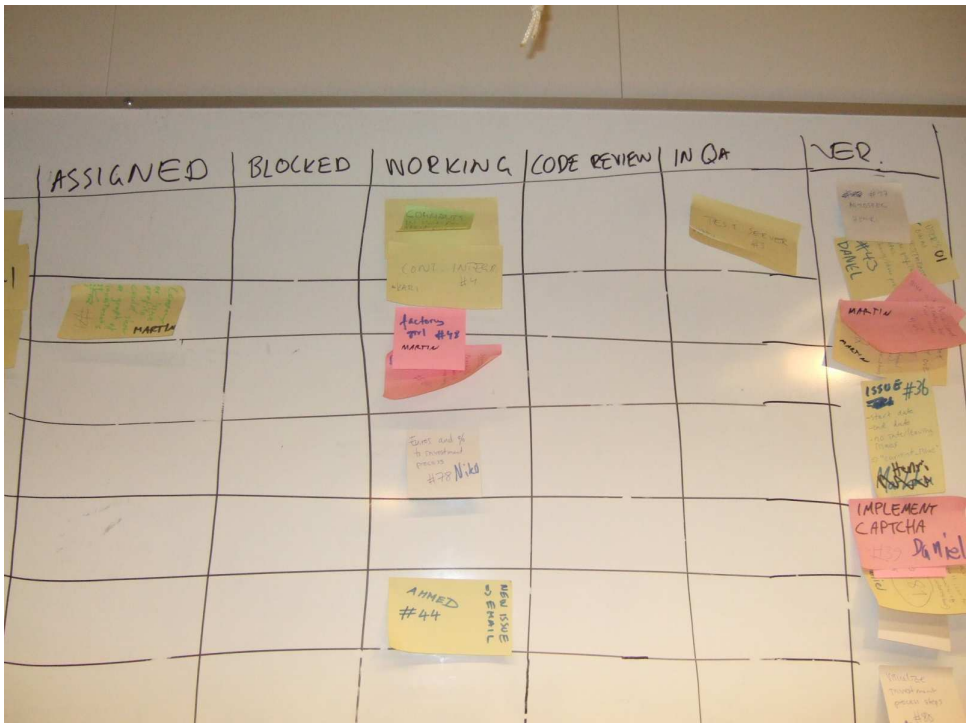
Nopeasti muuttuvassa toimintaympäristössä on olennaista, että tarvittava asiantuntemus on jokaisessa työpisteessä. Ei ole aikaa sille, että ylempi taho hyväksyy seuraavaksi tehtävät asiat ja ilmoittaa niistä eteenpäin suorittavalle tasolle. Kanban-menetelmässä Kanban-kortti kertoo edeltävälle ketjun vaiheelle, mitä siltä nyt kaivataan. Ilman tuota korttia ihmisten täytyisi jollakin muulla tavalla selvittää, mitä heiltä seuraavaksi vaaditaan. Poppendieck ja Poppendieck [PoP03] väittävät, että ennen Kanbania käännyttiin ylemmän tason henkilöiden puoleen sen selvittämiseksi. Kanbanin käyttöönoton jälkeen johtajien ei enää tarvitse kertoa työntekijöilleen kustakin seuraavasta työtehtävästä, vaan johtajien tehtäväksi jää tiimin auttaminen ja tukeminen.

Knibergin [Kni09] mukaan Kanbanissa on ohjelmistokehityksen kannalta kolme olennaista perusperiaatetta:

- visualisoi työnkulku,
- aseta rajat sille paljonko työtä saa yhtäaikaisesti olla tekeillä ja
- mittaa tehtävien läpimenoaikaa.

Kanbanissa Lean-filosofiaa mukaillen työnkulut jaetaan pieniin osiin, työvaiheisiin, ja

nämä visualisoidaan, dokumentoidaan esimerkiksi seinätaululle [Kni09]. Kuvassa 10 esitetään esimerkki työvaiheiden visualisoimisesta seinätaululle. Kuva on tämän tutkielman tapaustutkimuksesta, ensimmäiseltä projektityön viikolta. Asiakkaan vaatimuksiin pohjautuvat käyttäjäkertomukset voidaan dokumentoida esimerkiksi post-it -lapuille, jotka toimivat teollisuuden Kanban-korttien tapaan [PoP03]. Myös muita vaatimusten kuvaustapoja voidaan käyttää, esimerkiksi käyttötapauksia [Lad08]. Käyttötapaukset tai käyttäjäkertomukset pilkotaan osiin, pieniksi erikseen toteutettavissa oleviksi ominaisuuksiksi tai piirteiksi.



Kuva 10: Kanban-menetelmässä työ visualisoidaan seinätalululle. Kuva tutkielman tapaustutkimuksesta, projektin toiselta viikolta.

Kuvassa 10 nähdään, kuinka Kanban-laput sijoittuvat seinätaululle. Ladas [Lad08] ehdottaa esimerkin Kanbanin prosessiksi. Hänen esimerksissään kukin Kanban-lappu kuvaa yhtä käyttäjäkertomusta. Lean-filosofian periaatteen mukaisesti ohjelmistokehittäjillä on itsellään riittävä osaaminen arvioimaan kuinka paljon aikaa kunkin Kanban-lapun osoittaman käyttäjäkertomuksen suorittamiseen menee aikaa. Asiakas priorisoi käyttäjäkertomukset. Suunnittelukokouksessa valitaan seuraavassa

iteraatiossa toteutettavat ohjelmiston osat priorisoinnin mukaisesti. Päätös siitä, mitä asioita otetaan seuraavaksi toteutettavaksi, pyritään tekemään mahdollisimman viime hetkellä, jotta kaikki vaihtoehdot voidaan pitää avoimina niin pitkään kuin mahdollista ja siten valita paras ratkaisu [Kni09]. Näistä Kanban-lapuista muodostuu tehtävien asioiden alue (engl. To Do area), josta ohjelmistokehittäjät näkevät, mitä asioita on suoritettava seuraavaksi. Kun työtehtävä on tehty, Kanban-lappu siirretään testattavien asioiden alueelle seinätaululla. Kun ominaisuus on testattu, se siirretään valmiiden asioiden alueelle seinätaululla. Kanban-laput merkitsevät siis työtehtäviä, ei henkilöitä. Mikäli henkilö katsoo osaavansa myös seuraavan työvaiheen, hän voi lipua Kanban-lapun mukana seuraavaan työvaiheeseen ja suorittaa sen. Tämä vähentää töiden siirtämistä toiselle, mikä on yksi Lean-filosofian jätteen muoto.

Knibeg [Kni09] ehdottaa, että työtehtävien läpimenoa seinätaululla tekemättömien tehtävien listalta tehdyiksi töiksi seurataan järjestelmällisesti. Kanbanissa kiinnitetään erityistä huomiota tekeillä olevien asioiden rajoittamiseen sekä läpimenoaikaan, joita mitataan jatkuvasti [Lad08]. Mittaamalla läpimenoaikaa voidaan prosessia pyrkiä optimoimaan jotta läpimenoajasta tulisi mahdollisimman lyhyt ja samalla ennustettava [KnS10].

Ladas [Lad08] esittää, että tekeillä olevien asioiden maksimimäärä tulisi mukauttaa käytettävään kapasiteettiin. Jos käytettävissä on kolme ohjelmoijaa, ei tekeillä olevia asoita voi olla kovin paljon. Middleton väittää [Mid01], että tekeillä olevien asioiden lukumäärä tulisi minimoida laadun ylläpitämiseksi. Hän ehdottaa, että aina, kun yksi paperisivullinen suunnittelutyötä saadaan valmiiksi, työ luovutetaan eteenpäin toteuttajille. Näin virheet suunnittelussa löydetään huomattavasti nopeammin kuin mikäli ensin suunniteltaisiin koko ohjelmisto ja sitten luovutettaisiin se toteuttajille. Shinkle [Shi09] tukee Midletonin väitettä ja esittää, että kun tekeillä olevien asioiden lukumäärä on rajoitettu, voidaan laskea kuinka kauan kukin tehtävä voi olla linjalla niin, että kuitenkin päästään aikataulullisiin ja budjettitavoitteisiin. Myös Poppendieck ja Poppendieck [PoP03] väittävät, että töiden määrän pitäminen alhaisena parantaa huomattavasti läpimenoaikaa. Esimerkiksi, mikäli työtehtäviä on 20, on huomattavasti nopeampaa tehdä ne yksi kerrallaan kuin kaikki yhtä aikaa.

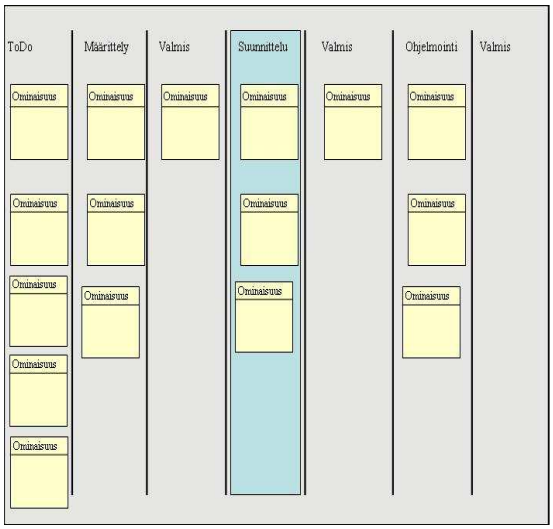
Mikä sitten olisi ihanteellinen tekeillä olevan työn määrä? Sinkle [Shi09] päätyy

tulokseen, että minimointi ei olekaan paras ratkaisu. Kun tiimi juuttuu yhteen tehtävään eikä sen ammattitaito riitä sen ratkaisemiseen, tuo tehtävä jätetään pöydälle ja tehdään muut tekeillä oleva työt valmiiksi. Kuitenkaan, kun uutta työtä ei saa aloittaa, aikaa kuluu turhaan odotteluun. Tiimin jäsenet myös turhautuvat ja heidän motivaationsa laskee. Myös liian suuri tekeillä olevien tehtävien määrä osoittautuu huonoksi. Tämä aiheuttaa enemmän uudelleen tehtävää työtä ja pidempiä odotteluaikoja. Tekeillä olevien tehtävien ihanteellinen määrä pitäisi siis jotenkin pystyä laskemaan tai päättelemään.

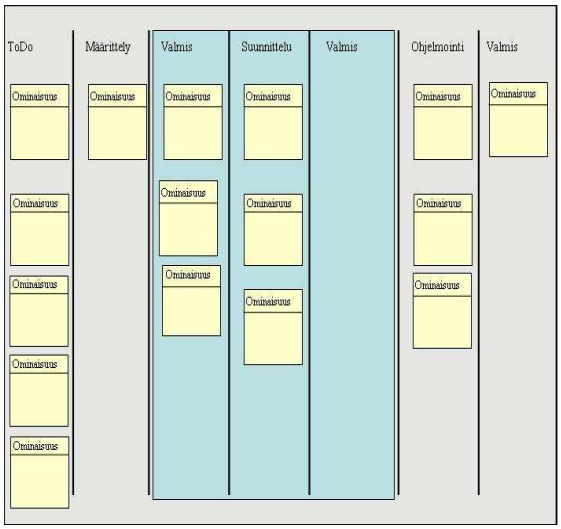
Jotta läpimenoaika olisi mahdollisimman lyhyt, olisi odottamista oltava Lean-filosofian mukaisesti työnkulussa mahdollisimman vähän [Lad08]. Kanbanissa ohjelmistoprosessi on visualisoitu siten, että kukin työvaihe luodaan omaksi sarakkeekseen, esimerkiksi määrittely, suunnittelu, ohjelmointi sekä testaus. Oletetaan, että ohjelmointivaihe saa työnsä valmiiksi ja voisi vetää suunnitteluvaiheelta seuraavan työn. Mitä tehdään, jos suunniteluvaiheessa kaikki työt ovatkin vielä kesken? Tämän vuoksi Ladas [Lad08] ehdottaa, että kutakin vaihetta seuraa valmis-vaihe. Tästä vaiheesta vedetään seuraavan vaiheen työ. Ladaksen mukaan ideaali olisi, että valmis-vaiheessa olisi aina yksi työ valmiina vedettäväksi eteenpäin, mutta korkeintaan siinä saa Ladaksen esimerkissä olla kolme työtä. Näin seuraava vaihe ei milloinkaan joutuisi odottamaan uutta valmistunutta työtä.

Ladas [Lad08] sekä Kniberg [Kni09] väittävät, että kun tekeillä olevien asioiden määrä on rajoitettu, huomataan ohjelmistoprosessissa esiintyvät ongelmat nopeasti. Myös Poppendieck ja Poppendieck [PoP03] väittävät, että Just-in-Time –tuotanto paljastaa laatuvirheet heti ja lisäksi vaatii paljon kommunikaatiota työntekijöiden kesken. Seuraava esimerkki pohjautuu Ladaksen [Lad08] kirjaan. Esimerkissä kukin vaihe voi tehdä korkeintaan neljää työtä yhtäaikaisesti, mukaan lukien vaiheesta valmistuneet työt. Kuvissa 11a, 11b, 11c ja 11d esitetään esimerkki työnkulusta, jossa alkaa ilmetä ongelmia. Asiakkaan käyttäjäkertomukset on tässä esimerkissä pilkottu pieniksi osakokonaisuuksiksi, ominaisuuksiksi. Kuvassa kunkin vaiheen jälkeen on valmistuneiden töiden välivarasto, josta seuraava työvaihe ottaa uudet työnsä. Esimerkin kuvassa 11a suunnitteluvaiheen työntekijöillä on ongelmia jonkin ominaisuuden suunnittelun kanssa, mutta kukaan ei tiedä siitä vielä, eivätkä ongelmat näy vielä

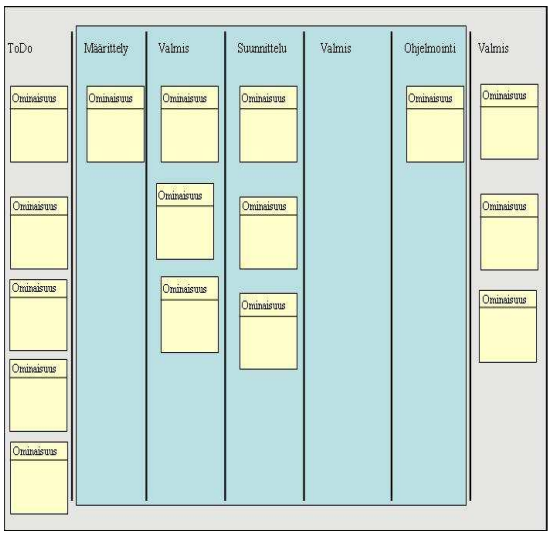
seinätaululla.



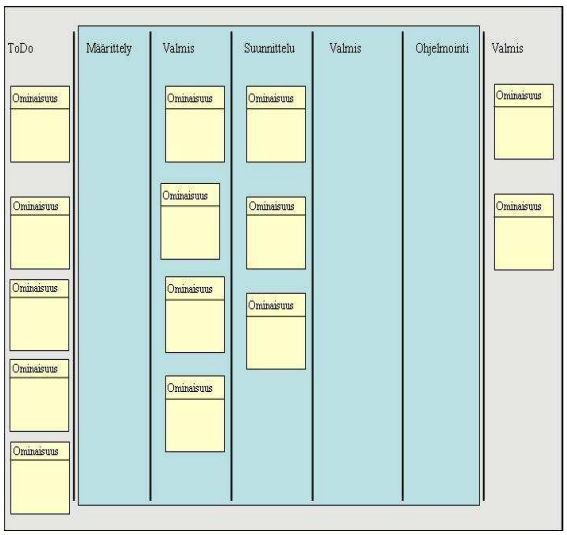
Kuva 11a



Kuva 11b



Kuva 11c



Kuva 11d

Kuvat 11a, 11b, 11c ja 11d: Esimerkki [Lad08, s.140] ongelmien paljastumisesta Kanbanin avulla.

Seinätaululla ongelma tulee kuitenkin esiin varsin nopeasti. Kuvassa 11b nähdään jo, kuinka määrittelyvaiheen valmistuneet-sarake alkaa täyttyä ja määrittelyvaiheen tekeillä olevan työn määrän maksimi alkaa lähestyä. Suunnitteluvaihe ei voi ottaa uusia töitä, koska se taistelee ongelman ratkaisemiseksi. Ohjelmointivaihe tekee edelleen työtään normaalisti, mutta vetää viimeisen suunnittelusta valmistuneen työn. Nyt, kun ongelma on jo huomattu, voidaan ryhtyä toimenpiteisiin. Koska määrittelyvaiheen tekeillä olevan työn maksimi alkaa olla täynnä, ja suurin osa sen työstä on jo tehtynä, voidaan määrittelyvaiheesta vapauttaa henkilö etsimään suunniteluvaiheen ongelman ydintä.

Kun suunniteluvaihe taistelee ongelman kanssa eikä saa työtä valmiiksi, loppuvat ohjelmointivaiheelta tulevat työt (kuvassa 11c). Näin myös ohjelmointivaiheessa työskentelevät henkilöt vapautuvat selvittämään ongelmaa. Määrittelyvaiheen lähes kaikki työ on valmiina. Näin määrittelyvaiheesta vapautuu lisää henkilöitä ratkaisemaan suunnitteluvaiheen ongelmaa.

Lopulta työnkulku on jumissa, kuten kuvassa 11d nähdään. Määrittelyvaihe on käyttänyt kaikki sen tekeillä olevien töiden Kanban-laput ja suunnitteluvaihe ei saa yhtään työtä valmiiksi. Ohjelmointivaiheelta loppuvat vastaanotettavat työt.

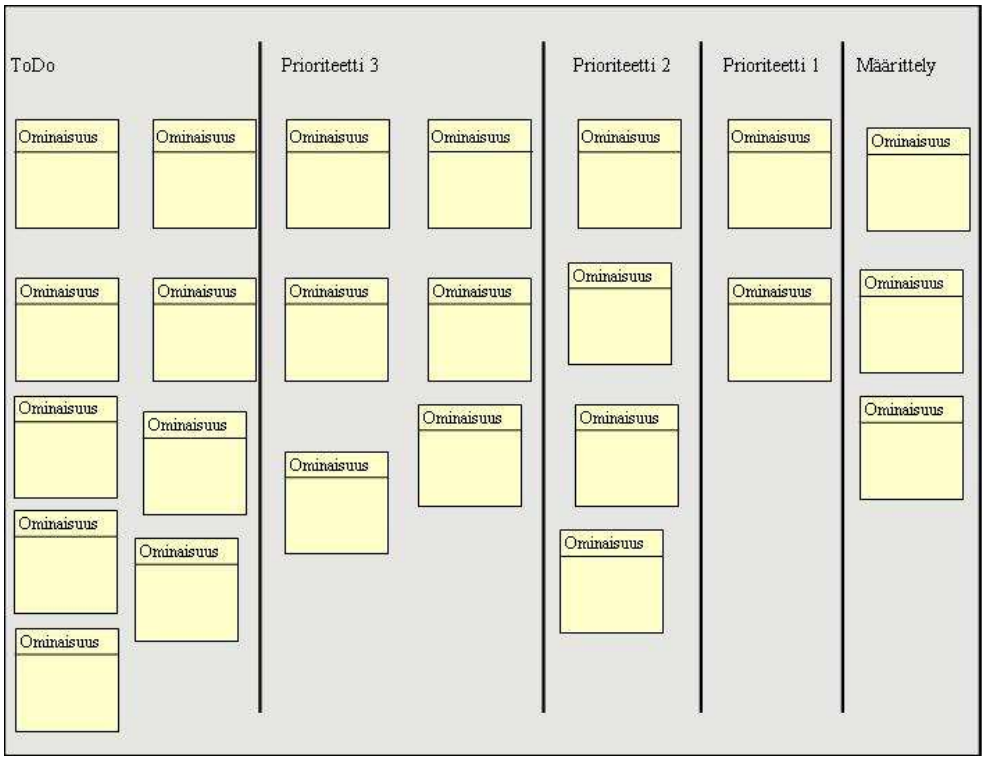
Työnkulku ei Ladaksen esimerkissä siis jumiudu heti, vaan ongelma voidaan Kanbanin avulla huomata nopeasti, jo ennen kuin se aiheuttaa suurempia hidastumisia kokonaisprosessissa. Ongelmatilanteeseen voidaan puuttua, kun se on päällä, sitä ei huomata vasta esimerkiksi kuukauden kuluttua katselmointitilaisuudessa. Ongelmanratkaisuun voidaan ottaa tilapäisesti työvoimaa muista vaiheista, ja he voivat selvittää Lean-filosofian mukaisesti ongelman perimmäistä syytä.

Middleton [Mid01] ehdottaa, että kussakin työnkulun askeleessa toimijat eivät saa siis ottaa mitään muita tehtäviä vastaan, kuin mitä linjaa pitkin on tullut. Näin työ ei kasaudu ja aiheuta tukoksia ja viivästymisiä sekä laadun huononemista linjalla. Shinkle [Shi09] väittää, että kasautuva työ aiheuttaa todellisia ongelmia kehitystiimeille. Mikäli joku tiimin henkilö juuttuu johonkin asiaan, tehtävä pystytään ratkaisemaan lisäämällä siihen henkilöitä, joilla on asiantuntemusta asiasta.

Alwardt ja kumppanit [AlM09] esittävät, että työt on priorisoitava ja korkeimman prioriteetin tehtävät tehdään ensin keskittymättä muihin tehtäviin samanaikaisesti. He ehdottavat, että projektin johdon on tehtävä priorisointi, mutta eivät kerro, miten

priorisointi tulisi käytännössä tehdä. Heidän mukaansa ohjelmistotuotannossa saattaa olla houkuttelevaa tehdä nopeasti tai helposti valmistuvia tehtäviä ensin, mutta tällainen johtaisi tärkeämmän prioriteetin toimintojen viivästymiseen aikataulussa. Esimerkiksi suunnittelun valmis-vaiheeseen saattaa juuttua jokin työ, jos muita helpommin toteutettavia töitä otetaan jatkuvasti ohjelmointivaiheeseen. Tämän vuoksi priorisoinnin mukainen töiden ottaminen onkin suositeltavaa.

Ladas [Lad08] esittää priorisointia käytettäväksi myös ToDo–listaa tehtäessä. Asiakkaalla on yleensä runsaasti vaatimuksia, jotka ovat kaikki korkealla prioriteetilla. Mitkä näistä asioista pitäisi siis ottaa ensin tehtäviksi, vaikkapa kolmeksi ominaisuudeksi? Kuvassa 12 esitetään Ladaksen [Lad08, s.166] ehdotus muodostaa kaikista tehtävistä asioista neljä eri saraketta. Ensimmäisessä sarakkeessa ovat kaikki asiakkaan määrittelemät ToDo –listan asiat, työtehtävät, jotka pitää tehdä, mutta joille ei vielä ole annettu prioritettiastetta. Prioriteetti 3 -sarakkeessa ovat ne työtehtävät, jotka ovat kohtuullisen pian tehtäviä, ei kuitenkaan ihan heti. Prioriteetti 2 –sarakkeessa ovat ne työtehtävät, jotka pitää aloittaa heti, kun vain vapaa työntekijä liikenee. Prioriteetti 1 –sarakkeessa ovat ne tehtävät, joita tehdään parhaillaan tai jotka on tehtävä heti. Jokaiselle sarakkeelle on määritelty maksimi Kanban-lappujen määrä, joka pienenee sarake sarakkeelta, esimerkiksi 8-5-3. Kun jokin asia otetaan tehtäväksi määrittelyvaiheeseen, laukaisee se tarpeen ottaa prioriteetti 2 –sarakkeesta uusi ominaisuus kolmoseen ja niin edelleen ToDo –listalta ottamiseen asti. Tämä malli tukee Lean-filosofian myöhäistä päätöksentekoa, kun jokaista toteutettavaa ominaisuutta harkitaan ainakin kolmesti ennen kuin se otetaan määrittelyvaiheeseen. Välillä tehtäväksi tulee pakollisia, välittömästi tarvittavia asioita, jotka hyppäävät suoraan prioriteetti 1 –sarakkeeseen. Välillä taas jokin ominaisuus ei koskaan etene listalla vaan putoaa siitä pois tarpeettomana.



Kuva 12: Runsaiden vaatimusten priorisointi Ladaksen [Lad08, s.166] mukaan.

Alwardt ja kumppanit [AlM09] väittävät, että kun töitä otetaan vastaan vain vähän linjalta, myös vuorottelu eri tehtävien välillä vähenee. Kun tehtävien välillä vuorotellaan, aikaa kuluu asioiden palauttamiseen mieleen, kokonaistyömäärä kasvaa ja työn laatu voi huonontua. Ihminen ei pysty tekemään montaa työtä yhtä aikaa. Periaatteena onkin, että vain yhtä asiaa tehdään kerrallaan eikä uutta aloiteta ennen kuin se on valmis. Tällainen työnkulussa olevien asioiden rajoittaminen itse asiassa johtaa tehtävien nopeampaan valmistumiseen, kuin jos tehtäisiin monta asiaa yhtäaikaisesti.

Middleton [Mid01] ehdottaa, että mikäli ohjelmistossa havaitaan virhe, joko määrittelyissä, suunnittelussa tai tuotannossa, pysäytetään työnkulku, työ palautetaan takaisin päin linjalla, esimerkiksi toteutuksesta takaisin suunnitteluun, ja virhe korjataan. Ladas [Lad08] esittää erillisen ongelmankorjausyksikön perustamista. Yksikön työntekijät pyrkivät Lean-filosofian mukaisesti löytämään ja poistamaan ongelman pohjimmaisen syyn. Tällöin linjan toimijat voivat ottaa muita tehtäviä tilapäisesti kunnes virhe on korjattu ja työnkulku taas toimii.

Ladas [Lad08] väittää, että työnkulkuun tulevien tehtävien tulisi olla mahdollisimman

samanlaisia ilman suuria vaihteluita. Ohjelmistoprojektissa kuitenkin vaatimukset voivat olla hyvinkin erilaisia. Jotkut saattavat olla helppoja toteuttaa, mutta hyvin vaikeita testattavia. Toiset puolestaan voivat olla hyvin vaikeita suunniteltavia, mutta yksinkertaisia ohjelmoida. Tähän Ladas [Lad08] ehdottaa seuraavanlaisia parannustoimia:

- määrittelyvaiheessa kiinnitetään huomioita suurien kokonaisuuksien tunnistamiseen ja
- tukoksia aiheuttavat asiat huomataan heti ja niihin puututaan välittömästi.

Ladas [Lad08] sekä Kniberg ja Skarin [KnS10] ehdottavat, että Kanbania käytettäessä suuria kokonaisuuksia ei oteta tekeille, sillä suuret kokonaisuudet pilkotaan ensin pienemmiksi tehtäviksi. Piirteet pidetään pieninä, koska silloin ne liikkuvat nopeasti linjan läpi [Lad08]. Myös jälkimmäiseen parannustoimeen Kanban tuo ratkaisun, sillä kuten edellä esitettiin, tukokset huomataan lähes välittömästi. Ladaksen [Lad08] mukaan valmis-puskurit jokaisen työvaiheen jälkeen pienentävät vaihtelujen merkitystä työnkulussa.

Shinkle [Shi09] väittää, että Software Engineering Professionals -yrityksessä tiimi saatiin Kanbanin avulla ymmärtämään sekä toteuttamaan menestyksekkäästi Lean-filosofian periaatteita ohjelmistotuotannossa. Tiimi otti käyttöön Kanban-menetelmän vuoden 2007 alussa. Kanbania käytettiin yli kahdessatoista projektissa puolen vuoden aikana. Työnkulun läpimenoaikaa sekä virheiden määrää seurattiin. Puolen vuoden jälkeen projektipäällikköä haastateltiin.

Seurannan alussa ohjelmistotuotantoprojekti oli paisunut käsistä, tiimi oli kasvanut kompaktista neljän hengen tiimistä yli kolminkertaiseksi, teknologinen alusta jouduttiin muuttamaan siitä, mihin tiimi oli tottunut, ja toimituksen takaraja alkoi painaa päälle. Ongelmia tuli myös tiimin johtamisessa: kukaan ei enää tiennyt, missä projektissa oikeastaan mennään. Koko tiimissä vallitsi turhautunut ja tyytymätön ilmapiiri. Liian vaikeita töitä oli kasautunut ilman, että kukaan otti niistä vastuuta.

Projektin puolivälissä Kanban esiteltiin projektille. Lähes heti tuli esiin olennainen seikka: tiimillä oli valtavasti työtä parhaillaan menossa. Projektille määriteltiin puoliväkisin rajat, kuinka monta asiaa kerrallaan voi olla tekeillä. Tämän jälkeen tiimi

alkoi ottaa vastuuta projektista, läpimenomäärä parani, virheiden määrä laski, sitoutuminen projektille parani ja jokainen alkoi osallistua toimintaan tekemällä ratkaisuehdotuksia. Kun Kanban oli ollut käytössä puoli vuotta, projektipäällikkö sanoi: ”Olisimmepa ottaneet Kanbanin käyttöön jo aiemmin” [Shi09].

Kanban-menetelmä kertoo työntekijälle, mitä työtehtäviä seuraavaksi odottaa [PoP03]. Työ on itseohjautuvaa Lean-filosofian mukaisesti, sillä työntekijä voi itse valita haluamansa työtehtävät. Asiakkaan vaatimukset ovat kuitenkin kaiken lähtökohta, Lean-filosofian mukaisesti [PoP03]. Meneillään olevan työn status, tilanne, on koko ajan kaikille näkyvillä seinätaululla [PoP03]. Lean-filosofian mukainen työnkulun hallinta kertoo kaikille kristallinkirkkaasti prosessin tilanteen [Lad08].

3.2.3 Kanban-menetelmän soveltuvuudesta ketterään ohjelmistokehitykseen

Ladas [Lad08] väittää, että ketterän ohjelmistokehityksen periaatteissa ja Lean-filosofiassa on paljon yhteistä. Hänen mukaansa ketterien menetelmien ongelmana on ollut keskittyminen tiettyyn, kapea-alaiseen prosessiin näkemättä asiaa laajasta perspektiivistä. Myös Abrahamsson ja kumppanit [Abr03] paljastavat, että johdon näkökulma puuttuu useimmista ketteristä menetelmistä. Poppendieck [PoP07] esittää, että Lean-filosofia tarjoaa teorian ketterien menetelmien käytäntöjen perustaksi. Lean-filosofian avulla yrityksen johto voi muokata ohjelmistoprosessistaan juuri omaan liiketoimintaympäristöönsä ja tilanteeseensa sopivan.

Lean-filosofiassa tai Kanbanissa ei ole varsinaista ajankäytön jaksottamista siinä mielessä kuin se ketterissä malleissa ymmärretään [Kni09]. Ketterissä malleissa suoritetaan tietyn kestoinen iteraatio, kun taas Kanbanissa tekeillä olevista asioista kertovat laput liikkuvat taululla sen mukaan, kuin asia etenee. XP-menetelmässä pariohjelmoijien työtä käsitellään ikään kuin työnkulkuina, joten XP:tä käytettäessä Kanban-menetelmän kolmen perusperiaatteen ottaminen käyttöön on Ladaksen [Lad08] mukaan pieni askel.

Kanbanissa tekeillä olevien asioiden määrä on rajoitettu, kun taas ketterissä malleissa kuhunkin iteraatioon voidaan ottaa tekeillä olevaan työhön periaatteessa kuinka paljon asioita tahansa [Kni09]. Kniberg [Kni09] väittää, että esimerkiksi Scrumin neljän viikon iteraatiossa tekeillä olevaa työtä on huomattavasti enemmän, kuin mitä Kanbanin

mukaan on suositeltavaa nopean läpimenon varmistamiseksi. Kukin iteraatiokierros tulisi pitää hyvin lyhyenä, muutoin järjestelmä saattaa muuttua Just-in-Time -pohjaisesta työntöpohjaiseksi huomaamatta [PoP03]. Jos iteraatio on liian pitkä, muodostuu liikaa Kanban-tehtäviä, eikä kokonaisuutta pysty enää hallitsemaan [PoP03, Lad08]. Ladaksen [Lad08] mukaan iteraatioita ei kuitenkaan tarvita lainkaan, jos Just-in-Time -pohjainen järjestelmä toimii hyvin. Iteraatioiden tuotteen työlistan sisältämiä ominaisuuksia on nimittäin huomattavasti enemmän, kuin mitä tarvittaisiin toteuttaakseen seuraava ominaisuus, joten ne ovat Lean-filosofian mukaan jätettä.

Ladas [Lad08] ehdottaa Scrumin toteutusvaiheen työlistan ja työnkulun väliin seuraavaksi toteutetaan -vaihetta, johon siirretään korkeimman prioriteetin asiat työlistalta. Kun henkilö työnkulusta vapautuu, hän ottaa työn seuraavaksi toteutetaan –vaiheesta ja työlistalta valitaan sinne uusi työ. Seuraavaksi toteutetaan –vaiheen töiden määrä on rajoitettu, esimerkiki kolmeen ominaisuuteen. Tämä toimintatapa toteuttaa Kanbanin Just-in-Time -periaatetta.

Poppendieck ja Poppendieck [PoP03] ehdottavat Scrumin päivittäisten lyhyiden tiimipalaverien ottamista käyttöön myös Kanban-mentetelmässä. Ohjelmistokehittäjät voivat palaverissa kertoa mitä he ovat edellisenä päivänä tehneet, minkä työtehtävän he ajattelevat ottaa seuraavaksi harteilleen, tarvitsevatko he mahdollisesti apua tehtävän suorittamisessa. Samaa suosittaa myös Ladas [Lad08], mutta hän ehdottaa tiimien keskittyvän palaverissa enemmän taktiseen suunnitteluun, esimerkiksi siihen, miten työnkulkua voitaisiin parantaa tai alkavia ongelmia ehkäistä.

Alwardt ja kumppanit [AlM09] väittävät, että ongelmaksi ohjelmistotuotannossa saattaa muodostua asiakkaan muuttuvat vaatimukset tai kokonaan uusien vaatimusten lisääminen. Tärkeämmän prioriteetin asiat tulisi Kanbanissa tehdä kokonaan valmiiksi testausta ja käyttöönottoa myöten ennen kuin otetaan uusia tehtäviä vastaan. Tämä toteutuukin ketterissä malleissa, joissa jokaisen syklin jälkeen asiakas saa osaltaan valmiin tuotteen.

3.3 Yhteenveto

Lean-filosofia tarjoaa toiminnalle perustan, filosofian. Lean-filosofiasta saadaan ne työkalut, joilla koko organisaatio saadaan viritettyä virtaviivaiseksi. Pyrkimyksenä on

tuoda prosessiin tärkeänä asiakkaan rooli sekä syvä ymmärrys markkinoista. Kanban-menetelmä on osa Lean-kokonaisuutta. Kanban toteuttaa Just-in-Time –tuotantoa, jossa asiakas ikään kuin imee tuloksen tuotantoprosessilta sen sijaan, että tuotettaisiin varmuuden vuoksi varastoon, jos asiakas sattuu tarvitsemaan. Kanban-menetelmän avulla tuotantoprosessista voidaan löytää ja poistaa jätettä, ongelmat tulevat heti esiin, tuotantomäärät saadaan pidettyä tasaisena ja prosessi visualisoidaan seinätaululle.

Kanban-menetelmä soveltuu tuotantotalouden lisäksi myös ohjelmistoprosessiin prosessimallista riippumatta. Kanbanissa työnkulut visualisoidaan seinätaululle, työnkulun vaiheet muodostavat seinätaulun sarakkeet. Tekeillä olevat asiat ovat näkyvillä seinätaululla. Niiden määrä on rajoitettu, jotta tehtävien kulku työnkulussa saataisiin optimaalisen nopeaksi ja tasaiseksi ja tukokset näkyville mahdollisimman pian.

4 Arvon tuottaminen asiakkaalle ohjelmistokehityksessä

Edellisessä luvussa esiteltiin Kanban-menetelmä. Jotta voidaan tutkia, kuinka Kanban vaikuttaa arvon muodostumiseen, perehdytään tässä luvussa arvon käsitteeseen kirjallisuudessa.

Taylor [Tay86] ja Patton [Pat08] esittävät, että arvon käsite on hyvin moniulotteinen; tietojärjestelmän arvo riippuu siitä kenen kannalta sitä katsotaan. Simmons [Sim96] väittää, että yleensä tietojärjestelmän arvo nähdään sen vaikutuksena osakkeenomistajan kannalta, taloudellisena kannattavuutena, ja muut sidosryhmät kuten yrityksen omat asiakkaat tai tietojärjestelmän käyttäjät jätetään vähemmälle huomiolle. Tässä tutkielmassa tarkastellaan arvon muodostumista asiakkaan, eli tietojärjestelmän ostajan, näkökulmasta.

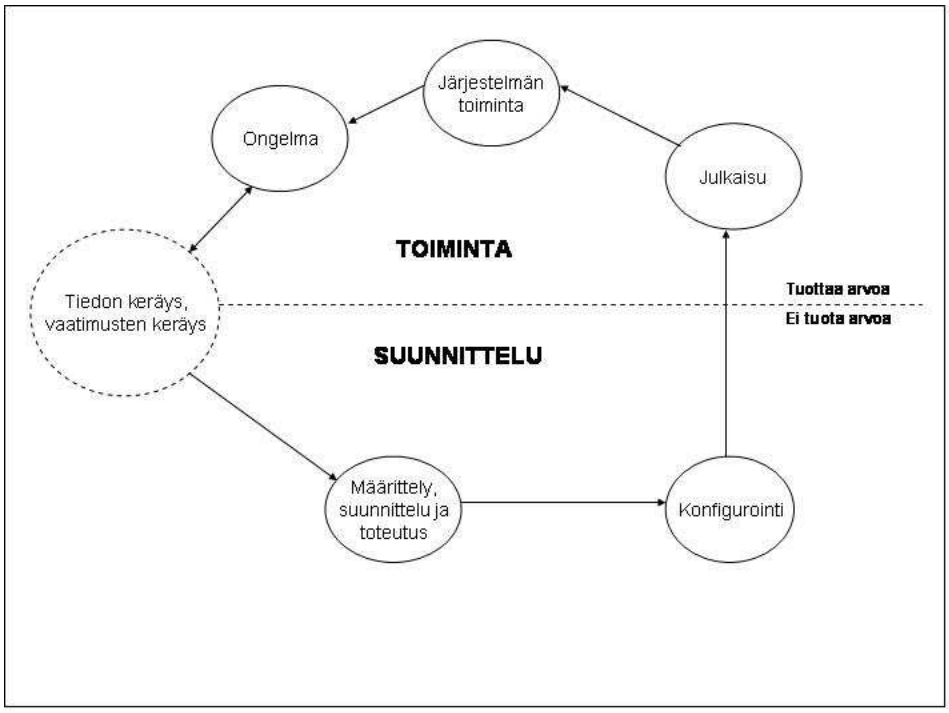
Luvun aluksi kuvataan, mitä kirjallisuuden perusteella arvo tarkoittaa ohjelmistokehityksessä. Tämän jälkeen esitellään keino tarkastella arvon muodostumista ohjelmistoprosessissa: työnkulun visualisointi. Lopuksi muodostetaan yhteenveto luvun pääkohdista.

4.1 Arvon muodostumisesta ohjelmistokehityksessä

Patton [Pat08] väittää, että tietojärjestelmä tuottaa yritykselle liiketoiminnallista arvoa, jos se tuottaa tulojen kasvua, vähemmän kustannuksia tai parannusta palveluun. Jokaista järjestelmän ominaisuutta tulisi Favaron [Fav96] mukaan tarkastella sen kannalta, tuottaako se kilpailuetua. Mohamed ja Wahba [MoW08] esittävät, että tietojärjestelmä voi tuottaa myös ei-liiketoiminnallista arvoa, kuten markkinaosuuden kasvua, yrityksen luotettavuuden paranemista, pienempiä riskejä tai turvallisuuden paranemista.

Pattonin [Pat08] mukaan jotta tietojärjestelmä voisi tuottaa arvoa, ei riitä sen julkaiseminen tuotantoon, vaan sitä on oikeasti myös käytettävä. Taylor [Tay86] muodostaa arvon määrittelemiseksi mallin, jossa informaatiojärjestelmän arvo muodostuu käyttäjän kokemuksen mukaan. Järjestelmän arvo voi hänen mukaansa muodostua myös teknologian perusteella tai tietojärjestelmän sisältämän tiedon mukaan. Taylor [Tay86] näkee arvon muodostumisen käyttäjälähtöisenä; arvoa ei ole ellei käyttäjä koe sitä. Esimerkiksi hyvin organisoitu tietokanta ei sellaisenaan ole arvoa. Arvo asiakkaalle muodostuu kahdesta asiasta: vaihtoarvo ja käytettävyysarvo. Vaihtoarvo tarkoittaa sitä hintaa, vaivaa ja aikaa jonka asiakas on valmis kuluttamaan tuotteen saadakseen. Tietokannassa olevalla tiedolla ei ole arvoa ellei joku käytä sitä. Käytettävyysarvo puolestaan kuvaa sita, kuinka käyttäjä kokee käyttöliittymän toimintojen tuottavan arvoa, olevan hyödyllisiä.

Kuvassa 13 esitetään, missä tietojärjestelmän kehityksen vaiheissa arvoa muodostuu asiakkaalle Taylorin [Tay86] mallin mukaan. Informaation keräysvaihe, käyttäjäympäristön ja ongelman kuvaus sekä vaatimusten keräysvaihe ovat osittain arvoa tuottavaa työtä, sillä nämä muodostuvat suoraan asiakkaan tarpeen perusteella. Järjestelmän määrittely-, suunnittelu- ja toteutusvaiheessa mikään ei vielä tuota käytettävyysarvoa asiakkaalle. Myöskään teknisen ympäristön muokkaaminen järjestelmän toimintaa varten, konfigurointi, ei tuota arvoa. Vasta, kun järjestelmä julkaistaan asiakkaan käyttöön, alkaa se tuottaa käytettävyysarvoa.



Kuva 13: Ohjelmistoprosessin käytettävyysarvoa tuottavat osat Taylorin [Tay86] mukaan.

Favaro [Fav02] esittää erilaisen näkökulman, jonka mukaan myös määrittely- ja suunnitteluvaihe tuottavat arvoa, sillä myös määrittelyistä on muodostettavissa uudelleenkäytettäviä osia. Näin asiakkaalle voidaan tuottaa arvoa jo ohjelmistoprosessin varhaisimmissa vaiheissa. Myös määrittelyprosessin tavoitteena on lisätä liiketoiminnallista arvoa. Poppendieck ja Poppendieck [PoP03] ehdottavat, että myös ohjelmakoodin kirjoittaminen, tarkastelu, korjaaminen ja uudelleenkirjoittaminen elegantin, ammattimaisen lopputuloksen aikaansaamiseksi tuottavat arvoa. Proosan tuottamisessa on tavallista, että teksti kirjoitetaan uudelleen muutamankin kerran täydellisen lopputuloksen saavuttamiseksi, miksei siis myös ohjelmakoodia kirjoitettaessa.

Taylorin [Tay86] käyttäjälähtöisessä mallissa on tietojärjestelmän arvo käyttäjälle luokiteltu kuuden kriteerin mukaan:

- käytön helppous,
- turhan tiedon suodattaminen,

- tiedon paikkansapitävyys, täydellisuus ja luotettavuus,
- vastaus ongelmaan, sopivuus, joustavuus, edelleenkehitettävyys,
- ajansäästö ja
- kustannussäästö.

Käytön helppous tarkoittaa sitä, kuinka järjestelmä auttaa käyttäjää. Selaustoiminnolla autetaan käyttäjää löytämään hänelle arvokas tieto. Tiedon esittämisellä tarkoitetaan tiedon muotoilemista näytölle käyttäjälle parhaalla mahdollisella tavalla, siten, että esimerkiksi tärkeät sanat on korostettu, jotta käyttäjän on helppo löytää olennainen. Myös kokonaisjärjestelmän ymmärtäminen on arvoa: käyttäjä voi hyödyntää paremmin järjestelmää. Fyysinen saatavuus tarkoittaa pääsyä tiedon säilytyspaikkaan.

Turhan tiedon suodattamisella tarkoitetaan sitä, että tietojärjestelmä suodattaa sinne syötettävää tietoa siten, että tarpeetonta tietoa ei voi syöttää. Turhan tiedon poistaminen tarkoittaa järjestelmän suorittamaa automaattista sellaisen tiedon poistamista, jota kukaan ei käytä. Tiedon laatuun liittyvät arvon mittarit ovat tiedon paikkansapitävyys eli se, että tieto näytetään käyttäjälle alkuperäisenä, tiedon pysyminen muuttumattomana järjestelmässä liikuessaan, tieto kuvaaminen mahdollisimman laajasti aihettaan, tiedon ajantasaisuus eli järjestelmässä ei ole vanhentunutta tietoa, tiedon luotettavuus sekä tiedon oikeellisuus.

Tiedon sopivuus tarkoittaa järjestelmän tekemiä valintoja, jotka vahvistavat vastauksen saamista käyttäjän ongelmaan. Näitä ovat järjestelmän tekemät aktiviteetit, jotka auttavat saamaan asiakkaan ongelman ratkaistua, järjestelmän joustaminen tarjoamalla käyttäjälle useita lähestymistapoja tietoon, tiedon näyttäminen ja säilyttäminen mahdollismman yksinkertaisessa muodossa siten, että käyttäjä voi sen helposti ymmärtää, ja tiedon jalostaminen toiseen muotoon siten, että sitä voi käyttää muussa yhteydessä.

Ajan säästäminen on arvo käyttäjälle: kuinka nopeasti haluttu tieto on saatavilla tai kuinka helposti tietty raportti saadaan tuotettua järjestelmästä. Kustannussäästö tarkoittaa niitä rahassa mitattavia järjestelmän suunnittelun ja toiminnan päätöksiä, jotka säästävät rahaa asiakkaalle. Arvon tuottaminen ei ole kiinni teknologiasta. Arvo käyttäjälle syntyy lähinnä siinä, että tieto löytyy silloin ja siellä missä sitä tarvitaan.

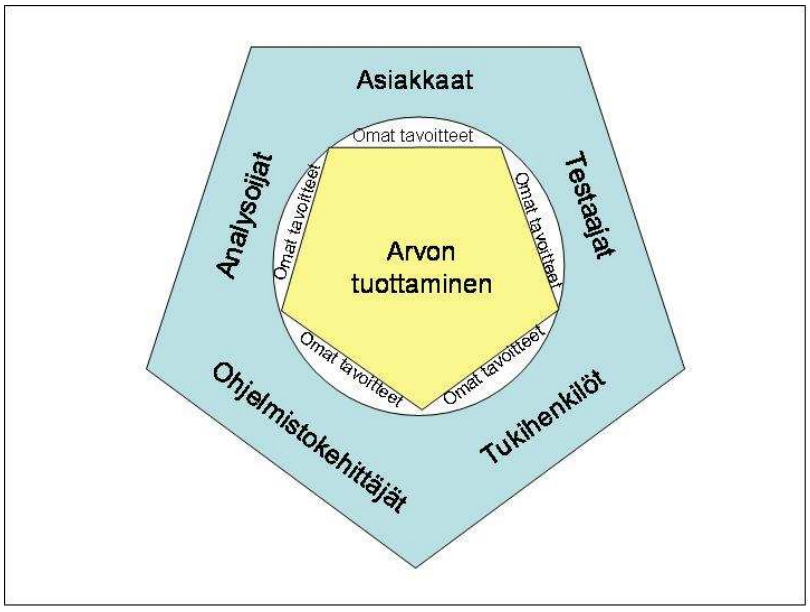
Raman [Ram98] esittää, että ohjelmistojen tuottamisessa Just-in-Time -tuotantoa toteutetaan tekemällä vain sitä, mitä asiakas tarvitsee. Raman [Ram98] väittää, että asiakkaan vaatimuksista tiedetään, mitä asiakas tarvitsee. Asiakas kuvailee vaatimuksiaan sekä kertoo, mikä hänelle tuottaa arvoa. Raman [Ram98] kuitenkin ehdottaa, että ohjelmistojen tekemisessä asiakkaan vaatimukset eivät voi olla ainoa arvo, vaan tuotteen on oltava myös muuten hyvin suunniteltu, esimerkiksi muunneltavissa ja ylläpidettävissä. Middleton [Mid01] väittää, että tuote itsessään on arvoton, kunnes se on valmis. Vaatimusmäärittelyillä ja suunnitelmilla voi olla oppimisarvoa, mutta asiakkaalle tuotteella on arvoa vasta, kun se on valmis ja alkaa tuottaa asiakkaalle jotakin.

Poppendieck ja Poppendieck [PoP03] esittävät, että nykyisin toimituksen nopeutta pidetään arvokkaana ohjelmistokehitysprojekteissa, sillä se lisää liiketoiminnallista joustavuutta nopeasti muuttuvassa toimintaympäristössä. Myös Raman [Ram98] väittää, että jokaisen ohjelmistoja ostavan yrityksen toiveena on saada ohjelmistoja edullisemmin, nopeammin ja laadukkaampina. Nopean tuotekehityksen avulla tuote voidaan toimittaa ennen kuin asiakas ehtii muuttaa mieltänsä tai korjata vaatimuksiansa. Tällöin myöskään kesken olevaa työtä ei ole niin paljon. Nopeaan toimitukseen sitoutuminen vaatii kulttuurinmuutosta koko organisaatiolta, eikä se tule sattumalta. Jokaisen henkilöstössä on tiedettävä mitä työtehtäviä kunakin päivänä tekee, jotta jätettä ei pääse syntymään. Tärkeimmän prioriteetin vaatimusten tuottaminen nopeasti valmiiksi osakokonaisuuksiksi asiakkaalle sen sijaan tuottaa asiakkaalle mahdollisimman paljon liiketoiminta-arvoa.

Poppendieck ja Poppendieck [PoP03] väittävät, että johtamiseen ja projektinhallintaan liittyvät toiminnot eivät suoraan lisää liiketoiminnallista arvoa lopputuotteeseen, mutta niillä on suuri vaikutus jätteen poistamisen kannalta. Esimerkiksi julkaisujen suunnittelu ja vaatimusten priorisointiprosessi vaikuttavat suoraan siihen, että keskeneräistä työtä on mahdollisimman vähän. Julkaisut tulisi suunnitella siten, että työtehtävät virtaavat tasaisesti vaatimuksista valmiiksi piirteiksi. Jos julkaisu suunnitellaan liian suureksi, saattaa tiimille tulla ongelmia laajan työn kanssa ja linjalle tulee tukos. Jos taas julkaisu suunnitellaan liian pieneksi, saattaa osa tiimistä olla toimettomana, kun työt on tehty ennen julkaisupäivää.

Poppendieck ja Poppendieck [PoP03] esittävät, että projektin seurantajärjestelmät eivät myöskään suoraan lisää liiketoiminnallista arvoa vaan päinvastoin saattavat olla turhaa työtä. Tällaisen järjestelmän hankkimisen sijaan tulisi huolehtia prosessin toimivuudesta. Monimutkainen muutosten ja muuttuneiden vaatimusten hyväksyttämisprosessi on myös usein implikaatio jätteestä eikä tuota arvoa asiakkaalle. Muutokset tulisi tunnistaa ja hyväksyä nopeasti.

Kuvassa 14 esitetään, kuinka Poppendieck ja Poppendieck [PoP03] ehdottavat, että kaikkien ohjelmistokehityksen osapuolten tulisi toimia yhteisen päämäärän saavuttamiseksi. Tämä päämäärä on liiketoiminnallisen arvon tuottaminen. Kuvassa 14 osapuolina on esitetty asiakkaat, analysoijat, ohjelmistokehittäjät, tukihenkilöt ja testaajat. Kullakin osapuolella on lisäksi oma tavoitteensa. Asiakas toivoo järjestelmän tuottavan liiketoiminnallista arvoa, analyytikot auttavat asiakkaan vaatimusten tarkentamisessa ja muokkaamisessa ohjelmistokehittäjiä varten, ohjelmistokehittäjät tuottavat toimivan ohjelmiston, tukihenkilöt suorittavat järjestelmän käyttöönoton ja käyttäjäkoulutukset, testaajat tuottavat kattavat testit, jotka takaavat, että ohjelmisto täyttää asiakkaan tarpeen. Asiakas, analyytikot, ohjelmistokehittäjät, tukitoimintojen työntekijät ja testaajat toimivat kaikki omien tavoitteidensa lisäksi yhteisen päämäärän eteen liiketoiminnallisen arvon tuottamiseksi asiakkaalle.



Kuva 14: Tiimin yhteinen tavoite on arvon tuottaminen Poppendieckin ja Poppendieckin [PoP03, s.107] mukaan.

Poppendieck ja Poppendieck [PoP03] esittävät, että ohjelmistokehityksessä osapuolia voi olla muitakin kuin kuvassa 14. Vastaavasti sidosryhmiä voi olla myös vähemmän, projektin luonteesta riippuen. Saattaa olla, että ohjelmistokehittäjät ottavat suoraan vastaan asiakkaan vaatimukset eikä analysoijia ole projektissa lainkaan. Voi myös olla, että ohjelmistokehittäjät suorittavat testauksen. Tärkeintä on, että kaikki osapuolet kommunikoivat suoraan keskenään ja pyrkivät helpottamaan tiedonvälitystä eri osapuolten välillä arvon kasvattamiseksi.

4.2 Työnkulun kartoitus keinona tarkastella arvon muodostumista

Poppendieck ja Poppendieck [PoP03] ehdottavat, että kartoittamalla työnkulku voidaan ohjelmistoprosessin työnkulun vaiheista erotella arvoa tuottavat sekä arvoa tuottamattomat vaiheet, jotka ovat jätettä. Työnkulku alkaa asiakkaan tarpeesta ja päättyy asiakkaan tarpeen täyttämiseen. Lopputuloksena saadaan kartoitus työnkulusta. Raman [Ram98] sekä Poppendieck ja Poppendieck [PoP03] väittävät, että jokaisen työnkulun askeleen pitäisi tuottaa arvoa, työnkulun askeleet yhdessä täyttävät asiakkaan tarpeen.

Kartoitus tarjoaa tarkkaa tietoa siitä, kuinka sisäiset prosessit toimivat – tai eivät toimi. Arvoa tuottaviksi prosesseiksi voidaan edellisen luvun perusteella laskea ne, jotka vaikuttavat arvoa tuottaviin aktiviteetteihin kehittämällä ja parantamalla niitä. Kartoittamalla työnkulku saadaan esiin kaikki yrityksen sisäiset prosessin osat, jotka jollakin tavalla kontribuoivat asiakkaan tarpeen täyttämiseksi.

Poppendieck ja Poppendieck [PoP03] ehdottavat, että työnkulun kartoitus tehdään kävelemällä kynä ja paperi kädessä organisaatiossa kuvitellen itsensä asiakkaan asemaan. Ei kysytä henkilöstöltä mitä missäkin vaiheessa tapahtuu, vaan havainnoidaan itse. Ensin mennään paikkaan, jossa asiakkaan tarve tulee organisaatioon. Päämääränä on piirtää kartta asiakkaan tarpeesta; kuinka se kulkee organisaatiossa saapumisesta valmistumiseen asti. Jokainen prosessin askel, joka käydään läpi tarpeen täyttämiseksi, luonnostellaan kartalle lisättynä keskimääräisellä ajalla, jonka asiakkaan tarve viipyy kussakin askeleessa. Kartan alareunaan piirretään aikaa kuvaava viiva, josta nähdään kuinka paljon tarve on ollut arvoa tuottavissa askeleissa sekä kuinka paljon tarve on

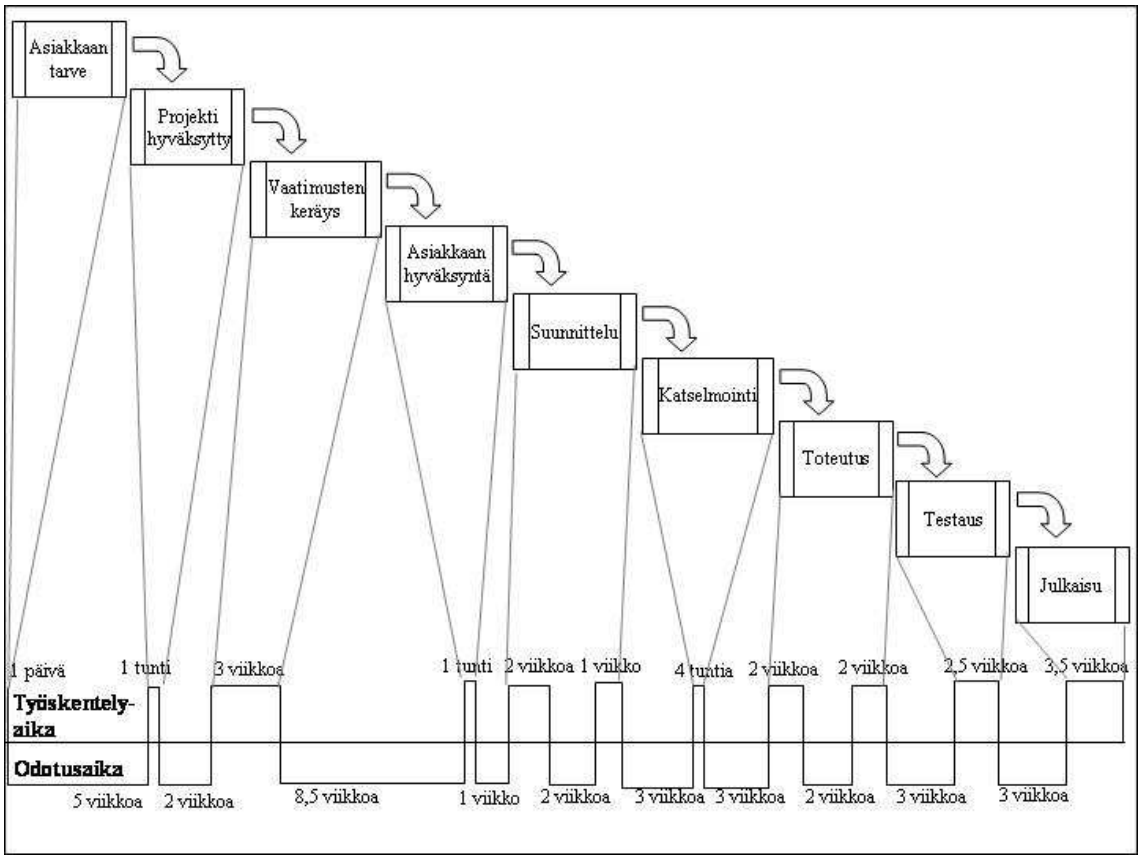
ollut arvoa tuottamattomissa askeleissa.

Poppendieck ja Poppendieck [PoP03] väittävät, että kartoitus tuo esiin sen, että tuottamattomat askeleet ovat suurimpia pullonkauloja ohjelmistokehityksessä. Analyysi tarjoaa lähtökohdan ohjelmistoprosessin arvioimiselle sekä kehittämiselle. Kun kartoitus on tehty, voidaan puuttua ensiksi kaikkein suurimpaan jatkuvan työnkulun ja jätteen ongelmaan. Tämän jälkeen kartoitus piirretään uudelleen ja analyysistä löydetään seuraava parannuskohde.

Organisaatiossa voidaan työnkulun kartoituksen avulla tarkastella prosesseista muodostuvaa kokonaisuutta. Työnkulun kartoitus on työkalu:

- jätteen löytämiseksi prosesseissa ja
- arvoa tuottavien toimintojen ryhmittämiseksi nopeaa virtausta varten.

Kun työnkulku on kartoitettu, tarkastellaan jokaista työnkulun askelta. Näitä askeleita arvioimalla voidaan löytää ja poistaa turhat askeleet, prosessin osat, jotka eivät tuota arvoa [PoP03]. Kuvassa 15 esitetään Poppendieckien [PoP03, s.10] kirjan perusteella esimerkki työnkulun kartoittamisesta. Esimerkissä ohjelmistokehitysmenetelmänä käytetään vesiputousmallia. Kuvasta nähdään, että projektin läpiviemiseen kuluu aikaa noin 48 viikkoa. Siitä noin 15,5 viikkoa on työskentelyaikaa ja noin 32,5 viikkoa kuluu odottamiseen. Poppendieck ja Poppendieck [PoP03] väittävät, että vesiputousmallia käytettäessä siis keskimäärin kolmannes työajasta tuottaa lisäarvoa, loput on Lean-filosofian mukaan jätettä.



Kuva 15: Esimerkki työnkulun kartoituksesta jätteen löytämiseksi, Poppendieckin ja Poppendieckin [PoP03, s.10] mukaan.

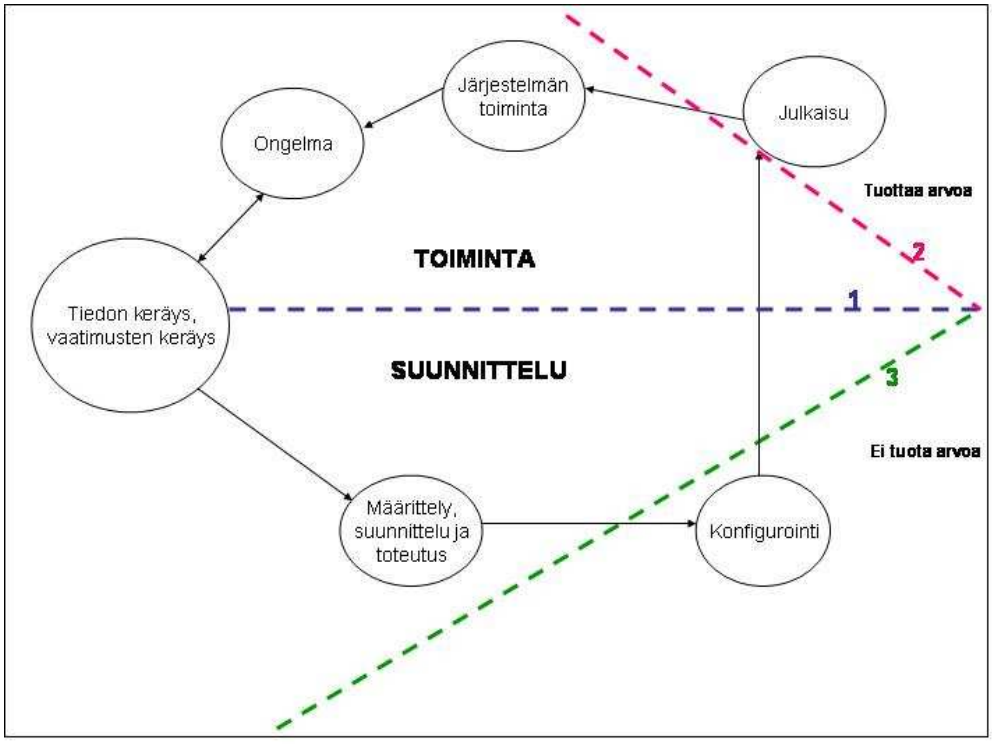
Työnkulun kartoitus on tehokas työkalu, sillä se kiinnittää huomion itse tuotteeseen ja sen arvoon asiakkaalle jättäen teknologian, organisaation rakenteen, kustannusrakenteen ja prosessimallit taka-alalle.

4.3 Yhteenveto

Tietojärjestelmän arvo asiakkaalle voi olla liiketoiminnallisen arvon kasvua, markkinaosuuden kasvua, yrityksen luotettavuuden paranemista, pienempiä riskejä tai turvallisuuden paranemista.

Kirjallisuudesta löytyi kolmenlaisia väitteitä siitä, missä ohjelmistokehityksen työvaiheissa muodostuu arvoa asiakkaalle. Kuvassa 16 esitetään kolmella eri katkoviivalla, missä ohjelmistokehityksen työvaiheissa arvoa muodostuu kirjallisuuden mukaan. Taylorin [Tay86] näkökannan mukaan arvoa muodostuu ongelman kuvaus-,

vaatimusten keräys-, julkaisu- sekä toimintavaiheessa (sininen viiva 1). Middletonin [Mid01] mukaan järjestelmä alkaa tuottaa arvoa asiakkaalle vasta julkaisuvaiheessa (punainen viiva 2). Favaro [Fav02] puolestaan esittää, että arvoa muodostuu kaikissa paitsi konfigurointivaiheessa (vihreä viiva 3).



Kuva 16: Kolme eri katsantokantaa ohjelmistoprosessin arvoa tuottaviksi asioiksi.

Arvoa tuottamattomia prosessin osia ovat ainakin konfigurointi, monimutkaiset projektin seurantajärjestelmät ja muutosten hyväksyttämisprosessit. Julkaisujen suunnittelu ei suoraan tuota arvoa, mutta vaikuttaa suoraan siihen, että keskeneräistä työtä eli jätettä on mahdollisimman vähän. Tällöin myös julkaisujen suunnittelu vaikuttaa epäsuorasti arvoon. Ohjelmistokehitystiimin jäsenillä sekä asiakkaalla on omia tavoitteitaan, joiden lisäksi kaikkien yhteisenä tavoitteena tulisi olla arvon tuottaminen.

Työnkulun kartoitus on keino tarkastella arvon muodostumista ohjelmistoprosessissa. Analyysin avulla voidaan pyrkiä poistamaan jätettä ohjelmistoprosessissa (esimerkiksi ylimääräisiä odotusaikoja) sekä muodostamaan arvoa tuottavista prosessin vaiheista sulava, jatkuva liike. Järjestelmän toimittamisen nopeudella on asiakkaalle arvoa;

toimittamalla nopeasti järjestelmän tärkeimmän prioriteetin osakokonaisuuksia asiakkaalle saadaan asiakkaalle tuotettua mahdollisimman paljon liiketoiminta-arvoa.

5 Kanban -näkökulma ohjelmistokehittäjän työhön sekä arvon muodostumiseen

Luvussa 2 esiteltiin ohjelmistokehittäjän työn erityispiirteitä riippuen ohjelmistokehityksen prosessimallista. Luvussa 3 puolestaan perehdyttiin Kanban-menetelmään ohjelmistokehityksessä. Tässä luvussa muodostetaan kirjallisuudesta löydetyistä ohjelmistokehittäjän työn erityispiirteistä teoreettinen malli, johon liitetään Kanban-näkökulma sekä vertailun vuoksi vesiputousmallin näkökulma. Tarkoituksena on osoittaa, kuinka ohjelmistokehittäjän työn erityispiirteet vaihtelevat prosessimallista riippuen.

Luvussa 4 tutkittiin, mitä arvo tarkoittaa kirjallisuudessa. Tässä luvussa esitellään, miten kirjallisuuden perusteella Kanban-menetelmää käyttäen voidaan vaikuttaa arvon muodostumiseen asiakkaalle. Lopuksi kuvataan, kuinka Kanban voi auttaa työnkulun kartoituksessa ja sitä kautta jätteen poistamisessa.

5.1 Malli ohjelmistokehitysmenetelmien kvalitatiivisista vaikutuksista ohjelmistokehittäjän työhön

Erilaisissa ohjelmistokehityksen prosessimalleissa tuotetaan eri määrä **dokumentaatiota**. Poppendieck ja Poppendieck [PoP03] väittävät, että perinteinen vesiputousmalli synnyttää paljon dokumentaatiota. Poppendieck ja Poppendieck [PoP03] esittävät, että Kanban noudattelee Lean-filosofiaa, jossa kaikki lähtee asiakkaasta. Näin ollen, mikäli asiakas ei vaadi dokumentaatiota, ei sitä myöskään tuoteta.

Sommerville [Som00] väittää, että vesiputousmalli lähinnä rohkaisee kätkemään ongelmia. Kun edellisessä vaiheissa muodostuneita ongelmia tai virheitä tulee esiin, tehdään niille tilapäinen korjausratkaisu tai ne jopa jätetään kokonaan ottamatta huomioon. Liker [Lik04] esittää, että Lean-filosofian mukaan, kun ongelma ilmenee, pyritäänkin löytämään sen perimmäinen syy ja korjaamaan se välittömästi ettei ongelma tulevaisuudessa enää esiintyisi. **Ongelman korjaamiseen** varataan runsaasti resursseja

ja jopa koko muu ohjelmistotuotanto keskeytetään siksi aikaa, jotta ongelma saadaan varmasti korjattua. Kanban-menetelmällä ongelmat tulevat näkyviin lähes välittömästi niiden tapahduttua [Lad08] (katso luku 3.6). Tämän jälkeen Lean-filosofian mukaisesti ongelmat ratkaistaan niiden pohjimmaiseen syyhyn asti.

Ohjelmistokehitysprosessissa edistymisen **visualisoiminen** auttaa ohjelmistokehittäjää näkemään, missä ohjelmistokehityksessä ollaan menossa: mitä työtä on jo tehty ja kuinka paljon sitä on vielä jäljellä [PoP03]. Sommerville [Som00] väittää, että vesiputousmalli on visuaalinen, sillä jokaisen vaiheen jälkeen tuotettavasta dokumentaatiosta nähdään kuinka paljon työtä on tehty sekä kuinka paljon sitä on vielä jäljellä. Poppendieck ja Poppendieck esittävät, että myös Kanban-menetelmä on visuaalinen, sillä Kanbanissa työnkulku on visualisoitu seinätaululle. Seinätaululta voidaan nähdä työtehtävien edistyminen, sekä myös kokonaisuuden edistyminen seuraamalla tehtäviä kuvaavien lappujen siirtymistä työlista-sarakkeesta valmis-sarakkeeseen [PoP03].

Ohjelmiston **kokonaisuuden hallinta** on yksinkertaisempaa vesiputousmallissa kuin ketterissä malleissa [Kom04]. Myöskään Kanban-menetelmä ei tarjoa kokonaisuuden hallintaan työkaluja [PoP03]. Kanbaniin avulla voidaan nähdä mitä työtä täytyy seuraavaksi tehdä ja mitä ollaan jo tehty, samoin se, mitä työtä on meneillään. Kuitenkaan Kanban ei kerro kokonaisprosessista, esimerkiksi sitä, kuinka paljon työtä on kaikenkaikkiaan vielä jäljellä [PoP03].

Vesiputousmallissa ei juuri rohkaista **kommunikointiin**, vaan yhden vaiheen päätyttyä tuotetaan seuraavalle vaiheelle dokumentaatiot, jota toteuttavat henkilöt seuraavat [PoP03]. Kanbanissa Lean-filosofian mukaisesti tiedonkulku on suoraa ja avointa [KaL94]. Kommunikointia pidetään hyvänä asiana ja esteet tiedonkulussa pyritään poistamaan.

Poppendieck ja Poppendieck [PoP03] väittävät, että mikäli prosessimalli ei ole ohjelmistokehittäjän kannalta ymmärrettävä eikä sitä ole helppo **omaksua**, ovat ohjelmistokehittäjät tehneet työtään omalla tavallaan, prosessimallista riippumatta. Abrahamsson [Abr03] osoittaa, kuinka ketterissä menetelmissä ensimmäinen iteraatio on ohjelmistokehittäjille lähinnä oppimisprosessi eikä ole yhtä tuottava kuin myöhemmät iteraatiot. Poppendieck ja Poppendieck [PoP03] esittävät, että Kanban-

menetelmä on hyvin intuitiivinen ja antaa melko vapaat kädet ohjelmistokehittäjille.

Vesiputousmallissa **palautetta** on sallittua antaa vain eri vaiheiden välissä [Som00]. Palautteen katsotaan vaarantavan ennalta määrätyn suunnitelman toteutumisen [PoP03]. Lean-periaatteen mukaisesti Kanbanissa ohjelmistokehittäjien tulisi toimia yhteistyössä asiakkaan kanssa siten, että asiakkaan palautetta saadaan säännöllisesti [PoP03]. Ohjelmakoodin kirjoittamisen jälkeen tulisi olla välittömästi mahdollisuus antaa ohjelmistokehittäjälle palautetta siitä, toimiiko ohjelma siten, kuin asiakas on toivonut. Ongelmatilanteissa nopea palaute asiakkaalta on erityisen olennaista [PoP03].

Vesiputousmallissa jokaisen vaiheen jälkeen dokumentaatio **hyväksytetään** ylemmällä taholla ennen seuraavaan vaiheeseen siirtymistä [Kom04]. Poppendieck ja Poppendieck [PoP03] esittävät, että Lean-filosofian periaatteen mukaan paras asiantuntemus on kullakin työpisteellä. Käskyjen tai ohjeiden kulkemiselle edestakaisin organisaation hierarkiassa ole aikaa saati tarvetta. Kussakin työpisteessä asiantuntijoilla on valta tehdä suunnittelupäätöksiä.

Abrahamsson ja kumppanit [AbS04] väittävät, että vesiputousmallissa ohjelmistokehittäjillä ei ole välttämättä laaja-alaista tietämystä sovellusalueesta, vaan ainoastaan työtehtävien osalta tarpeelliset tiedot. Vesiputousmalli ei kuitenkaan ota kantaa siihen, kuinka **työtehtävät valitaan**. Kanbanissa ohjelmistokehittäjät voivat ohjata omaa työtään Lean-filosofian mukaisesti, ja heille annetaan mahdollisuus valita itse seuraavaksi toteuttamansa työtehtävä tehtävien asioiden alueelta. Mikäli ohjelmistokehittäjille määrätään käytettävät työkalut ylhäältä päin eivätkä he saa itse vaikuttaa niiden valintaan, voi ohjelmistoprojektissa tulla ongelmia [Sin89]. On tärkää kuunnella ohjelmistokehittäjien omia mielipiteitä myös työkaluista [Sin89]. Lean-filosofiassa työntekijät saavat itse ehdottaa tarvitsemiaan työkaluja. Jotta työ voisi olla itseohjautuvaa on kaikkien tiedettävä mitä on meneillään, mitä vielä puuttuu, mitä ongelmia esiintyy ja miten ollaan edistytty [PoP03]. Kanban-taulun avulla kaikki voivat nähdä mitä vielä puuttuu, mitä ollaan jo tehty ja kuka on tekemässä mitäkin asiaa [PoP03].

Taulukossa 1 esitetään kuinka kirjallisuustutkimuksen perusteella vesiputousmalli ja toisaalta Kanban ohjaavat ohjelmistokehittäjän toimintaa.

Työhön vaikuttavia seikkoja	Käsittely vesiputous-mallissa	Käsittely Kanbanissa
Dokumentaation määrä	Jokaisen vaiheen jälkeen runsaasti hyväksytettäviä dokumentteja.	Dokumentaatiota on sen veraan kuin asiakas tarvitsee tai on ohjelmistokehittäjien työskentelyn kannalta välttämätöntä.
Ongelmanratkaisu	Ongelmat tulevat esiin vasta myöhäisemmässä ohjelmistotuotannon vaiheessa. Niille tehdään yleensä kiertotie.	Ongelmat tulevat näkyviin välittömästi seinätaululla. Ongelma pyritään ratkaisemaan Lean-filosofian mukaisesti perussyyhyn asti ja poistamaan pohjimmainen syy. Ongelmat ratkaistaan heti.
Visualisoiminen	Kattava dokumentaatio visualisoi eri vaiheet.	Työnkulku on visualisoitu seinätaululle.
Kokonaisuuden hallinta	Kokonaisuus tiedossa ensimmäisen vaiheen jälkeen (dokumentaatio).	Kanban noudattaa Lean-filosofiaa, jossa pyritään ymmärtämään kokonaisuus. Kuitenkaan Kanban-taulu ei tarjoa selkeää ohjelmistokokonaisuuden hallinnan työkalua.
Kommunikoinnin määrä	Ei juuri kommunikointia. Vaiheiden välillä kommunikointi voi olla jopa kielletty.	Lean-filosofian mukaisesti kommunikointiin pyritään ja kannustetaan.
Menetelmän ymmärtäminen ja omaksuminen	Ohjelmistokehittäjien sopeuduttava prosessiin. Kehittäjät eivät kuitenkaan ole omaksuneet prosessia, vaan ovat tehneet työtään omalla tavallaan, riippumatta prosessista.	Intuitiivinen menetelmä, joka antaa hyvin vapaat kädet ohjelmistokehittäjälle. Ainoa sääntö Kanbanissa on se, että työnkulku on visualisoitava.
Palautteen määrä	Vain kunkin vaiheen välissä palautteen aika.	Lean-filosofian mukaisesti palautteeseen kannustetaan. Asiakkaan kanssa järjestetään usein tilaisuuksia palautteen antamiseen.
Asioiden hyväksyttämisprosessi	Dokumentaatio hyväksytetään kunkin vaiheen jälkeen.	Paras asiantuntemus on kussakin työpisteessä. Ei monimutkaisia hyväksyttämisprosesseja.
Työtehtävien valinta	Vesiputousmalli ei ota kantaa työtehtävien valintaan.	Työntekijät voivat itse valita työtehtävänsä.

Taulukko 1: Vertailu vesiputousmallin ja Kanbanin välillä vaikutuksista ohjelmistokehittäjän työhön.

5.2 Arvon muodostuminen Kanban-ohjelmistokehityksessä

Luvussa 4 esiteltiin ohjelmistokehtityksen asiakkaalle arvoa tuottavat työvaiheet. Arvoa muodostavia työvaiheita ovat vaatimusten keräys, määrittely, suunnittelu, toteutus, julkaisu sekä järjestelmän toimintavaihe. Myös nopea toimitus sekä priorisointi tuottavat arvoa, kun asiakas saa toimivaan järjestelmään haluamiaan piirteitä mahdollisimman pian. Poppendieck ja Poppendieck [PoP03] väittävät, että Kanban-menetelmän avulla pystytään vaikuttamaan toimituksen nopeuteen virtaviivaistamalla ohjelmistotuotantoprosessia.

Kanbanin toisen perusperiaatteen mukaan kullekin työvaiheelle on määritelty enimmäismäärät sille, kuinka paljon työvaiheessa voi olla samanaikaisia tehtäviä [KnS10]. Ladaksen [Lad08] mukaan tämän perusperiaatteen avulla tehtävien virtaus läpi työnkulun pysyy tasaisena ja mahdollisimman nopeana. Myös Kniberg ja Skarin [KnS10] esittävät, että myös Kanbanin kolmas perusperiaate, mittaaminen läpimenoajan lyhentämiseksi ja prosessin optimoimiseksi, auttaa viemään tehtävät Kanban-taulun läpi valmiiksi piirteiksi mahdollisimman nopeasti. Näillä on arvoa asiakkaalle, sillä ne ilmenevät toimituksen nopeutena.

Kanbanin ensimmäiseen perusperiaatteeseen, visualisointiin, kuuluu seinätaulun käytön ohella asiakkaan visioiden pilkkominen tehtäviksi sekä priorisointi [KnS10]. Poppendieck ja Poppendieck [PoP03] väittävät, että priorisointi tuottaa arvoa asiakkaalle, sillä sen avulla asiakas saa eniten haluamiaan piirteitä. Nämä piirteet toteutetaan ensimmäisenä, jolloin asiakas saa arvoa nopeimmin.

Usein ohjelmistotuotantoprosesseissa käytetään monimutkaisia hyväksyttämisprosesseja sekä projektien seurantajärjestelmiä [PoP03]. Näillä ei kuitenkaan ole arvoa asiakkaalle [PoP03]. Kanban noudattaa Lean-filosofiaa, kuten luvussa 3 todettiin. Muutokset tulisi hyväksyä nopeasti; Lean-periaatteen mukaan paras asiantuntemus on kussakin työpisteessä ja asiakkaan kanssa kommunikoidaan jatkuvasti. Lean-filosofian mukaan monimutkaisia hyväksyttämisprosesseja sekä projektien seurantajärjestelmiä ei käytetä ohjelmistotuotantoprosessissa [PoP03]. Näin Kanban-menetelmää käyttämällä prosessista karsiutuu pois osia, joilla ei ole arvoa asiakkaalle. Just-in-Time -pohjaisella

Kanban-menetelmällä työtehtävät virtaavat niin nopeasti, että tarkka seuranta on tarpeetonta. Jos projektinseurantaa varten on hankittu monimutkainen järjestelmä, on projektissa todennäköisesti monenlaisia jätteeksi luokiteltavia toimintoja [PoP03]. Tällaisen järjestelmän hankkimisen sijaan tulisi huolehtia tehtävien nopeasta ja tasaisesta virtauksesta.

Jätteen poistaminen on yksi Lean-filosofian olennaisista näkökulmista, kuten luvussa 3 esitettiin. Kanban perustuu Lean-filosofiaan, joten myös Kanbanissa pyritään eliminoimaan jäte. Työnkulun kartoitus on keino tutkia arvoa [PoP03]. Kun käytetään Kanban-menetelmää voidaan tehdä työnkulun kartoitus yhden piirteen tai käyttäjäkertomuksen toimittamisesta asiakkaalle. Jokainen Kanban-taulun työnkulun askel on yksi kartan vaihe. Tämän jälkeen tarkastellaan muodostettua kartoitusta arvon sekä jätteen muodostumisen kannalta. Esimerkiksi kiinnitetään huomiota siihen, onko paljon odotusaikoja vaiheiden välillä tai joudutaanko odottamaan asiakkaan kommentteja johonkin asiaan. Arvoa tuottamattomat askeleet pyritään poistamaan prosessista. Näin Kanbanin avulla voidaan ohjelmistotuotantoprosessi virtaviivaistaa käyttämällä työnkulun kartoitusta. Tämä vaikuttaa suoraan asiakkaalle tuotettavaan arvoon [PoP03].

6 Empiiriset tulokset

Tämän luvun tavoitteena on tarjota Kanban-mentelmään kohdistuvan emipiirisen tutkimuksen tulokset. Tutkimusta tehtiin kahdesta näkökulmasta:

- kvalitatiiviset vaikutukset ohjelmistokehittäjien työhön ja
- arvon muodostuminen Kanban-ohjelmistokehityksessä.

Kirjallisuustutkimuksen perusteella Kanban-mentelmällä on kvalitatiivisia vaikutuksia ohjelmistokehittäjän työhön, kun vertailukohtana käytetään vesiputousmallia (katso luku 5.1). Kirjallisuudesta valittiin luvussa 2.3 yhdeksän asiaa, joilla ohjelmistokehitysmenetelmä voi vaikuttaa ohjelmistokehittäjän työhön. Näistä muodostettiin teoreettinen malli, joka esitettiin taulukossa 1. Tapaustutkimuksen avulla teoreettista mallia validoidaan empiirisesti; kuinka Kanban vaikuttaa näihin valittuihin yhdeksään asiaan.

Kirjallisuustutkimuksen perusteella ohjelmistokehitysprosessissa arvoa tuottavia osia ovat tiedon ja vaatimusten keräys, suunnittelu ja toteutus, julkaisu sekä toimintavaihe. Myös toimittamisen nopeudella on asiakkaalle arvoa; tärkeimmän prioriteetin valmiit osakokonaisuudet mahdollisimman nopeasti tuottavat asiakkaalle liiketoiminta-arvoa. Tapaustutkimuksessa tutkittiin, kuinka käyttämällä Kanban-ohjelmistokehitysmenetelmää voidaan vaikuttaa arvon tuottamiseen.

Tämä luku koostuu tapaustutkimuksen kohteen kuvailusta, tiedonkeräämisen välineiden esittelystä sekä tutkimustulosten esittämisestä. Lopuksi muodostetaan yhteenveto tutkimustuloksista.

6.1 Tapaustutkimuksen kohteen esittely

Ohjelmistoprojektitiimiin kuului 14 henkilöä. 13 henkilöä oli tietojenkäsittelyalan opiskelijoita, yksi ulkopuolinen asiantuntija. Tiimin 13 jäsentä toimi projektissa ohjelmistokehittäjinä, asiantuntija kävi paikalla päivittäin lyhyesti auttamassa opiskelijoita. Tiimi oli jakautunut kolmeen pöytätiimiin siten, että kussakin pöydässä työskenteli yksi seniori, jolle käytettävät työkalut olivat tuttuja ja jolla oli enemmän kokemusta tietojärjestelmien tekemisestä, ja kolme vähemmän kokenutta opiskelijaa. Lisäksi kaksi henkilöä toimi tiimissä pöytäryhmien ulkopuolella, ohjaten ja johtaen ohjelmistokehittäjiä. Asiakkaan edustajana oli kaksi henkilöä.

Projektiryhmän tavoitteena oli tuottaa uusi, internetpohjainen rahoittamisen ja sijoittamisen järjestelmä. Projektin kesto oli seitsemän viikkoa.

6.2 Tiedonkeräämisen välineet

Tutkimusta tehtiin havainnointitutkimuksena ilman osallistumista sekä puolistrukturoituna haastattelututkimuksena. Havainnointia suoritettiin kahdeksan päivän aikana, yhteensä 30 tuntia. Havainnointi tapahtui ilman osallistumista, tilassa, jossa kaikki projektin jäsenet työskentelevät. Havainnoinnissa oltiin paikalla normaalissa työskentelytilanteessa 22 tuntia, asiakasesittelyssä 7 tuntia ja tiimin katselmuksessa 1 tunti. Havainnoinnista kirjoitettiin havainnointipäiväkirja, jota muodostui 16 sivua [Pir10a].

Haastattelututkimus suoritettiin haastattelemalla kuutta ohjelmistokehitystiimin jäsentä välittömästi projektin päätyttyä. Haastateltavat valittiin roolituksen perusteella: yksi heistä oli tiimin johtaja, hänen lisäkseen kaksi oli kokeneempaa ohjelmistokehittäjää, senioria, joilla oli aikaisempaa kokemusta ohjelmistokehityksestä yliopistokurssien ulkopuolelta, ja kolme oli junioria. Haastattelut kestivät noin tunnin kunkin henkilön osalta. Keskustelut nauhoitettiin ja tallenteet litteroitiin sanasta sanaan analyysin tekemistä varten. Litteroitua aineistoa analyysia varten muodostui 84 sivua [Pir10b]. Tässä luvussa siteerataan sekä tutkimuspäiväkirjaa [Pir10a] että ohjelmistokehittäjien vastauksia teemahaastattelujen kysymyksiin [Pir10b]. Tunnistamista varten kehittäjät on numeroitu K1 (kehittäjä 1) – K6 (kehittäjä 6). Kehittäjä1 on tiimin johtaja.

6.3 Projektiryhmän työskentelytavan kuvaus

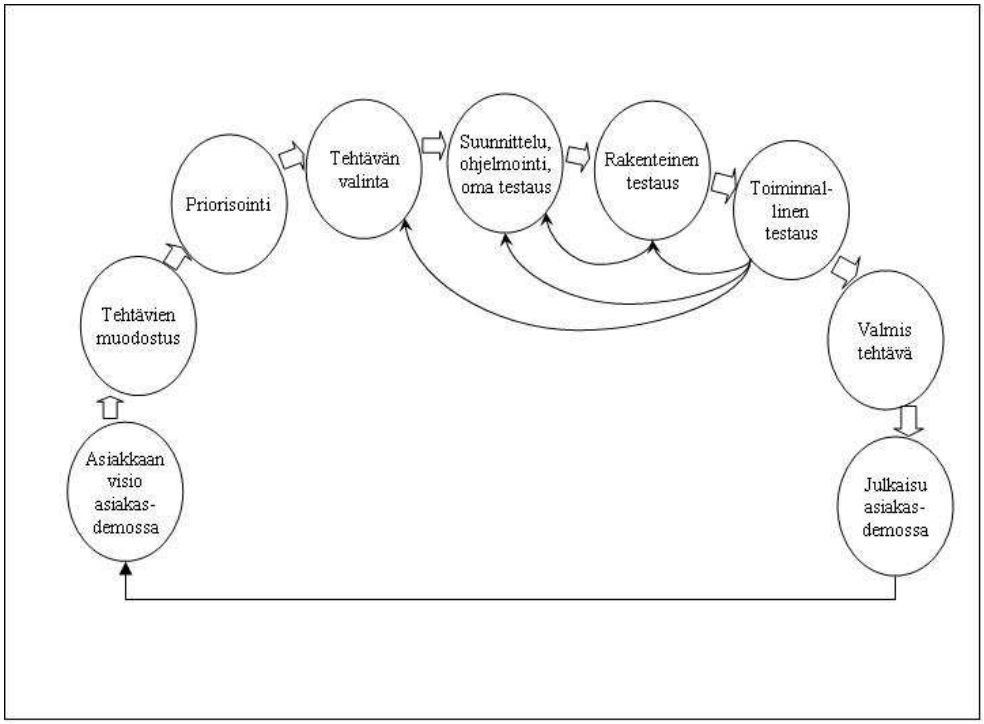
Tämä kappale perustuu havainnointitutkimukseen [Pir10a]. Kappaleessa kuvataan kuinka projektissa tehtiin työtä havainnoinnin mukaan.

Projektissa käytettiin ohjelmistokehitysmenetelmänä Kanban-menetelmää. Prosessiin oli yhdistetty piirteitä ketterästä ohjelmistokehitysmenetelmästä, Scrumista. Kanban-menetelmän mukaisesti työ oli visualisoitu seinätaululle. Scrum-menetelmästä käyttöön oli otettu joka-aamuinen päivittäistapaaminen, jossa jokainen kertoi vastauksen kolmeen kysymykseen (mitä olet tehnyt eilen, mitä aiot tehdä tänään ja onko ongelmia jonkin asian kanssa). Kirjallisuuden perusteella Kanbanissa ei välttämättä ole iteraatiota, mutta projektiin oli ketteristä menetelmistä otettu käyttöön iteraatio, joka projektissa oli noin viikon mittainen.

Kerran viikossa pidettiin asiakastapaaminen, demo, jossa asiakas kertoi tarpeistaan ja vaatimuksistaan, asiakkaalle demonstroitiin tuotannon sen hetkistä tilaa ja määriteltiin, mihin asiakkaan vaatimuksiin keskitytään seuraavaan demoon mennessä. Asiakastapaamisen jälkeen pidettiin jokaviikkoinen toteutusvaiheen katselmus, retrospektiivi, jossa jokaisen tuli kertoa mitä on mennyt hyvin sekä mitä on mennyt huonosti; tilaisuudessa asioista keskusteltiin avoimesti.

Kuvassa 17 esitetään projektiryhmän työn malli, se, minkälainen heidän työskentelyprosessinsa oli havainnoinnin perusteella. Työskentelyn lähtökohtana olivat asiakasdemot, joissa asiakas esitti vaatimuksiaan sekä toiveitaan toteutettavasta

järjestelmästä. Asiakkaan toiveet pilkottiin tehtäviksi, usein jo asiakaspalaverin aikana. Tehtäville asetettiin prioriteetit tehtävienhallintajärjestelmään. Ohjelmistokehittäjät ottivat itselleen tehtäviä tehtävienhallintajärjestelmästä ja toteuttivat ne. Kun tehtävä oli valmis ohjelmistokehittäjän osalta, hän pyysi jotakuta muuta henkilöä tarkastamaan ohjelmakoodin. Tämä tarkastus koostui rakenteisesta testauksesta (engl. white-box testing), jossa varmistettiin, että ohjelmakoodi on käytettävän ohjelmointikielen käytäntöjen mukaista, ovatko metodien nimet järkeviä tai onko turhia kommentteja. Kun rakenteinen testaus oli läpäisty, tehtiin tehtävälle toiminnallinen testaus (engl. black-box testing), jossa tehtävää testattiin loppukäyttäjänä kokeillen vapaasti eri toimintoja. Kun tehtävä oli läpäissyt laadunvarmistukset, oli se valmis esiteltäväksi asiakaspalaverissa.



Kuva 17: Projektiryhmän työskentelyn malli [Pir10a].

Kuvassa 18 esitetään projektiryhmän Kanban-seinätaulu, jossa olivat sarakkeet: Assigned, Blocked, Working, Code Review, in QA (quality assurance) ja Verified. Lisäksi oikealla oli Backlog-tila, jossa oli uusia tehtäviä, ei kuitenkaan kaikki tehtävät, sekä erillinen alue kaikkein tärkeimmille tehtäville. Tilassa Assigned joku oli valinnut tehtävän suorittaakseen, mutta ei vielä ollut ryhtynyt tekemään sitä.

> ”Assigned: kukin ottaa jonkun tehtävän systeemistä, track systeemistä suunnilleen minkä haluaa.” [Pir10a]

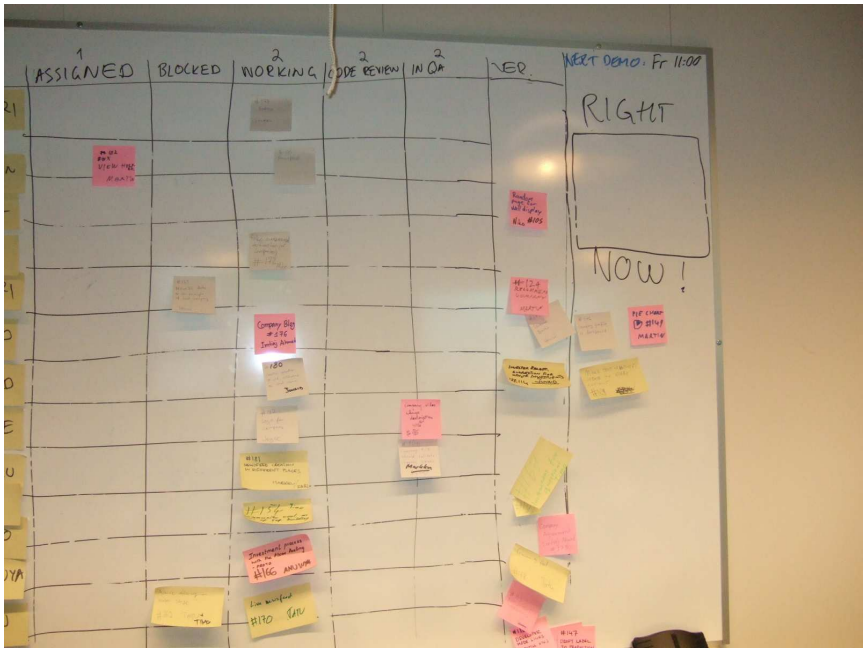
Kun henkilö ryhtyi tekemään työtehtävää, hän siirsi lapun sarakkeeseen Working. Jos tehtävän suorittamiselle oli jotakin esteitä, hän siirsi tehtävän sarakkeeseen Blocked.

> ”Blocked: tänne siirtyy Working-sarakkeesta, jos jokin tehtävä odottaa ulkopuolista juttua. Esim IT-osaston konfigurointi- ja implementointiapua. Testiserverin pystyttämistehtävä oli täällä, kun IT-osaston toimenpiteitä odotettiin.” [Pir10a]

Tehtävän valmistuttua hän etsi seniorin tekemään rakenteisen tarkastuksen, jolloin tehtävä siirtyi sarakkeeseen Code Review. Kun rakenteinen testaus oli läpäisty hyväksytysti, siirtyi tehtävä sarakkeseen in QA ja tehtävän tekijä etsi sille toiminnallisen testauksen tekijän. Kun testit oli läpäisty, siirtyi tehtävä sarakkeeseen Verified eli valmis, jonka jälkeen tehtävä odotti esittelemistä asiakkaalle.

Testausvaiheista tehtävä saattoi myös palata takaisin aikaisempiin vaiheisiin, jos vikoja tai ongelmia tuli esiin, kuten kuvassa 17 esitetään:

> ”Paljon laput seilaavat edestakaisin Working – CodeReview - inQA välillä.” [Pir10a]



Kuva 18: Projektiryhmän Kanban-taulu projektin kolmannella viikolla [Pir10a].

Kanbanin tehtävien hallintaan projektiryhmällä oli käytössään seinätaulun lisäksi myös tehtävienhallintajärjestelmä. Järjestelmässä tehtävillä oli samat tilat kuin seinätaululla, jossa tilat oli esitetty eri sarakkeissa. Tehtävienhallintajärjestelmässä kustakin tehtävästä oli lisätietoa tekijän ja työn vaiheen ohella, esimerkiksi tehtävän muutoshistoria sekä tieto siitä kuka on viimeksi muokannut tehtävää. Työlistalla olevat tehtävät priorisoitiin. Tehtävien tila muutettiin järjestelmään samalla, kun lappua siirrettiin seinätaululla sarakkeesta toiseen.

> ”Track system, jonne kaikki laitetaan. Backlog on myös Trackissa ja kun otetaan tehtäväksi, tehdään lappu.” [Pir10a]

6.4 Tutkimustulokset Kanbanin vaikutuksista ohjelmistokehittäjän työhön

Tapaustutkimuksen perusteella Kanbanilla on vaikutuksia ohjelmistokehittäjän työhön. Kirjallisuustutkimuksen perusteella valittuihin yhdeksään seikkaan, joihin ohjelmistokehitysmenetelmällä on vaikutusta, saatiin seuraavassa kuvattuja tutkimustuloksia. Lainausmerkeissä olevat lauseet ovat haastatteluista [Pir10b], mikäli lähdettä ei ole mainittu. Näitä lainauksia on enemmistö. Jos lause on havainnointipäiväkirjasta [Pir10a], on lause yksinkertaisissa lainausmerkeissä ja lähde mainittu lauseen lopussa.

Dokumentaatiota projektissa tehtiin vain sen verran, mitä oli tiimin työskentelyn kannalta tarpeen. Kaikessa työskentelyssä lähdettiin Kanban-periaatteen mukaisesti asiakkaan tarpeesta, eikä asiakas ilmaissut tarvetta saada dokumentaatiota:

> ”Ei nyt asiakas olis ollut kiinnostunutkaan lukeen mitään läpyskoitä, ja niilläkin oli kiire koko ajan, et ei niillä ollut aikaa käydä kuin vähän siellä demossa sanomassa mielipidettä. Ei ollut mitään tarvetta tehdä mitään dokumentaatiota.” [K2]

Syntyneeksi dokumentaatioksi ohjelmistokehittäjät näkivät asiakasdemojen muistiinpanot, ohjelmointikielen (Ruby on Rails) kommentointikäytännöt, projektin

aikana syntyneet kuvat seinätaululle, esimerkiksi näyttöjen prototyypit, jotka talletettiin tietokantaan, ja tehtävät tehtävienhallintajärjestelmässä. Tiimillä oli myös sisäinen wiki, jossa tiimin itse kokoamia yleisiä ohjeita, esimerkiksi ohjeistuksia standardeihin, käytettävien sanastojen kuvauksia sekä tieto siitä, mikä tässä projektissa tarkoittaa valmista osakokonaisuutta (definition of done). Asiakasdemojen muistiinpanot olivat arvokasta dokumentaatiota, koska niistä tehtiin suoraan tehtäviä tehtävienhallintajärjestelmään:

> ”Ehkä se on dokumentaatiota, että me niistä asiakasdemoista tehtiin muistiinpanot**.** Muistiinpanot laitettiin siihen valkoiselle kankaalle ja ruvettiin käymään niitä läpi ja sitten tajusi, että tässähän ne ovat, mitä niitä käymään, kaikki olivat demossa paikalla. Dokumentaation tekeminen oli arvokasta, koska suoraan niistä muistiinpanoista tehtiin tikettejä. Ne oli hyvin tähän otettu ylös ne asiat, sen kun vaan tehdään tikettejä näistä!” [K1]

Ongelmanratkaisun osalta sekä havainnoinnissa että haastatteluissa tuli ilmi, että ongelmat pyrittiin Lean-periaatteiden mukaisesti ratkaisemaan heti:

> ’Yksi työntekijä tuli paikalle aamulla klo 8.45 ja kysyi miksi testit feilaavat? Ja heti toinen alkoi auttaa häntä’ [Pir10a]

> ”Minulle oli hyvin luontevaa kysyä naapureilta apua. Ihmiset oppivat siellä muutenkin, että apua saa, kun kysyy ja kun sitä uskaltaa kysyä.” [K1]

> ”Se rupesi olemaan silleen, että heti kun jengi törmäsi seinään, niin ne pyysivät kaveria vierestä. Et se toimi hirveen hyvin.” [K4]

> ”Minulla ainakin oli siinä alkuajatuksena, että tehdään nyt semmoisella Lean-ajattelulla, missä heti yritetään kohdata ne ongelmat ja poistetaan ne. Jos on vaikka joku luokka, joka on huonosti koodattu, niin koodataan se nyt kerralla kuntoon.” [K1]

Ongelmat ratkaistiin lähinnä oman tiimin kesken, kysymällä apua toisilta. Tässä auttoi se, että kaikki olivat samassa tilassa, jolloin avun kysyminen oli helppoa. Myös Lean-filosofian mukainen kannustaminen kommunikointiin auttoi asiaa, eli apua sai heti, jos vain uskalsi kysyä.

> ”No, siinä meitä oli 13 henkeä noin pienessä tilassa, niin ei tarvitse kun vähän

huudella siinä, niin on hyvin mahdollista että joku tietää.” [K5]

Jos ohjelmistokehittäjä tarvitsi apua pidemmäksi ajaksi, vapaana olevan henkilön löytäminen oli Kanbanin avulla helppoa:

”How did you know that somebody was free?”

“Just look at the Kanban-board, and at the moment he has no task.” [K6]

Ongelmat myös tulivat nopeasti esiin Kanbanin ansiosta:

”Tulee mieleen se yksi tapaus, missä kehittäjällä oli kaksi lappua ja Working:ssä limitti oli kaksi, mutta hän oli ottamassa kolmatta, ja myös minulle sitten ensimmäistä kertaa aukesi, että okei hei, tässä nyt on wip:n pointti, että ne tukokset pitää saada pois. Vaikka ne tuntuvat vähän inhottavilta, ne pitää jollain tavalla ottaa pois. Että ei voi ottaa enempää.” [K1]

Jos ongelma oli sellainen, jota ei tiimin kesken saatu ratkaistua, kysyttiin apua ulkopuoliselta asiantuntijalta. Ongelmanratkaisussa ei kuitenkaan havainnoinnin perusteella kysytty aina viittä kysymystä perussyyn löytämiseksi. Esimerkiksi, kun tiimillä oli ongelma testipalvelimen pystytyksen kanssa, odotettiin, että ulkopuolinen it-osasto sai testipalvelimen pystyyn, mutta ei kysytty viittä kysymystä, koska palvelimen pystytys oli kertaluonteinen asia [Pir10a].

Työprosessin **visualisoimisen** Kanbanin avulla ohjelmistokehittäjät kokivat pääosin positiivisena. Tiimin johtajan mielestä visualisoimisesta oli hyötyä työn jakamisessa sekä ongelmakohtien löytämisessä:

”Minulle ainakin tärkein oli, että vaikkapa aamulla kun tuli töihin, niin ei mennyt koneelle vaan käveli siihen ja katsoi, niin sai sen käsityksen siitä, että mitä kukin tekee. Sitten vielä kun sitä täydensi sen aamutapaamisen informaatiolla, niin sitten oli käsitys siitä, että mitä tiimillä on tällä hetkellä meneillään, ja että jos on vaikka odottamassa jotain, mitä pitäisi saada tehtyä, niin että kuka sitä voisi tehdä ja näin.” [K1]

Myös ohjelmistokehittäjien mielestä visualisoiminen oli tärkeätä. Visualisoiminen toimi kommunikoinnin apuvälineenä, esimerkiksi tehtävien siirtymisessä sarakkeesta seuraavaan:

> ”Se taulu toimii hyvin. Jos on tietojärjestelmä, niin sen saa piiloon, niin silloin se ei kommunikoi.**”** [K4]

> ”When I asked someone to do the Code Review, and maybe he was busy, so I could just put my ticket to the Kanban board in the Code Review block, and later when he was free, he could see that oh, there is a ticket from me.” [K6]

> ”Tarvii vaan tietää että ihmisillä on jotakin meneillään, ei tarvitse tietää ihan mitä, kun taustalla on se oletus siitä prioriteettijärjestyksestä, ja ihmiset tekevät listaa alaspäin, niin se takaa, että jos siinä on lappuja niin ne ovat hyviä lappuja.. siinä ei tarvitse manageroida, ja se on osa sitä ketterää luottamusta.” [K4]

Visualisoiminen tarjosi myös motivaatiohyötyä. Tämä liittyy siihen, että taululla oli myös sarake valmiille piirteille, jota seuraamalla ohjelmistokehittäjät saattoivat nähdä edistymistä tapahtuneen.

> ”Minusta se tarjoaa motivaatiohyötyä siinä. Tai minusta ei henkilökohtaisesti se nyt ei ollut niin motivoivaa liikutella niitä lappuja siinä, mutta kun viikko kului, niin se rupesi se Verified–sarake siinä täyttymään, niin se motivoi.” [K2]

> **”**Jos taulu on tyhjä, niin sitten joku voi herätä ja lähteä kyselemään muilta, et mitä te nyt oikeasti teette, että teillä on nyt tyhjää siellä. Sitten taas jos liput kasaantuvat, niin siitä näkee selkeästi että nyt joku tarvitsee apua. Sitten meillä on se viimenen vaihe, eli se verifioitu-vaihe, se oli hirveen motivoivaa, kun niitä lappuja alkoi kerääntyä sinne.” [K4]

> ”It motivated me to work faster. Maybe we had a lot of tasks in the Verified section, and one single ticket on the Kanban board, which was my task. So I would look and see..’oh everyone has finished the tasks..’ and you are in the working section, so go on and do it. So that was a motivating thing.” [K3]

Myös tekeillä olevan työn rajoittamisessa, joka on yksi Kanbanin kolmesta perusperiaatteesta, visualisoiminen auttoi ymmärtämään prosessia:

> ”Siinä oli selkee visualisointi siitä, että jokaisella on oma linjastonsa ja siinä voi olla kaksi hommaa niinku kerralla tai noin.” [K5]

Yksittäisten tehtävien liikkumista ohjelmointikehittäjät eivät kuitenkaan välttämättä

seuraa:

> ”Niin no, niin miten sitä nyt sitten mitäkin tikettiä tuli sit seurattua minkäkin verran.” [K5]

Lean-periaatteen mukaan on oltava jatkuva pyrkimys **kokonaisuuden ja markkinaympäristön ymmärtämiseen**. Tähän tiimissä pyrittiin siten, että ohjelmistokehittäjätkin selvittivät kilpailjoita markkinoilla ja lisäksi kaikki olivat läsnä asiakasdemoissa:

> ”The team had to pick up a website, make an analysis of it. Otherwise people come across a set of ideas, but then again, the ideas are the one and the same all the time. Certain things impressed me on those websites. I made a list of them and then I thought that how can I get this to my project. So I had suggestions, and after the approval we thought that ok, this can satisfy the customer.” [K3]

> “I think the best way is to attend the demo. The customer will raise a lot of questions, and you will know what they really want, and you will have a general, a big idea of the project.” [K6]

> “Ensinnäkin kaikken läsnäolo demoissa lisää kokonaisymmärrystä itse ihan tuotteesta, mitä yksittäisillä kehittäjillä ei välttämättä ole. Sitten tietenkin vielä, mikä tärkeempi, se lisää ymmärrystä siitä asiakkaan tarpeesta. Ja sitten samalla sitä kautta se domain-knowledge kasvaa myös, joka on aivan keskeisiä ohjelmistokehityksessä.” [K4]

Lisäksi tiimin sisällä kaikki saivat tehdä kaikkea ja sitä kautta kaikilla oli kokonaiskuva tehtävästä järjestelmästä:

> ”Meillä ei ollut niinkun että yksi tekee yhtä pientä palaa, vaan kaikki voivat tehdä ihan mitä tahansa aluetta. Kaikki näkevät koko ajan sen koko systeemin.” [K2]

Kuitenkaan, haastattelujen perusteella, ohjelmistokehittäjillä ei ole tarvetta välttämättä tietää kokonaisuudesta sitä, mitä muut ovat tekemässä:

> ”Sitten minä huomasin, että minulla ei ollut joka hetki käsitystä että mitä joku toinen tekee tai että missä vaiheessa joku menee, eikä se minun mielestäni ollut tarpeellista tietoa millään tasolla. Siinä kun kuuli kerran päivässä että saiko joku

> jonkun homman valmiiksi, onko siinä jotain esteitä ja sitten että mitä on ajatellut tehdä, siinä pystyi esimerkiksi puuttumaan, että tekisitkö sä mielummin jotakin tällaista, koska tämä on tärkeempää. Se oli periaatteessa kaikki hallinta mitä siinä tarvitsi minun, kehittäjän, tietää.” [K4]

Kommunikointi oli tiimissä haastattelujen perusteella vapaata:

> ”Minä koen, että kommunikointi oli hyvin vapaata, eli ketterää. Kaikki pystyivät kommentoimaan mihin tahansa asiaan, mihin he halusivat.” [K4]

> ”We had a pretty good amount of communication between the team because there were people from bachelors, there were researchers, there were masters. Everyone was new to the project and a new acquaintance with everyone, but still the project helped glowing everyone, we felt we were a single entity. The communication was extremely flexible.” [K3]

Myös havainnoinnissa tuli esiin kommunikaation merkitys:

> ’Yksi tehtävä ollut kaksi päivää Working-sarakkeessa. Nyt tiimissä löydettiin tapa pilkkoa se pienemmiksi. Kommunikaatio tässäkin: jos huomaa jonkin tehtävän liian iso, joku muu voi löytää miten sen saisi pilkottua.’ [Pir10a]

Kanban-menetelmän **omaksuminen** oli ohjelmistokehittäjien mielestä helppoa. Projektin alussa asiakas sekä tiimin johtaja selittivät muulle tiimille lyhyesti Kanbanin periaatteen. Menetelmä oli tiimin mielestä kevyt ja ymmärrettävä:

> ”Kanban oli suhteellisen selvä siinä vaiheessa kun mulle selitettiin se taulu.” [K4]

> ”At the beginning they explained how to use the Kanban board briefly. It was clear to me, and everything can be seen on the Kanban board.” [K6]

Havainnoinnin alussa tuli ilmi, että kaikki tiimin jäsenet eivät olleet sisäistäneet, kuinka Kanban-taulu toimii. Tällöin tiimin johtaja käytti Kanban-taulua koulutukseen prosessin ymmärtämiseksi:

> ’Työntekijä teki jotakin muuta, kuin mitä taululla näytti. Tehtävä on itseasiassa valmis, työntekijän pitäisi antaa lappu eteenpäin, se tehtävä testattavaksi jollekin. Työntekijä oli vain jättänyt lapun taululle, mutta alkanutkin tehdä jotakin muuta. Laput olivat hieman sekaisin taululla heidän osaltaan. Tiimin johtaja vei

> työntekijän taululle ja kertoi: Taulun oltava ajantasalla sen mukaan, mitä ollaan tekemässä. Jos on tehty homma, on etsittävä joku, joka voi tehdä seuraavan sarakkeen homman ja sitten otettava uusi tehtävä systeemistä. Eli Kanban-taulua käytettiin koulutukseen.' [Pir10a]

Kanban-taulun rakennetta ei ole menetelmässä säädelty. Kanban-taulussa tapahtui projektissa huomattavasti evoluutiota [Pir10a]. Projektin alussa projektiryhmä päätti itse taulun rakenteesta:

> ”Kanban toimii äärettömän hyvin.. minä en näkisi miks se ei toimisi.. siinä on vain se kun se on niin väljä, niin siinä pitää proaktiivisesti miettiä se, mitkä ne loogiset työvaiheet ovat, ja mitkä ovat tarpeellisia ja näin pois päin, koska muuten se voi mennä aika käsien heilutteluksi, varsinkin jos ei ole yhtään kokenutta kehittäjää.” [K4]

Jo ensimmäisellä viikolla muutoksia tuli taulun rivijärjestelyissä: määritelläänkö rivit pareittain vai henkilöittäin. Myös sarakkeet muuttuivat hieman. Tehtävät asiat päätettiin ottaa pois taululta kokonaan ja pitää ainoastaan tehtävienhallintajärjestemässä, kun huomattiin, että molempien ylläpito aiheutti ristiriitaisuuksia. Taululle lisättiin sen sijaan kolmannella viikolla sarake 'Right Now', johon kaikkein kiireisimmät tehtävälaput laitettiin. Projektissa taulussa aluksi sarakkeissa ei ollut rajoitteita, toisella viikolla rajoitteet lisättiin:

> 'Sarakkeisiin tuli maksimimäärät, kun eräällä henkilöllä 2 työtä kesken, mutta halusi ottaa yhden nice-to-do homman vielä lisää.' [Pir10a]

Palautetta tuli tapaustutkimuksen perusteella jatkuvasti tiimin kesken ja asiakkaalta noin kerran viikossa asiakastapaamisissa. Tiimin sisällä palautteen paikkoja oli määritelty prosessiin: Code Review, QA sekä retrospektiivit. Lisäksi palautetta tuli jatkuvasti, spontaanisti:

> ”Siellä oli hyvä ilmapiiri, porukat juttelevat keskenään ja jos joku tehnyt jotakin hienosti, niin sai sit pari positiivista sanaa.” [K2]

> ”Kun alkoi miettiä jotain, että onko tämä nyt järkevää tehdä näin, niin siinä sitten heti koputti jonkun olkapäätä, että mitä mieltä sinä nyt olet, onko tässä mitään järkeä ja näyttääkö tämä nyt yhtään hyvältä.” [K5]

> ”Se meidän Code Review oli merkittävä palautteen lähde.” [K1]

Asiakaspalautetta tiimille tuli asiakastapaamisissa, kerran viikossa. Kolmen ohjelmistokehittäjän mielestä asiakkaalta olisi tarvittu palautetta useammin:

> ”Asiakasdemoissa me saimme asiakkaalta näkemystä siihen, että olemmeko me menossa oikeaan suuntaan.. tiimi olisi tarvinnut asiakasta enemmän.. Oli suuri vapaus tehdä mitä haluaa, mutta asiakkaalta olisi tarvittu enemmän palautetta.” [K1]

> ”Demot olivat valitettavasti ainoat hetket, jolloin me pääsimme oikeasti puhumaan asiakkaan kanssa.” [K4]

> ”Esimerkiksi kaikissa projekteissa missä minä olen ollut, niin asiakas saa melkein täyspäiväisesti vastata, jos ei ole paikan päällä, niin sitten ainakin sähköposteihin, et onko tämä nyt hyvä, ja nyt me ollaan ajateltu tälleen ja näin pois päin. Sen sijaan me, kun saatiin se tehtyä, me venattiin siihen demoon.” [K4]

Asioiden **hyväksyttämisprosessi** oli hyvin kevyt. Asiantuntemus oli työpisteellä, eikä ulkopuolisia tai muita hyväksyttämisiä ollut. Yksittäisten tehtävien osalta tiimin kesken varmistettiin laadunvarmistuksella, että tehtävä hyväksytään valmiiksi. Projektitiimin kokeneemmat ohjelmistokehittäjät loivat projektin alussa kullekin työvaiheelle valmismääritykset (engl. definition-of-done), joiden piti olla täyttyneet, jotta tehtävä voi siirtyä seuraavaan vaiheeseen:

> ”Kun se sinun mielestäsi oli valmis, sinä siirsit sen Code Review:n ja hommasit sen arvostelijan siihen, sitten kun hän sanoi, että okei se on ok, niin se siirtyi QA-vaiheeseen, jossa sinun piti hankkia sille QA-tyyppi, ja sitte kun hän sanoo ok, niin sitten se on valmis.” [K4]

Asiakastapaamisissa puolestaan asiakas hyväksyi valmiit piirteet. Jos asiakas halusi valmiisiin piirteisiin muutoksia, ne vietiin tehtäviksi tehtävien asioiden listalle:

> ”Kaikki mitä me demottiin oli periaatteessa valmista, paitsi jos sitten se tuli seuraavalla viikolla korjattavaksi.” [K4]

Tehtävien valinnassa ohjelmistokehittäjät saivat olla itsenäisiä, kunhan priorisointijärjestystä noudatettiin. Kenelläkään ei tarvinnut hyväksyttää, että saako

jonkin tehtävän ottaa:

> ”Meillä oli aika pitkälle se, että itse valittiin, ja sitten tietenkin aina kysyttiin, jos oli vaikka useampi samalla prioriteetillä, että onko näistä joku oleellisesti tärkeämpi, kun niidenkin sisällä on tietenkin joitain riippuvuuksia, joita ei välttämättä näe, että onko joku juttu mikä pitäisi saada tehtyä että saadaan jotkut muut tiketit auki. Mielipidettä kysyttiin paljon, mutta kyllä se aika omalähtöistä oli.” [K4]

Ohjelmistokehittäjillä oli erilaisia omia motiiveja ottaa tehtäviä korkeimman prioriteetin tehtävien joukosta:

> ”Otti sieltä tärkeimmistä tehtävistä jonkun, mikä kiinnosti.” [K1]

> ”Kannattaa aloittaa jostakin, että tämä on helppo, että pääsee sisälle siihen systeemiin. Sitten voi ottaa jotain missä vähän erilaisia juttuja tulee, että oppii koko ajan. Ei tarvitse välttämättä ottaa jotain täysin uutta.” [K2]

> ”Some tasks I’m really interested in, and something was related to my previous tasks so I took it.” [K6]

> “If I pick up a task**,** how can it help another individual, so that it could reduce the massive coding part or something.” [K3]

6.5 Tutkimustulokset Kanbanin vaikutuksista arvon muodostumiseen

Kirjallisuustutkimuksen perusteella ohjelmistokehitysprosessissa arvoa tuottavien osien määrästä on kolmea eri näkökantaa. Tässä tutkimuksessa katsotaan, että arvoa tuottavia ovat tiedon ja vaatimusten keräys, määrittely, suunnittelu ja toteutus, julkaisu sekä tietojärjestelmän toimintavaihe. Myös toimittamisen nopeudella sekä priorisointiprosessilla on asiakkaalle arvoa; tärkeimmän prioriteetin valmiit osakokonaisuudet mahdollisimman nopeasti tuottavat asiakkaalle liiketoiminta-arvoa. Arvoa tuottamatonta eli jätettä ovat sen sijaan järjestelmän konfigurointi, projektin seurantajärjestelmät sekä monimutkaiset muutosten hyväksyttämisprosessit. Konfiguroinnilla tässä tarkoitetaan teknisen ympäristön valmistelemista järjestelmän toimintaa varten.

Arvon määrittelemisen pohjana käytetään kuvassa 17 esitettyä projektitiimin

työskentelyn mallia, josta nähdään kuinka projektissa asiakkaan tarpeista ja vaatimuksista muodostui valmiita toimivia järjestelmän osakokonaisuuksia.

Kaikki projektin työskentelyssä lähti asiakkaan visiosta, asiakkaan tarpeesta. Kanban-menetelmän käyttö vaikutti olennaisesti arvon muodostumiseen, sillä projektiryhmä pyrki tekemään Kanbanin mukaisesti vain asiakkaalle arvokasta työtä. Kirjallisuuden perusteella Kanbanissa ei välttämättä ole iteraatiota, mutta tapaustutkimuksen mukaan ainakin lyhyet iteraatiot sopivat hyvin käytettäväksi Kanban-menetelmän kanssa.

Haastatteluissa ilmeni, että asiakastapaamiset olivat paras keino, jolla projektitiimi sai tietoa siitä, mitkä ovat asiakkaan tarpeet. Tapaamisten aikana asiakkaan kanssa keskusteltiin kasvokkain, ja tämä koettiin erittäin hyödylliseksi tarpeiden selvittämisessä:

> ”Tarpeista ei saa selvää, ellet sä puhu asiakkaan kanssa. Eli ne demot olivat valitettavasti ainoat hetket, jolloin me pääsimme oikeasti oikeasti puhumaan asiakkaan kanssa.” [K4]

Keskustelulla oli arvoa myös siinä mielessä, että projektitiimi pystyi niissä auttamaan asiakasta selventämään omia tarpeitaan:

> ”The customer really needed our help to take the decision should I really require this thing, do I really need it. So at that time, our team had to help him there.” [K3]

Ne järjestelmän piirteet, jotka asiakas haluaa ensimmäisenä sekä mahdollisimman nopeasti, projektitiimi sai selville kysymällä asiakastapaamisen lopuksi pari kaikkein tärkeintä piirrettä:

> ”Minulle se on ihan selvä, että se meni niin, että muutoksia tippui demon aikana, mutta sitten se yhteenvetokysymys, että sanokaa ne kaksi tai kolme asiaa, mihin meidän pitäisi ensi viikolla keskittyä. Siinäpä ne tulivat niin kuin tarjottimella.” [K1]

Asiakkaan haluamat piirteet saattoivat olla hyvinkin abstrakteja asioita, suuria kokonaisuuksia, jotka tiimin piti pilkkoa suoritettaviksi tehtäviksi. Pilkkominen tapahtui kuitenkin suoraan asiakasdemossa, joten siihen ei kulunut jätteeksi luokiteltavaa aikaa. Arvokasta oli siis myös se, että henkilö, joka kirjasi asiakasdemossa ilmenevät

asiakkaan tarpeet, oli kyvykäs kirjaamaan tarpeet suoraan tehtäviksi tehtävienhallintajärjestelmään.

Pilkkomisen jälkeen tehtävät laitettiin prioriteettijärjestykseen Kanban-mentelmän mukaisesti. Priorisoinnilla pyrittiin varmistamaan, että tehdään vain sitä, mitä asiakas todella haluaa. Yleensä priorisoinnin teki tiimin johtaja, joka laittoi tehtävät tärkeysjärjestykseen siten, että tärkeimpinä olivat asiakasdemossa asiakkaan viimeisenä mainitsemat kaksi tai kolme asiaa tärkeintä piirrettä. Priorisointi on kirjallisuustutkimuksen perusteella arvokasta, koska näin voidaan varmistaa, että asiakas saa ensimmäisenä eniten haluamiaan piirteitä.

Priorisoinnin jälkeen tehtävät olivat valmiina tehtävienhallintajärjestelmässä odottamassa, että ohjelmistokehittäjät valitsevat niitä toteutettaviksi. Valintaperusteena oli ensisijaisesti priorisointi. Näin Kanban-menetelmää kayttämällä ja tehtävät priorisoimalla tehdään asiakkaalle kaikkein tärkeimpiä asioita, eli tuotetaan asiakkaalle eniten arvoa. Myös ohjelmistokehittäjät pitivät priorisointia arvoa tuottavana:

> ”Siinä on tietenkin vaara, että jos se product back log ei ole prioriteettijärjestyksessä, niin ei tehdä niitä tärkeitä asioita, mutta jos pidetään aktiivisesti prioriteettijärjestyksessä, niin pystytään aina tekemään tärkeimpiä asioita.” [K4]

Kirjallisuudesssa väitetään, että asiakkaalle on arvokasta se, että hän saa haluamansa piirteet käyttöönsä mahdollisimman nopeasti. Kanban-menetelmän avulla tehtävät saadaan virtaamaan mahdollisimman nopeasti valmiiksi piirteiksi. Tapaustutkimuksen projektin kuvassa 17 esitetyt projektiryhmän työn mallin vaiheet tehtävän valinta, suunnittelu, ohjelmointi, oma testaus, rakenteinen testaus, toiminnallinen testaus ja valmis tehtävä vietiin Kanban-taululle sarakkeiksi. Lisäksi taululle muodostettiin sarake ongelmatehtäville. Yhden tehtävän saattoi siirtää ongelmatehtävien sarakkeeseen, jolloin kaikki näkivät, että kyseisen tehtävän kanssa on ongelmia ja pystyivät tulemaan apuun, jos oma aikataulu sen salli:

> ”To help the work with the urgent blockers, that was one thing, the Kanban board helped me to take decisions.” [K3]

Tapaustutkimuksessa tuli esiin, kuinka sarakkeisiin asetettujen rajoitteiden ansiosta

tukokset tulivat nopeasti esiin ja ne saatiin purettua. Uutta tehtävää ei saanut ottaa, mikäli tekeillä oli jo kaksi tehtävää, vaan tehtävät tuli ensin saada virtaamaan eteenpäin Kanban-taululla:

> ”Jos liput kasaantuvat, niin siitä näkee selkeesti et nyt joku tarvii apua.” [K4]

> ’Yksi henkilö muisti asiakkaan haluaman jutun, ja ajatteli alkaa tekemään sitä. Hänellä oli kuitenkin jo 2 tehtävää tekeillä. Tiimin johtaja sanoi, että ensin on tehtävä nuo pois alta ennen kuin voi ottaa sitä tehtäväksi.’ [Pir10a].

Suhteellisen pienten tehtävien ansiosta virtaus Kanban-taululla saatiin pysymään tasaisena ja nopeana. Tapaustutkimuksen perusteella pienet tehtävät, jotka virtaavat nopeasti taululla tuottavat asiakkaalle arvoa nopeutensa ansiosta. Lisäksi pienet tehtävät mahdollistivat sen, että projektiryhmä voi muuttaa nopeasti suuntaa, mikäli asiakkaan visio oli ymmärretty väärin tai asiakas halusikin jotakin erilaista. Jos taulu olisi ollut täynnä suuria, aikaa vieviä tehtäviä, olisi suunnan muutos ollut hitaampaa:

> ”Päätettiin, että tehdään vain pieniä taskeja, että kun suunta muuttuu äkkiä, niin pystytään nopeasti reagoimaan. Se oli meillä koko ajan mielessä, että pystyy vaihtamaan nopeasti suuntaa.” [K1]

Nopeaan suunnanmuutokseen projektiryhmällä oli toinenkin työkalu: Kanban-taulu pyrittiin pitämään mahdollisimman tyhjänä ennen asiakastapaamista. Tällöin tekeillä oli mahdollimman vähän työtä, ja asiakastapaamisen jälkeen voitiin ottaa uusia, tapaamisessa tulleita arvokkaimpia asioita työn alle:

> ”Yleensä ennen demoa varmaan meillä tuottavuus kasvoi, että silloin teimme aina parhaimpamme, että saamme taskit pois alta. Täytyy saada tavallaan pöytä tyhjäksi ennen demoa. Ja sitte katsotaan, että mikä on tärkeätä seuraavalle viikolle.” [K2]

Toimittamisen nopeuteen auttaa myös Kanbanin visuaalisuus. Tiimin johtaja näkee joka aamu, missä mennään, mikä on ohjelmistokehityksen sen hetkinen tilanne, ja hän voi puuttua esimerkiksi siihen, jos joku on toimettomana tai jos jollakin on ongelmia ja taululle muodostuu tukoksia.

Kaikki ohjelmistokehittäjien tekemä työ ei näkynyt Kanban-taululla. Joskus tehtävä oli niin pieni, esimerkiksi yhden ohjelmakoodirivin korjaaminen, ettei sitä viety lainkaan

Kanban-taulun läpi. Taululla ei myöskään näy uusien työkalujen ja työskentelytapojen oppiminen tai opettaminen. Kaiken työn ei haastattelujen mukaan kuitenkaan tarvitsekaan näkyä Kanban-taululla:

> ”Tiketin liikkuminen ei aina näy siellä, se on ehkä jossakin tilanteessa turhan byrokraattista, eikä sen tarvitse näkyä siellä, että jos se on.. mä koodaan että hei voits sä tehdä mulle cr:n.. siitä vierestä.. pieni asia vaikka.. tai isompi asia, ja se kattoo, että tommoisia juttuja voisi tehdä paremmin, niin onko siitä kenellekään hyötyä, että se käy laittamassa sen sinne ja kolmen minuutin päästä ja laittaa sen takaisin...Et ei mun mielestä se ole mikään tavoitekaan... Mutta erityisesti silloin, jos se jää odottamaan sitä Code Review:iä, että se silloin käytäis siirtämässä ... niin se visualisoi sen, niinkun jossain vaiheessa mun mielestä oli silloin alkupuolella, että hei täällä nyt on aika paljon jommassa kummassa laatupuolen sarakkeessa tikettejä, niin näitä pitäisi nyt saada eteenpäin.” [K1]

Asiakastapaamisessa valmiit piirteet esiteltiin asiakkaalle. Asiakkaan esittämä visio, usein hyvin abstrakti asia, oli luotu järjestelmään toimivaksi osakokonaisuudeksi. Asiakastapaamisen havainnoinnissa sekä haastatteluissa tuli esiin, että asiakas on hyvin tyytyväinen, kun hän saa joskus melko epämääräisen visionsa toteutettuna ja näkyvänä. Hän kokee saaavansa arvoa:

> ”The customer in the next demo got this working, expected thing, and he was even more happy, because he got something new, which really impressed him, he got this wow-feeling.” [K3]

Kirjallisuuden mukaan yksi Kanbanin perusperiaatteista on mittaaminen. Mittaamalla pyritään kehittämään prosessia tehokkaammaksi. Prosessia optimoidaan läpimenoajan minimoimiseksi sekä läpimenoajan ennustettavuuden parantamiseksi. Mittaamista käytetään myös oman toiminnan kehittämistä varten. Tapaustutkimuksen projektiryhmässä ei kuitenkaan mittaamista käytetty. Projektiryhmän mukaan mittaaminen ei ollut tarpeen:

> ”Ei ollut tarvetta mitata, koska se olisi ollut ylimääräistä byrokratiaa, eikä se olisi auttanut hirveästi mihinkään.” [K1]

> ”Knibergiltä luin, että ’jotta voit kehittää omaa toimintaasi’, niin en mä tiedä

> tarvitaanko siihen sitä aikamittausta. Voi olla, että jossain toisessa ympäristössä, toisenlaisessa projektissa näkisin asian eri tavalla, mutta me koko ajan yritämme tehdä niitä tärkeimpiä asioita, teemme niitä tiettyjen laatustandardien mukaan, jotka se tiimi määrittää, että minkälaista laatua se haluaa tehdä, ja teemme ne mahdollisimman nopeasti. Ja sitten teemme ne niin pienissä askelissa, että me voimme koska vain muuttaa suuntaa.” [K1]

Mittaamisella ei siis nähty tuotettavan arvoa kenellekään tässä projektissa, joten sitä ei käytetty.

6.6 Tutkimustulokset työnkulun kartoituksesta

Työnkulun kartoitusta voidaan kirjallisuuden perusteella käyttää prosessin virtaviivaistamiseen sekä jätteen löytämiseen. Havainnoinnista [Pir10a] valittiin yksi asiakkaan demossa esittämä visio, abstrakti piirre, jonka asiakas haluaa mahdollisimman nopeasti. Tämän vision matkasta valmiiksi toimivaksi järjestelmän piirteeksi seurattiin ja siitä piirrettiin visualisoitu esitys, kartoitus, kuva 19. Projektin Kanban-seinätaulun sarakkeista saatiin askeleet, jotka tehtävät käyvät prosessissa läpi.

Havainnoinnissa [Pir10a] havaittiin, että asiakkaan abstrakti visio pilkottiin tehtäviksi jo asiakasdemon aikana, mikä oli siis suoraan arvokasta työtä. Seurattavaksi valittiin asiakkaan visio yhteisöllisyyden saamisesta järjestelmään. Asiakas ei suoraan sanonut, miten tuota yhteisöllisyyttä saadaan järjestelmään, mutta hän halusi sen näkyvän järjestelmän käyttäjille:

> ’Asiakas sanoi: Community feeling mukaan kaikkialle. Se erottaa meidät muista.’ [Pir10a]

Seurattavaksi valittu asiakkaan visio, community feeling, muodosti seitsemän erillistä tehtävää: community feeling proto, recommendations, blog, evaluation, newsfeed sekä community page. Näiden tehtävien yksityiskohtainen kuvaaminen ei liity tähän tutkimukseen, kun haluttiin ainoastaan seurata asiakkaan vision toteutumista järjestelmään. Tehtävien yhteisvaikutuksena asiakkaalle tuotettiin hänen haluamaansa yhteisöllisyyden tunnetta järjestelmän käyttäjille.

Ensimmäisenä tehtävänä tehtiin Community Feeling Proto, josta tiimi keskusteli sen

valmistumisen jälkeen. Lähinnä keskusteltiin siitä, miten asiakas saa yhteisöllisyyden tunteen, onko proto vastaus asiaan:

> ’Protosta keskusteltiin 20 minuuttia. Sen jälkeen se jaettiin tehtäviksi ja alettiin hommiin.’ [Pir10a].

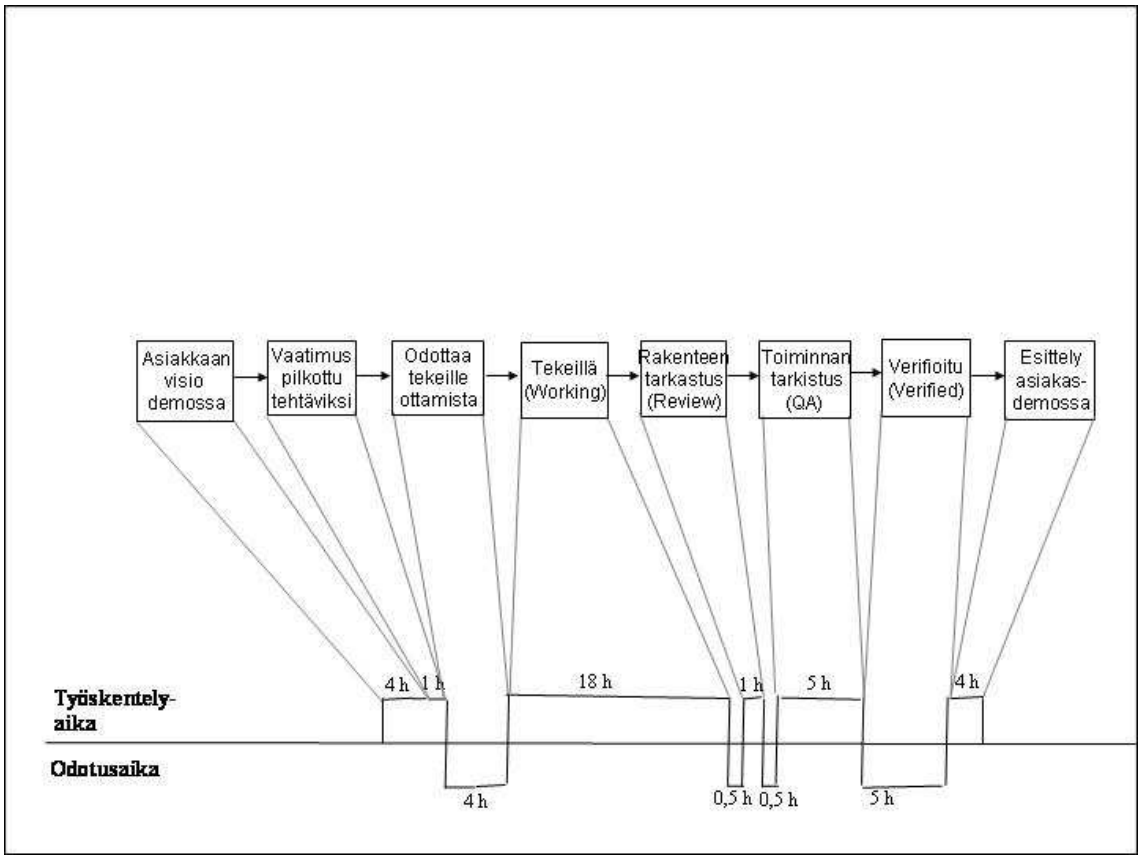
Tehtävät virtasivat Kanban-taulun läpi eri aikaan. Esimerkiksi commynity feeling proto oli valmiina yhdessä päivässä, kun taas newsfeed vei 6 päivää. Työnkulun kartoitus tehtiin hitaimpien tehtävien perusteella, jotta saadaan esiin mahdollisimman paljon jätettä, joka voidaan poistaa. Analyysin mukaan arvoa tuottavaa aikaa oli 33 tuntia, ja odotusaikaa 10 tuntia, eli noin neljännes kokonaisajasta.

Tehtävien matka working-vaiheesta valmiiksi piirteiksi sujui hyvin jouhevasti Kanban-taulun avulla. Kun ohjelmistokehittäjä oli saanut tehtävän oman osuutensa valmiiksi, hän sopi sille rakenteellisen testauksen tekijän, yleensä jonkun henkilön samasta pöytätiimistä, ja laittoi tehtävää kuvaavan lapun hänen rivilleen Kanban-taululle odottamaan tarkastuksen tekemistä. Kun rakenteinen testaus oli tehty, tehtiin sama toiminnallisen testauksen osalta. Kun toiminnallinnen testaus oli hyväksytty, oli tehtävä valmis.

Kaikki tehtävät yhtä lukuunottamatta menivät tämän kaavan mukaan. Yhdessä tehtävässä löytyi virhe toiminnallisessa testauksessa. Tehtävä vietiin takaisin tekeille ja virhe korjattiin välittömästi. Tämän jälkeen tämä tehtävä vietiin verifioitu-tilaan odottamaan asiakasdemoa. Seuraavassa asiakasdemossa yhteisöllisyyden tunne esiteltiin asiakkaalle. Tutkija havainnoi asiakasdemossa seuraavaa:

> ’Asiakas innostui, kun näki yläpalkin, kesken demon jo. Asiakkaalle tuli tunne, että osa on valmiina.’[Pir10a]

> ’Asiakaskaan ei ole varma, mitä haluaa ennen demoa, vaan kun saa jotakin valmista, sitten hänenkin visionsa kirkastuu, mitä lisää hän haluaa.’ [Pir10a]



Kuva 19: Työnkulku havannoinnin [Pir10a] perusteella.

Työnkulun kartoituksen avulla voidaan löytää sellaisia työvaiheita, jotka eivät tuota arvoa. Huomataan esimerkiksi, missä kohdissa prosessia tehtävät joutuvat odottamaan. Näihin odotusaikoihin, eli jätteeseen, voidaan kiinnittää huomiota ja niitä voidaan pyrkiä karsimaan prosessista.

Kartoituksen perusteella projektissa jätettä muodostui heti asiakasdemon jälkeen, kun tehtäviä ei vielä otettu tekeille. Tämä johtui osittain heti demon jälkeen pidettävästä retrospektiivistä, jossa tiimi pyrkii parantamaan toimintaansa, ja joka on kirjallisuuden perusteella arvokasta, vaikka ei näy Kanban-taululla. Retrospektiivin aikana kukaan ei luonnollisesti ottanut vielä mitään tehtäviä tekeille. Asiakasdemon sekä retrospektiivin jälkeen tiimi tarvitsi pienen hengähdystauon, ja tiimin yhteisellä päätöksellä työhön päätettiin ryhtyä vasta seuraavana päivänä. Asiakasdemoon valmistautuminen oli tiimille ponnistus, ja havainnoinnissa tiimi tarvitsi pienen tauon ennen uusiin tehtäviin sitoutumista.

Toinen jätteen muodostumisen olennainen tilanne oli se, että tehtävä oli saatu valmiiksi,

mutta se odotti asiakasesittelyä. Projektiryhmällä oli havainnoinnissa runsaasti valmiita tehtäviä, jotka odottivat asiakasdemoa. Valmiita tehtäviä kerääntyi viikon mittaan ensimmäisestä päivästä alkaen, joten pahimmillaan ne odottivat kuusi päivää esittelyä asiakkaalle. Tämä oli toisaalta asiakkaan toive, että Kanbaniin yhdistetään viikon mittaiset iteraatiot, jolloin odotusaikoja tulee väistämättä. Kun asiakas halusi valmiita piirteitä kerran viikossa, ei odotusaikaa ehkä voidakaan pitää jätteenä, vaan asiakkalle arvokkaana. Eri asia olisi ollut, jos järjestelmä olisi ollut tuotannossa, ja asiakkaalla olisi ollut kiire saada valmiita piirteitä tuotantoon mahdollisimman nopeasti.

6.7 Yhteenveto

Taulukossa 2 esitetään empiirisen tapaustutkimuksen tulosten perusteella tutkijan teoreettiseen malliin suhteutettuja Kanbanin kvalitatiivisia vaikutuksia ohjelmistokehittäjien työhön.

Työhön vaikuttavia seikkoja	**Kanbanin kvalitatiivisia vaikutuksia ohjelmistokehittäjän työhön.**
Dokumentaation määrä	Ohjelmistokehittäjä ei tee dokumentaatiota ellei asiakas sitä tarvitse.
Ongelmanratkaisu	Ongelmat ratkaistaan heti kysymällä lähimpänä istuvilta työkavereilta tai joltakulta tiimissä, jolla on asiantuntemusta asiasta. Ulkopuolista asiantuntijaa käytetään sellaisissa ongelmissa, joihin tiimi ei löydä vastausta. Ohjelmistokehittäjät eivät lykkää ongelmien ratkaisua eivätkä piilottele ongelmia. Tiimi myös huomaa nopeasti, jos jollakulla on ongelmia.
Visualisoiminen	Ohjelmistokehittäjä tietää koko ajan, mitä muulla tiimillä on meneillään. Kanban-taulu on kommunikaation väline. Kanban-taulun avulla työtehtäviä siirretään henkilöltä toiselle. Visualisoiminen motivoi ohjelmistokehittäjää. Visualisoiminen auttaa ymmärtämään Kanban-prosessia.
Kokonaisuuden hallinta	Ohjelmistokehittäjät pyrkivät ymmärtämään kokonaisuuden sekä markkinaympäristön: tutustumalla kilpailijoihin, osallistumalla asiakasdemoihin ja tekemällä kaikenlaisia työtehtäviä tiimin sisällä.
Kommunikoinnin määrä	Paljon vapaata kommunikointia. Päivittäistapaamiset, joissa jokaisella puheenvuoro.

Menetelmän ymmärtäminen ja omaksuminen	Menetelmän omaksuminen ja ymmärtäminen helppoa Kanban-seinätaulun avulla.
Palautteen määrä	Palaute työkavereilta on välitöntä. Palautteen paikkoja määritelty Kanban-prosessiin (Code Review sekä Quality Assurance) Asiakkaalta palaute asiakastapaamisissa.
Asioiden hyväksyttämisprosessi	Ohjelmistokehittäjän ei tarvitse hyväksyttää asioita ylemmällä taholla, vaan tiimin sisällä on sovittu hyväksymiskäytännöt. Asiakastapaamisissa asiakas hyväksyy valmiit piirteet.
Työtehtävien valinta	Kukin ohjelmistokehittäjä ottaa järjestelmästä tai seinätaululta minkä työtehtävän haluaa ottaen tehtävien priorisoinnin huomioon.

Taulukko 2: Kanbanin kvalitatiivisia vaikutuksia ohjelmistokehittäjän työhön tapaustutkimuksen perusteella.

Taulukosta 2 nähdään, että Kanbanin taustalla olevalla Lean-filosofialla on tapaustutkimuksen mukaan suurin vaikutus ohjelmistokehittäjän työn yhdeksään kirjallisuudesta valittuun ominaisuuteen. Lean-filosofian mukaisesti dokumentointia on vähän, ongelmat ratkaistaan heti, pyritään ymmärtämään kokonaisuus, kommunikointia sekä palautetta on runsaasti, asioiden hyväksytttämisprosessi on kevyt jä paras asiantuntemus työtehtävien valintaan on työpisteellä. Kanban-menetelmän työn visualisoiminen seinätaululle vaikuttaa ohjelmistokehittäjän työhön tukemalla työtehtävien valintaa, motivoimalla sekä lisäämällä kommunikointia ja tuomalla ongelmat heti näkyville. Kanban-menetelmä myös auttaa tapaustutkimuksen mukaan ohjelmistokehittäjää oppimaan ja omaksumaan mentetelmän käytön, kun menetelmä on visualisoitu seinätaululle.

Arvon muodostukseen Kanban-menetelmällä on seuraavia vaikutuksia:

- tehdään vain sitä, mitä asiakas haluaa, millä on asiakkaalle arvoa,
- mikäli asiakkaan visio on suuri kokonaisuus, pilkotaan se ensin pienemmiksi osakokonaisuuksiksi, jotta ne virtaavat tasaisesti ja nopeasti valmiiksi piirteeksi,
- vaatimukset priorisoidaan, mikä on arvokasta, koska asiakas saa haluamiaan piirteitä ensimmäisenä ja

- Kanban auttaa toimittamaan valmiita piirteitä mahdollisimman nopeasti, ja nopeus on arvoa asiakkaalle.

7 Keskustelu

Tässä luvussa keskustellaan edellisten lukujen sekä empiirisen tutkimuksen tutkimustuloksista. Tutkimustuloksia verrataan kirjallisuudesta löydettyihin tuloksiin. Kirjallisuudesta valittiin yhdeksän asiaa, joiden osalta ohjelmistokehitysmenetelmällä on vaikutusta ohjelmistokehittäjän työhön. Empiirisessä tapaustutkimuksessa tarkasteltiin, minkälaisia vaikutuksia näillä yhdeksällä asialla on Kanban-menetelmää käytettäessä. Tässä luvussa keskustellaan Kanban-menetelmän vaikutuksesta ohjelmistokehittäjän työhön. Tämän jälkeen analysoidaan sitä, kuinka arvo muodostuu Kanban-ohjelmistokehityksessä. Lopuksi keskustellaan tämän tutkimuksen rajoituksista.

7.1 Tutkimustulokset

Suurin vaikutus ohjelmistokehittäjän työhön on Lean-filosofialla, jota Kanban-menetelmä noudattaa. Itse Kanban-menetelmä on kirjallisuuden mukaan hyvin kevyt ja joustava; sitä voidaan käyttää lähes millä tavalla tahansa ja minkä muun ohjelmistokehitymenetelmän kanssa tahansa, kuten jo Poppendieck ja Poppendieck [PoP03] sekä Ladas [Lad08] väittävät. Empiirisen tapaustutkimuksen tutkimustulokset tukivat kirjallisuuden väitteitä. Kanbaniin yhdistettiin onnistuneesti piirteitä ketterästä Scrum-menetelmästä ja itse Kanbanin osalta tiimillä oli vapaat kädet muodostaa prosessistaan sellainen, mikä heidän tilanteeseensa oli sopiva.

Kniberg [Kni09] väittää, että Kanbanin kolmea perusperiaatetta on noudatettava. Nämä ovat:

- työnkulun visalisoiminen,
- yhtäaikaisen työn rajoittaminen ja
- tehtävien läpimenoajan mittaaminen.

Väite on ristiriidassa tapaustutkimuksen tutkimustulosten kanssa. Mittaamista ei projektissa käytetty, vaikka kirjallisuudessa se on yksi Kanbanin perusperiaatteista.

Kirjallisuudessa esitetään, että mittaamista tulisi käyttää toiminnan ja prosessin tehostamiseksi ja toiminnan ennustettavuuden parantamiseksi. Tässä projektissa mittaaminen ei kuitenkaan ollut tarpeen. Empiirisen aineiston perusteella voidaankin kysyä, onko mittaamisen tarpeen olla yksi Kanbanin perusperiaatteista vai voisiko se olla vain suositus?

Kirjallisuuden perusteella Kanban-menetelmää käytettäessä **dokumentaatiota** tehdään ainoastaan sen verran, kuin mitä asiakas haluaa ja tarvitsee. Tämä toteutui myös projektissa, jossa asiakas ei halunnut dokumentteja, joten niitä ei myöskään tehty. Tiimin omaan käyttöön oli jonkin verran dokumentaatiota, kuten esimerkiksi tiimin sisäinen wiki. Vaikka Lean-filosofiassa kaiken toiminnan tulisikin lähteä asiakkaasta, on kuitenkin sen mukaan arvokasta myös kehittää omaa toimintaa, ja tätä omalle toiminnalle tarpeellisen dokumentaation tuottaminen juuri on.

Ongelmanratkaisu tapahtui Lean-filosofiaa noudatellen välittömästi. Ongelmia ei piiloteltu, vaan päinvastoin pyrittiin tuomaan esiin. Projektiryhmä oli lisäksi ottanut käyttöön Scrum-menetelmästä aamupalaverin ja sen kolmen kysymyksen käytännön, joten myös sitä kautta tiimi sai tietää, jos jollakulla oli ongelmia. Ohjelmistokehittäjä sai välittömästi apua muilta. Likerin [Lik04] mukaan Lean-filosofian periaatteena on ratkaista ongelmat aina perussyyhyn asti kysymällä viisi kysymystä ja poistaa sitten perussyy. Tämä ei tapaustutkimuksen mukaan aina toteutunut. Tässä kirjallisuuden väitteiden ja empiiristen tutkimustulosten välillä on ristiriita. Ongelmat poistettiin tapaustutkimuksessa siinä määrin, kuin mitä oli toiminnan kannalta järkevää. Voidaankin kysyä, onko ohjelmistokehityksessä tarpeen kysyä aina viisi kysymystä perussyyn löytämiseksi?

Visualisointi toteutui projektissa, ja Kanban-taulu koettiin monella tavalla hyödylliseksi ohjelmistokehittäjille. Visualisoinnin avulla ohjelmistokehittäjät tiesivät koko ajan missä projektissa ollaan menossa, etenkin että onko jossakin ongelmia. Myös mentelmän prosessin ymmärtämiseen visualisointi antoi tukea. Nämä löytyivät myös kirjallisuudesta. Visualisointi tarjosi kuitenkin ohjelmistokehittäjälle muutakin: Kanban-taulu kommunikoi itsessään, auttoi tehtävien valinnassa ja tarjosi motivaatiohyötyä. Tämä on täysin uutta tietoa, tätä ei löytynyt kirjallisuudesta, vaan on löydös empiirisestä tutkimuksesta. Yksittäisten tehtävien liikkumista Kanban-taululla ohjelmistokehittäjät

eivät välttämättä seuraa.

Uutta tietoa, jota ei kirjallisuudesta löytynyt, mutta empirisessä tutkimuksessa tuli selkeästi esiin, on Kanban-taulun evoluutio projektin aikana. Projektin aluksi Kanban-taulun rakenteeseen käytettiin runsaasti aikaa. Siitä huolimatta Kanban-taulu eli koko projektin ajan, eli kuusi viikkoa. Projektissa taulussa aluksi sarakkeissa ei ollut rajoitteita. Näin kuitenkin tukokset jäävät piiloon. Kanbanin toinen perusperiaate, tekeillä olevan työn rajoittaminen, osoittautui todellakin tarpeelliseksi. Tässä empirinen tutkimus tuki kirjallisuuden väitteitä.

Kanban-taulun rakenne poikkesi alusta lähtien siitä, mitä Poppendieck ja Poppendieck [PoP03], Ladas [Lad08] tai Kniberg [Kni09] väittävät. Tässä on siis ristiriita kirjallisuuden ja empirian väillä. Projektissa taulun sarakkeet muodostettiin tiimin kesken projektin aluksi. Niistä osa oli lähellä kirjallisuudessa esitettyjä: määrittely, suunnittelu, ohjelmointi sekä testaus. Sarakkeiden välille ei kuitenkaan muodostettu erillistä Valmis-saraketta, kuten kirjallisuudessa esitettiin.

Projektin Kanban-taululla oli myös kullekin ohjelmistokehittäjälle oma rivinsä, jota pitkin hänen Kanban-tehtävänsä kulkivat taulun läpi. Kirjallisuudessa tällaisia rivejä ei ole; tässä on siis ristiriita kirjallisuuden ja empirian välillä. Projektipäällikkö huomasi kuitenkin haastattelussa, että itse asiassa nuo rivit ovat täysin turhat, jopa haitalliset. Kun ohjelmistokehittäjillä on omat rivit taululla, ei tekeillä olevien asioiden määrää voi laskea alemmas kuin noin kahteen kutakin kehittäjää kohti. Kuitenkin, kirjallisuuden mukaan juuri tekeillä olevia asioita vähentämällä saadaan Kanbanin edut esiin. Kun rajoitteet eivät koske kehittäjiä, vaan koko tiimiä, voidaan niillä mahdollisesti parantaa prosessia. Se ei kuitenkaan näy tässä tutkimuksessa vaan vaatii uuden tapaustutkimuksen.

Rivien pois jättäminen saattaa myös parantaa visualisoimista. Kun ohjelmistokehittäjä saa työnsä valmiiksi, hän voi siirtää sen seuraavaan sarakkeeseen, jolloin kuka tahansa, joka vapautuu, voi ottaa työn tehtäväksi. Projektiryhmän Kanban-taululla työ siirrettiin seuraavalle, jonka kanssa se oli sovittu tehtäväksi. Tämän todistaminen jää myös toisen tutkimuksen piiriin.

Projektissa oli selkeä pyrkimys Lean-periaatteen mukaisesti siihen, että ohjelmistokehittäjät **ymmärtävät kokonaisuuden**. Kirjallisuudessa Lean-filosofia ei

tarjoa siihen työkaluja, mutta empiirisessä tutkimuksessa niitä löydettiin. Uutena tutkimustietona siis kokonaisuuden ymmärtämistä voidaan tukea seuraavilla tavoilla:

- ohjelmistokehittäjät olivat kaikki paikalla asiakasdemoissa. Tämä kasvatti kokonaisymmärrystä,
- lisäksi tiimin sisällä kaikki voivat tehdä kaikkia työtehtäviä, mikä auttoi ymmärtämään kokonaisuutta,
- ohjelmistokehittäjät tutustuivat lisäksi markkinaympäristöön selvittämällä kilpailjoita ja kokeilemalla heidän vastaavia järjestelmiään. Näin kehittäjät saivat käsitystä myös asiakkaan asemasta ja
- yksittäisillä ohjelmistokehittäjillä ei kuitenkaan ole välttämättä tarvetta tietää koko ajan, mitä muut ovat tekemässä.

Kommunikointi oli projektissa hyvin vapaata Lean-filosofian mukaisesti. Kommunikointi osoittautui sellaiseksi, kuin Poppendieck ja Poppendieck [PoP03] esittävät, empiirinen tutkimus tuki kirjallisuuden väitteitä. Projektissa oli hyvin eri taustaisia ohjelmistokehittäjiä, mutta tiimille muodostui sen jäsenten mielestä kommunikoinnin ansiosta hyvä yhteishenki, ja he kokivat olevansa yksi yksikkö.

Kanban-mentelmän **omaksuminen** oli projektissa helppoa, kuten kirjallisuudenkin tulosten perusteella. Menetelmä on hyvin intuitiivinen, kevyt ja ymmärrettävä. Kaikki työ on nähtävissä seinätaululla. Seinätaulun muokkautuminen projektin tarpeisiin vie kuitenkin aikaa, ja siihen kannattaa todellakin paneutua.

Palautetta tuli Lean-filosofian mukaisesti runsaasti. Tiimiläiset antoivat toisilleen palautetta jatkuvasti. Asiakkaalta saatiin palautetta kerran viikossa asiakastapaamisessa. Asiakkaan palautetta ohjelmistokehittäjät olisivat toivoneet vieläkin useammin, esimerkiksi jatkuvaa sähköpostiyhteyttä asiakkaaseen.

Asioiden hyväksyttämisprosessi oli Lean-filosofian mukaisesti hyvin kevyt. Tiimillä oli omat valmismääritykset jokaiselle työvaiheelle, ne oli dokumenotoitu sisäiseen wikiin. Asiakas puolestaan hyväksyi valmiit piirteet asiakastapaamisissa. Asiakasta olisi kuitenkin tarvittu enemmän. Kuten jo Korkala ja Abrahamsson [KoA04] väittävät, on asiakkaan läsnäolon puute ongelma ohjelmistoprojektissa. Tähän ei Kanban-menetelmä tuo ratkaisua.

Tehtävien valinnassa ohjelmistokehittäjät saivat olla hyvin itsenäisiä. Liker [Lik04] väittää, että Lean-filosofian mukaan paras asiantuntemus työtehtävien valintaan on työpisteillä, tämä toteutui projektissa. Tehtävät valittiin Just-in-Time -periaattella. Ne otettiin toteutettaviksi prioriteettijärjestyksessä, mutta muutoin ohjelmistokehittäjät saivat käyttää omia motiivejaan tehtävien valinnassa. Toisin kuin kirjallisuudessa esitetään, projektiryhmä ei käyttänyt suunnittelukokouksia seuraavien toteutettavien asioiden valintaan. Tähän vaikuttivat projektin lyhyet viikon iteraatiot. Asiakaspalaverin lopuksi asiakas kertoi pari asiaa mitä hän seuraavan viikon jälkeen haluaa nähdä järjestelmässä, ja ne vietiin tehtävienhallintajärjestelmään korkeimmalla prioriteetilla. Priorisointiin alkuperäinen Kanban-menetelmä ei tarjoa mallia, mutta sittemmin kirjallisuudessa esitetään tapoja priorisoida. Tässä tutkimuksessa kuvattiin luvussa 3.6 eräs ehdotettu tapa priorisoida. Projektiryhmä ei käyttänyt kuitenkaan varsinaista priorisoinnin mallia, vaan asiakkaan tärkeimmät toiveet vietiin tehtävienhallintajärjestelmään heti asiakastapaamisen lopuksi.

Poppendieck ja Poppendieck [PoP03] sekä Ladas [Lad08] esittävät, että Kanbanissa ei ole lainkaan iteraatioita tai iteraatiokierrokset tulee pitää lyhyinä. Empiria tuki kirjallisuuden väitteitä. Projekti käytti lyhyitä, viikon iteraatioita. Iteraatiot toimivat hyvin, sillä asiakastapaamisessa tiimille selkeni mitä seuraavalla viikolla tehdään. Havainnoinnissa asiakastapaamisen aikana tiimille tuli täysin eri tehtäviä seuraavalle viikolle, kuin mitä he olivat alunperin ajatelleet tehdä. Asiakas nosti uusia asioita tärkeimmiksi, jotka priorisoitiin korkeimmalle. Vaikutti myös, että tiimille iteraatiot olivat hyvin tärkeitä. Viikon aikana Verified-sarake alkoi täyttyä, ja haastattelujen perusteella se oli hyvin motivoivaa ohjelmistokehittäjille. Asiakastapaamisiin panostettiin tosissaan, ja tiimi oli valmiina uuden viikon haasteisiin.

Kanban-menetelmää käytettäessä arvoa asiakkaalle muodostuu kirjallisuuden mukaan siitä, että jatkuvasti tehdään vain sitä, mitä asiakas haluaa. Myös asioiden priorisointi tuo arvoa, sillä näin asiakas saa eniten tarvitsemiaan piirteitä ensimmäisenä. Nopeudella on asiakkaalle kirjallisuuden mukaan suuri arvo. Nämä toteutuivat projektissa. Projektissa tehtiin vain sitä, mitä asiakas haluaa. Nämä asiat saatiin selville asiakasdemoissa. Havainnoinnin perusteella asiakas ei kuitenkaan yksin sanellut vaatimuksiaan, vaan yhteistyössä ohjelmistokehittäjien kanssa keskustellen päätyi

valintoihin seuraavalla viikolla tehtävistä asioista. Ohjelmistokehittäjä siis osallistui omalla asiantuntemuksellaan vaatimusten antamiseen.

Kuten jo Ladas [Lad08] suosittelee, suuret kokonaisuudet kannattaa pilkkoa pieniksi tehtäviksi. Tämä osoittautui tarpeelliseksi myös projektissa. Pienten tehtävien avulla virtaus saatiin pysymään nopeana, ja toisaalta suunnan muutos oli helppoa, kun asiakkaan vaatimukset muuttuivat. Pienet tehtävät auttavat ohjelmistotiimiä toimittamaan asiakkaalle hänen abstraktina visiona esittämiään toiveita. Tapaustutkimuksen mukaan asiakas on hyvin tyytyväinen, kun hän näkee visionsa toteutettuna ja toiminnassa. Tällä on siis asiakkaalle arvoa. Kun asiakas näkee yhden version abstraktista toiveestaan, hän voi tarkentaa omia tarpeitaan.

Kanban-taulu auttaa tapaustutkimuksen mukaan toimittamaan valmiita piirteitä nopeasti asiakkaalle, kuten kirjallisuudessa esitetään. Pienet tehtävät auttavat nopeuteen ja sarakkeissa olevat rajoitteet estivät tukoksia muodostumasta. Nopeuteen auttaa myös Kanban-taulun visuaalisuus. Tiimin johtaja näkee jatkuvasti onko joku toimettomana tai alkaako jonnekin muodostua tukoksia, jotka pitää selvittää.

Kanban sopii hyvin työnkulun kartoituksen pohjaksi jätteen löytämiseksi, kuten Poppendieck ja Poppendieck [PoP03] esittävät. Piirtämällä yhdestä asiakkalle tärkeästä piirteestä työnkulun kartoitus, saatiin tutkimuksessa esiin projektin prosessissa olevia odotusaikoja. Odotusaikaa muodostui lähinnä heti asiakasdemon jälkeen, kun tiimi ei ottanut mitään asioita vielä tekeille, vaan piti retrospektiivin. Asiakastapaamisen sekä retrospektiivin jälkeen tiimi tarvitsi yhteisellä päätöksellä pienen hengähdystauon ennen seuraavaan viikon tehtävien aloittamista.

Toinen tilanne, jossa työnkulun kartoituksen mukaan muodostui odotusaikoja, tuli silloin, kun tehtävä oli jo valmis, mutta odotti asiakasdemossa esittelyä. Tämä odotusaika johtui projektin käyttämästä viikon iteraatiosta yhdistettynä Kanbaniin. Asiakas kuitenkin halusi valmiita piirteitä kerran viikossa, joten viikon iteraatioilla oli asiakkaalle arvoa. Näin tehtiin mitä asiakas haluaa, vaikka tehtävien osalta muodostuikin hieman odotusaikoja.

Taulukossa 3 esitetään edellä kuvatut tutkimuksen perusteellä löydetyt empiiriset johtopäätökset.

Tutkimuskysymys	Empiirinen johtopäätös	Lyhenne
1	Kanban on kevyt ja joustava.	EJP1
	Kanban muuttaa ohjelmistokehittäjän työtä vesiputousmalliin verrattuna.	EJP2
	Kanbanin kirjallisuudessa esitettyä kolmea perusperiaatetta ei välttämättä tarvitse noudattaa.	EJP3
	Kanbanissa iteraatiokierrokset tulee pitää lyhyinä.	EJP4
2	Nopeudella on asiakkaalle suuri arvo, ja Kanban auttaa toimittamaan valmiita piirteitä nopeasti asiakkaalle.	EJP5
	Tehtävät kannattaa pitää pieninä.	EJP6
	Kartoittamalla työnkulku voidaan prosessista erotella arvoa tuottavia sekä tuottamattomia osia. Kanban sopii hyvin työnkulun visualisoinnin analysoinnin pohjaksi.	EJP7

Taulukko 3: Tutkimuksen empiiriset johtopäätökset.

7.2 Tutkimuksen rajoitukset

Kirjallisuustutkimuksessa rajoitteena oli vähäisen olemassa olevan tutkimuksen määrä. Kanbanista ohjelmistokehitysmenetelmänä ei ole juurikaan tutkimusta. Myöskään ohjelmistokehittäjän työn prosessimallista riippuvia erityispiirteitä ei ole kirjallisuudessa tutkittu. Tämän vuoksi tutkijalla ei ollut käytössään valmista mallia, johon peilata tutkimuslöydöksiään. Kirjallisuudesta löytyi kuitenkin runsaasti tutkimuksia ohjelmistoprosessimalleista, joiden perusteella muodostettiin tutkijan oma teoreettinen malli ohjelmistokehittäjän työstä, ja empiirisessä osuudessa mallia voitiin testata sekä havainnoimalla että haastatteluin.

Tutkijan muodostama teoreettinen malli ohjelmistokehittäjän työn erityispiirteistä ei ole riittävä kuvaamaan ohjelmistokehittäjän työtä kokonaisuudessaan, vaan mallia tulisi vielä kehittää. Ohjelmistokehittäjän työ sisältää muitakin erityispiirteitä, kuin mitä tähän tutkimukseen valittiin. Muodostettu malli kuvaa vain osan siitä työstä, mitä ohjelmistokehittäjä tekee. Mallissa ohjelmistokehittäjän työtä kuvataan yhdeksän kirjallisuudesta löytyneen kohteen perusteella. Prosessimalleista kuitenkin löydettiin runsaasti kuvauksia malliin valituiden erityispiirteiden osalta ja nämä löydökset validoitiin empiirisessä tapaustutkimuksessa.

Tapaustutkimus tehtiin yhdestä kohteesta. Tämän takia tutkimustuloksia ei voi suoraan yleistää, mutta niistä saattaa kuitenkin saada viitteitä siitä, miten Kanbania voidaan toteuttaa todellisissa ohjelmistoprojekteissa. Projektissa ohjelmistokehittäjiä oli kuitenkin 14, joten projekti oli kohtuullisen kokoinen ja antoi runsaasti materiaalia tutkimuksen suorittamiselle.

Kanban-menetelmä oli uusi kaikille tapaustutkimuksen projektin jäsenille. Tämän vuoksi työtavat ja myös Kanban-taulu elivät projektin aikana eivätkä ne olleet vielä vakiintuneet. Kanban-menetelmä on kuitenkin joustava eikä vaadikaan tiettyjä toimintatapoja. Perusperiaate, visualisointi, toteutui projektissa alusta alkaen.

Osalla projektin jäsenistä ei myöskään ollut kovin paljon kokemusta ohjelmistokehityksestä. Tällöin heidän mielipiteensä kvalitatiivisista vaikutuksista perustuvat vain tähän yhteen projektiin. Tällaisia henkilöitä oli haastatteltujen osalta kolme. Tutkimusmenetelmänä käytettiin kuitenkin haastattelujen osalta myös havainnointia, jossa saatiin lisävahvistusta löydetyille tutkimustuloksille.

Vertailu Kanbanin vaikutuksista ohjelmistokehittäjän työhön vesiputousmalliin verrattuna oli pääosin kirjallisuuden varassa. Vain yhdellä haastateltavalla oli kokemusta vesiputousmallista; muiden haastateltavien osalta vertailua tehtiin kirjallisuudesta löydettyihin kuvauksiin vesiputousmallista.

Vaatisi myös enemmän tutkimusta selvittää, mitkä tutkimuksen löydöksistä ovat Kanban-sidonnaisia ja mitkä muista seikoista johtuvaa. Esimerkiksi se, että tapaustutkimuksen projektissa oli runsaasti kommunikaatiota sekä mahdollisuuksia palautteeseen johtui tapaustutkimuksen tulosten mukaan osin myös projektiryhmän käyttämistä Scrum-menetelmän käytännöistä, kuten päivittäistapaamiset ja retrospektiivit. Lean-filosofian mukaisesti kommunikaatioon rohkaistaan, mutta itse Kanban-menetelmä ei anna työkaluja miten kommunkaatiota pitäisi oikeastaan lisätä.

8 Johtopäätökset

Tässä luvussa vastataan aluksi tutkimuskysymyksiin. Tämän jälkeen analysoidaan, mitä liikkeenjohdollisia vaikutuksia tutkimuksella voi olla. Pohditaan sitä, mitä tämä tutkimus voisi antaa käytännön elämään eli miten yritykset, yritysjohtajat sekä

kehittäjät voisivat hyödyntää tutkimustuloksia. Lopuksi ehdotetaan aiheita jatkotutkimukseksi.

8.1 Vastaukset tutkimuskysymyksiin

Tutkielman alussa esitettiin seuraavat tutkimuskysymykset, joihin pyrittiin löytämään vastaukset. Tutkimuskysymykset ovat:

1. Mitkä ovat Kanban-menetelmän kvalitatiiviset vaikutukset ohjelmistokehittäjän työhön?
    1.1. Miten voidaan määritellä ohjelmistokehittäjän työ?
    1.2. Millaisia muutoksia omaan työhönsä ohjelmistokehittäjä kokee Kanban-menetelmällä olevan?
2. Kuinka Kanban-menetelmä vaikuttaa arvon muodostumiseen?
    2.1. Miten ohjelmistojärjestelmän arvo muodostuu?
    2.2. Miten ohjelmistokehityksessä arvo voidaan määritellä Kanban-menetelmää soveltaen?

Tutkimuskysymykseen 1.1. vastattiin luvussa 2 ja kysymykseen 1.2. luvussa 5. Ensimmäiseen tutkimuskysymykseen vastattiin luvussa 5.1, jossa muodostettiin teoreettinen malli yhdeksästä kirjallisuudesta löydetystä asiasta, joilla on kvalitatiivista vaikutusta ohjelmistokehittäjän työhön. Tämän jälkeen analysoitiin miten vesiputous sekä Kanban –menetelmät vaikuttavat ohjelmistokehittäjän työhön mallin osalta. Analyysin tulosten yhteenveto esitettiin mallissa, taulukossa 1. Empiirinen evaluointi suoritettiin tapaustutkimuksena havainnoiden sekä haastatteluina. Empiirisen tutkimuksen tulokset esitettiin taulukossa 2.

Tutkimuskysymykseen 2.1. vastattiin luvussa 4. Toiseen tutkimuskysymykseen vastattiin luvussa 5.2, jossa esitettiin kuinka arvo muodostuu Kanban-ohjelmistokehityksessä. Arvon muodostuksen osalta empiisen evaluoinnin tulokset esitettiin luvussa 6.5. Tutkimuskysymykseen 2.2. vastattiin luvussa 5.2. Empiirinen aineisto analysoitiin luvussa 6.6.

8.2 Tutkimuksen liikkeenjohdolliset vaikutukset

Kanban-menetelmä vaikuttaa tutkielman perusteella hyvin toimivalta myös ohjelmistoprojekteissa. Kanban –menetelmää voidaan käyttää ohjelmistokehityksen prosessimallista riippumatta. Se sopii empiiristen johtopäätösten 1 sekä 4 perusteella hyvin käytettäväksi yhdessä ketterän prosessimallin kanssa, esimerkiksi viikon iteraatioin. Kanban on ohjelmistokehitysmenetelmänä hyvin kevyt ja joustava, siinä on noudatettava vain kolmea perusperiaatetta. Empiirisen johtopäätöken 3 mukaan näistä viimeinen, mittaaminen, ei ole välttämätön jokaisessa ohjelmistokehitysprojektissa.

Kanban-taulun rakenne ei ole pysyvä. Projektin alussa käytetään aikaa taulun rakenteen määrittelemiseksi sen mukaiseksi, mitä työnkulku projektissa on. Kuitenkin, tutkimuksen mukaan on odotettavissa, että taulu elää tämänkin jälkeen ja evoluutiota taululla tapahtuu. Taulun evoluutiota voi tapahtua useiden viikkojen, mahdollisesti koko projektin, ajan.

Empiirisen johtopäätöksen 2 mukaan Kanban-menetelmä ohjaa ohjelmistokehittäjän työtä. Tutkijan muodostaman teoreettisen mallin perusteella Kanban vaikuttaa olennaisesti yhdeksään tutkittuun ohjelmistokehittäjän työn erityispiirteeseen. Dokumentteja tehdään hyvin vähän, ongelmat tulevat heti esiin ja ne ratkaistaan välittömästi, työnkulku on visualisoitu seinätaululle, joka tarjoaa esimerkiksi motivaatio- sekä kommunikointihyötyä, ohjelmistokehittäjillä on pyrkimys kokonaisuuden hallintaan, kommunikointia on runsaasti, Kanban-menetelmä on yksinkertainen ja intuitiivinen menetelmä käyttää, palautetta saadaan ja annetaan runsaasti, asiat hyväksytään työpisteellä ylemmällä tasolla hyväksyttämisen sijaan ja työntekijät voivat itse valita työtehtävänsä.

Kanban-menetelmää käytettäessä asiakas ohjaa koko ohjelmistoprosessia siten, että pyritään tekemään vain sitä, mikä on arvokasta asiakkaalle. Ohjelmistokehittäjän työ on tuottaa tätä arvoa asiakkaalle. Empiiristen johtopäätösten 5 ja 6 mukaan ohjelmistokehittäjä tuottaa arvoa asiakkaalle tekemällä ensin asiakkaan eniten haluamia piirteitä ohjelmistoon. Ohjelmistoprojeteissa tärkeätä on empirisen johtopäätöksen 5 perustella nopeus – sillä on monesti runsaasti arvoa asiakkaalle. Kanban-menetelmän avulla voidaan puuttua toimittamisen nopeuteen ja parantaa sitä. Tehtävät tulee empiirisen johtopäätöksen 6 mukaan pitää pieninä, jotta ne valmistuvat nopeasti. Toinen

tapa, jolla ohjelmistokehittäjä voi tuottaa arvoa, on asiakasdemoissa. Niissä ohjelmistokehittäjä voi auttaa asiakasta selventämään toiveitaan ja vaatimuksiaan. Asiakkaan ei tarvitse yksin päättää kaikista vaatimuksista, vaan hän voi yhteistyössä ohjelmistokehittäjän kanssa suunnitella, mitä piirteitä hän ohjelmistoon haluaa.

Empiirisen johtopäätöksen 2 mukaan Kanban visualisoi työnkulun. Visuaalisuuden avulla voidaan heti huomata, jos ohjelmiston tekemisprosessissa on ongelmia. Ongelmat tulevat esiin välittömästi visuaalisen Kanban-taulun avulla. Jos Kanban-taulu alkaa mennä tukkoon eli tehtäviä on suurin sallittu määrä, on jossakin ongelma ja se on poistettava. Ongelmia ei siis voi piilotella, vaan ne ovat heti näkyvillä, jolloin niiden korjaaminen ja poistaminen voidaan aloittaa välittömästi. Empirisen johtopäätöksen 7 mukaan Kanban sopii hyvin työnkulun kartoituksen pohjaksi, jolloin prosessista saadaan esiin arvoa tuottamattomia vaiheita.

8.3 Ehdotuksia jatkotutkimuksen aiheiksi

Jatkotutkimusta tarvitaan vielä siitä, mikä on sopiva linjalla olevan työn määrä. Minimointi ei ole osoittautunut parhaaksi ratkaisuksi, sillä silloin saattaa muodostua odotteluaikoja. Jos jokin tehtävä on liian vaikea tiimille ratkaistavaksi, mutta muita tehtäviä kuitenkaan saa ottaa vastaan, saattaa tiimi jumiutua kyseisen ongelman pariin.

Myös Kanban-taulun rakenne vaatii jatkotutkimusta. Sarakkeet ovat hyvin erilaisia eri tapauksissa. Kirjallisuudessa ehdotetaan esimerkiksi sarakkeita: ToDo, määrittely, suunnittelu, ohjelmointi ja valmis. Tapaustutkimuksessa käytettiin sarakkeita: Backlog, Assigned, Blocking, Working, in Code Review, in QA ja Verified. Ilmeisesti Kanban-taulun rakenteen kannattaa riippua ainakin projektista, siitä millaiset sarakkeet ovat projektin tarpeisiin parhaat. Tapaustutkimuksessa käytettiin myös rivejä: kullakin ohjelmistokehittäjällä oli oma rivinsä Kanban-taululla. Tarvittaisiin lisätutkimusta siitä, minkälainen on optimaalinen Kanban-taulu eri olosuhteissa.

Tapaustutkimuksen perusteella kolmas Kanbanin perusperiaate, mittaaminen, ei ehkä ole tarpeen joka projektissa. Tapaustutkimuksen projektissa sitä ei käytetty, ja projekti saatiin silti läpivietyä menestyksekkäästi. Mittaamisen käyttäminen vaatisi siis jatkotutkimusta: millaisissa projekteissa sitä kannattaa ja toisaalta ei ole tarpeen käyttää.

Lähteet

Abr03 Abrahamsson, P., Extreme Programming: First Results from a Controlled Case Study. *The proc. of the 29th Euromicro Conf*. Belek-Antalya, Turkey. IEEE Computer Society, 2003, sivut 259-266.

Abr10 Abrahamsson, P., Unique Infrastructure Investment: Introducing the Software Factory. *Software Factory Magazine*, 1,1(2010), sivut 2-3.

AbS04 Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J., Agile Software Development Methods: Review and Analysis. VTT Publications 478. VTT Electronics. Espoo, 2002.

AbW03 Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J., New Directions on Agile Methods: A Comparative Analysis. *The proc. of the 25th Internat. Conf. on Software Engineering (ICSE'03)*, 2003, sivut 244-254.

AlM09 Alwardt, A., Mikeska, N., Pandorf, R., Tarpley, P., A Lean Approach to Designing for Software Testability. *AUTOTESTCON, 2009 IEEE*, 2009, sivut 178-183.

BaT75 Basili, V.R., Turner, A.J., Iterative Enhancement: A Practical Technique for Software Development, *IEEE Trans. Software Engineerin*g, 1,4(1975), sivut 390-396.

Bec99a Beck, K., Embracing Change with Extreme Programming. *IEEE Computing*, 32, 10(1999), sivut 70-77.

Bec99b Beck, K., *Extreme programming explained: Embrace change*. Addison-Wesley, 2000.

BeS98 Becker, M., Szczerbicka, H., Modeling and Optimization of Kanban Controlled Manufacturing Systems with GSPN Inluding QN. *1998 IEEE Internat. Conf. on Systems, Man, and Cybernetics*, 1998, sivut 570-575 Vol 1.

Bro87 Brooks, F.P., Jr, No Silver Bullet: Essence and Accidents of Software Engineering. *Computer*, 20, 4(1987), sivut 10-19.

Cha08 Chai, L.L.S., e-based inter-enterprise supply chain Kanban for demand and order fulfilment management. *2008 IEEE Internat. Conf. on Emerging Technologies and Factory Automation*, 2008, sivut 33-35.

CoW98 Cook, J.E., Wolf, A.L., Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7,3(1998), sivut 215-249.

CoC96 Cooper, R., Chew, W.B., Control Tomorrow's Costs Through Today's Designs. *Harward Business Review*, 01(1996).

DuF95 Duri, C., Frein, Y., Di Mascolo, M., Performance evaluation of Kanban multiple-product production systems. *1995 INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, 1995, sivut 557-566.

Dye94 Dyer, J., Dedicated assets: Japan's Manufacturing Edge. *Harward Business Review*, 11(1994).

Ett95 Ettl, M., Determination of Optimal Kanban Allocation Patterns in Serial Production Lines. *Production Planning & Control*, 10,5 (1999), sivut 439-447.

Fav96 Favaro, J., When the Pursuit of Quality Destroys Value. *Software IEEE*, 13, 3 (1996), sivut 93-95.

Fav02 Favaro, J., Managing Requirements for Business Value. *Software IEEE*, 19, 2 (2002), sivut 15-17.

Gra89 Graham, D.R., Incremental Development and Delivery for Large Software Systems. Teoksessa [Sof89], sivut 156-195.

HaR06 Hartwell, J.K., Roth, G.L, Ariens Company: A lean enterprise change case study. Lean Aerospace Initiative, Massachusetts Institute of Technology, 2006. [8.1.2010]

HiH01 Hirsijärvi, S., Hurme, H., *Tutkimushaastattelu.* Yliopistopaino, Helsinki 2001.

Hum89 Humphrey, W.S., *Managing the Software Process.* Addison-Wesley, 1989. Sivut 1-30.

KaL94 Kajaste, V., Liukko, T., *Lean-toiminta*. Metalliteollisuuden Kustannus Oy, Tampere 1994.

Kni09 Kniberg, H., Kanban vs Scrum – how to make the best of both, 2009. http://blog.crisp.se/henrikkniberg/2009/04/03/1238795520000.html [30.11.2009].

KnS10 Kniberg, H., Skarin, M., Kanban and Scrum making the most of both, 2010. http://www.infoq.com/minibooks/kanban-scrum-minibook [7.4.2010].

KoA04 Korkala, M.,Abrahamsson, P. Extreme Programming: Reassessing the Requirements Management Processfor an Offsite Customer. *Software Process Improvement 11th European Conf., EuroSPI 2004*, Trondheim, Norway, November 10-12, 2004 Proceedings.

Kom04 Komi-Sirviö, S., Development and Evaluation of Software Process Improvement Methods. Ph.D. Thesis, Faculty of Science, University of Oulu, 2004.

Lad08 Ladas, C, *Scrumban*. Modus Cooperandi Press, Seattle WA 98109, 2008.

LaS90 Lambrecht, M., Segaert, A., Buffer Stock Allocation in Serial and Assembly Type of Production Lines. *Internat. Journal of Production Management*, 10,2 (1990) sivut 47-61.

Lar07 Larman, C., *Agile & iterative development: a manager's guide*. Addison-Wesley, 2007.

Lik04 Liker, J., *The Toyota Way*. The McGraw-Hill Companies, 2004.

McT05 McConnell, S., Tripp, L., Professional Software Engineering: Fact or Fiction? Teoksessa *Software Engineering*, R.H. Thayer ja M.J. Christensen, toim. John Wiley & Sons, IEEE Computer Society Publications, 2005, sivut 73-78.

McR89 McKay, K.N., Rooks, M., Watmims JIT/Kanban benchmark summary and recommendations. *Proc. of the 1989 Winter Simulation Conf.*, 1989.

Mid01 Middleton, P., Lean Software Development: Two Case Studies. *Software Quality Journal*, 9,4 (2001), sivut 241-252.

MoW08 Mohamed, S.I., Wahba, A.M., Value Estimation for Software Product Management. *2008 IEEE Internat. Conf. on Industrial Engineering and Engineering Management*, 2008, sivut 2196-2200.

NiR04 Nightingale, D.J., Rhodes, D.H., Enterprise Systems Architecting: Emerging Art and Science within Engineering Systems. *MIT Engineering Systems Symposium, Engineering Systems Monograph*, 2004.

Nyf08 Nyfjord, J., Towards integrating agile development and risk management. Dissertation, Stockholm University, Department of Computer and Systems Sciences, 2008.

Ost87 Osterweil, L., Software Processes are Software Too. *Proc. of 9th Internat. Conf. Software Engineering.* IEEE Computer Society, 1987, sivut 2-13.

Ost97 Osterweil, L., Software Processes are Software Too. *Proc. of 19th Internat. Conf. Software Engineering, revisited: an invited talk on the most influential paper of ICSE 9.* 1997, sivut 540-548.

Pat08 Patton, J., Amiguous Business Value Harms Software Products. *Software IEEE*, 25, 1 (2008), sivut 50-51.

Pir10a Pirinen, E., Havainnointipäiväkirja. Software Factory –projekti, Helsingin Yliopisto, Tietojenkäsittelytieteen laitos 2010.

Pir10b Pirinen, E., Teemahaastattelut litteroituna. Software Factory –projekti, Helsingin Yliopisto, Tietojenkäsittelytieteen laitos 2010.

PoP03 Poppendieck, M., Poppendieck, T., Lean Software Development: An Agile Toolkit, Addison-Wesley, 2003. http://www.poppendieck.com/ld.htm [30.11.2009].

Pop07 Poppendieck, M., Lean Software Development. *29th Internat. Conf. on Software Engineering* (ICSE'07 Companion), 2007.

Pre05 Pressman, R., Software Engineering. Teoksessa *Software Engineering*, R.H. Thayer ja M.J. Christensen, toim. John Wiley & Sons, IEEE Computer Society Publications, 2005, sivut 1-27.

Ram98 Raman, S., Lean Software Development: is it feasible. *The AIAA/IEEE/SAE*

Digital Avionics Systems Conf., 1998. Proc., 17TH DASC, 1998, sivut C13/1 - C13/8 vol.1.

RoL06 Roth, G.L., Labedz, C.S., Rockwell Collins lean enterprise change case study. Lean Aerospace Initiative, Massachusetts Institute of Technology, 2006. http://lean.mit.edu/index.php?option=com_docman&task=doc_details&gid=1295&Itemid=332 [8.1.2010].

Rou00 Routio, P., *Tuote ja tieto*. Taideteollisen korkeakoulun julkaisu, Gummerus Kirjapaino Oy, 2000. http://www2.uiah.fi/projects/metodi/064.htm#teemahaas [01.02.2010].

Roy70 Royce, W.W., Managing The Development of Large Software Systems. *Technical Papers of Western Electronic Show and Convention (WesCon),* Los Angeles, USA, 25,28 (1970). http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf [8.1.2010].

Sal07 Salo, O., *Enabling Software Process Improvement in Agile Software Development Teams and Organizations*. Ph.D. Thesis, Faculty of Science, University of Oulu, 2007.

ScB02 Schwaber, K., Beedle M., *Agile Software Development with Scrum*. Prentice Hall, 2002.

Shi09 Shinkle, C., Applying the Dreyfus Model of Skill Acquisition to the Adoption of Kanban Systems at Software Engineering Professionals (SEP). *2009 Agile Conf.*, 2009, sivut 186-191.

Sih94 Siha, S., The Pull Production System: Modelling and Characteristics. *Internat. Journal of Production Research*, 32,4 (1994), sivut 933-949.

Sim96 Simmons, P., Quality Outcomes: Determining Business Value. *Software IEEE*, 13, 1 (1996), sivut 25-32.

Sin89 Sinclair, G., The Recovery of Software Projects. Teoksessa [Sof89], sivut

209-219.

Sof89 *Software Engineering for Large Software Systems*, B.A Kitchenham, toim. Elsevier Science Publishers LTD, 1989.

Som00 Sommerville, I., *Software Engineering*, Addison-Wesley, 2000, sivut 8, 42-71, 557-579.

Swa03 Swank, C.K., The Lean Service Machine. *Harward Business Review*, 81,10 (2003), sivut 123-129.

Tay86 Taylor, R.S., *Value-added processes in information systems,* Alex Publishing Corporation, New Jersey, USA, 1986.

WiD02 Williams, E.J., DeWitt, C., An approach and interface for building generic manufacturing Kanban-systems models. *Proc. of the 2002 Winter Simulation Conf., 2002,* Volume 2, 2002, sivut 1138 - 1141.

Wir95 Wirth, N., A Plea for Lean Software. *Computer*, 28,2 (1995), sivut 64 - 68.

WoJ90 Womack, J.P., Jones, D.T., Roos, D., *The Machine That Changed The World*. Simon & Schuster, 1990.

WoJ94 Womack, J.P., Jones, D.T., From Lean Production to the Lean Enterprise. *Harward Business Review*, 03 (1994).

Yam82 *Yamashina, H., Japanilainen valmistusfilosofia ja Kanban-järjestelmä (Japansk tillverkningsfilosofi och Kanban-systemet),* Suomen Metalliteollisuuden Keskusliitto, Metalliteollisuuden Kustannus Oy, 1982.

Liite 1. Teemahaastattelun kysymykset

Tapaustutkimuksen kuudessa teemahaastattelussa keskusteltiin seuraavana esitellyistä teemoista. Kysymykset ovat ohjeellisia, lähinnä haastattelijan muistilistaksi.

1. TEEMA: Mikä on työntekijöiden lähtötaso ohjelmistokehittäjinä?

Miten arvioisit kokemuksesi ohjelmistokehittäjänä?
Oliko sinulla odotuksia / ennakkoluuloja ohjelmistokehitysmenetelmän suhteen?
Ovatko ne toteutuneet?

2. TEEMA: Kanbanin kvalitatiiviset vaikutukset ohjelmistokehittäjän työhön?

Dokumentit
Miten kuvalisit työssä syntyneitä dokumentteja?
Kuinka paljon dokumentteja syntyi?
Miksi dokumentteja syntyi vähän?
Ongelmanratkaisu
Mitkä asiat aiheuttivat ongelmia ohjelmistokehityksessä?
Miksi ongelmia ilmeni?
Miten ongelmia käsiteltiin?
Mitä ennakoimattomia asioita kohdattiin?
Miten ennakoimattomia asioita käsiteltiin?
Visualisointi
Miten kuvailisit työskentelytapojen ymmärtämistä?
Miten kuvalisit työnkulkua?
Mitä mieltä olet siitä, että prosessi oli näkyvillä siinä seinätaululla?
Tuliko seinätaulua katseltua?
Miksi taulu muotoutui ajan kanssa erinäköiseksi?
Kokonaisuuden hallinta
Miten kokonaisuuden ymmärtämistä tuettiin / ei tuettu?
Mitkä asiat edistivät kokonaisuuden ymmärtämistä?
Entä mitkä asiat haittasivat sitä?
Miten itse pidit kokonaisuuden hallinnassa?
Kommunikoinnin määrä
Paljonko on ollut mahdollisuuksia keskusteluun?
Miksi kommunikointia oli paljon?
Menetelmän ymmärtäminen ja omaksuminen
Miten olet kokenut käytettävän ohjelmistokehitysmentelmän?
Mitä olet oppinut?
Toimiiko seinätekniikka menetelmän oppimisen apuvälineenä?
Miksi?
Miten mielestäsi kaikki ovat olleet mukana / kiinnostuneita ohjelmistokehitysmenetelmän käytöstä ja kehittämisestä?
Palautteen määrä
Miten on rohkaistu, kannustettu, tunnustettu ja kannustettu?

Miten ja kuka on kannustanut?
Miksi kannustettiin?
Miten työtoverit ovat tukeneet toisiaan?
Annettiinko rakentavaa palautetta?
Milloin sitä annettiin?
Asioiden hyväksyttämisprosessi
Miksi asiat piti hyväksyttää?
Miten asiat tulivat hyväksytetyiksi?
Kuka hyväksyi seuraavaksi tehtävän asian valinnan?
Kuka hyväksyi tehdyn asian valmiiksi?
Työtehtävien valinta
Miten kuvailisit työtehtävien valintaa?
Millä perusteilla itse valitsit työtehtäviä?
Miksi käytit niitä perusteita?

Millaisia positiivisia yllätyksiä olet kokenut ohjelmistokehitysmentelmän osalta?
Onko toiminta muuttunut, näkyykö se käytännössä?

3. TEEMA: Miten Kanban-menetelmä vaikuttaa arvon muodostukseen?

Miten olet saanut tietoa asiakkaan tarpeista ja vaatimuksista?
Miten arvoa on toimitettu asiakkaalle?
Miten asiakkaan tunne arvosta on saatu maksimoitua?
Miten mielestäsi on saatu valmiita piirteitä asiakkaalle mahdollisimman usein?
Kuinka suuri osa tekemästäsi työstä on näkynyt seinätaululla?
Miksi?
Kuinka suuri osa tekemästäsi työstä on mielestäsi vaikuttanut asiakkaalle toimitettavaan arvoon?
Miten toimintaa mitattiin?
Miksi mitattiin/ei mitattu?