Tampereen yliopisto

Olli Ruotsalainen

PROJEKTINHALLINTAMENETELMIEN YHTEEN NIVOUTUMINEN OHJELMISTO-PROJEKTEISSA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Huhtikuu 2020

TIIVISTELMÄ

Olli Ruotsalainen: Projektinhallintamenetelmien yhteen nivoutuminen ohjelmistoprojekteissa
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Huhtikuu 2020

Ohjelmistoprojektien hallinta on ollut haasteena ensimmäisistä suuremmista projekteista lähtien. Haasteeseen on vastattu projektinhallintamenetelmillä. Osa varsinkin varhaisista menetelmistä on muilta teollisuuden aloilta ohjelmistotuotantoon sovellettuja ja mukautettuja. Projektinhallintamenetelmien kehitys on edennyt aalloissa. Uusien menetelmien käyttöönotto on mahdollistanut aiempaa syvällisemmän haasteiden tunnistamisen ja siten aina seuraavan sukupolven projektinhallintamenetelmien kehittämisen. Vanhemmat menetelmät eivät ole kuitenkaan kokonaan poistuneet käytöstä. Joissain projekteissa projektinhallinnan haasteet liittyvät ensisijaisesti ongelmiin, joita vanhemmilla menetelmillä on pyritty ratkaisemaan ja niissä nämä menetelmät voivat olla sellaisenaan käytössä. Projektin erilaiset hallinnolliset haasteet saatetaan myös kohdata eri projektinhallintamenetelmien avulla, jolloin projektin hallintamenetelmä kokonaisuutena on nivoutunut yhteen erilaisista ja mahdollisesti myös eri sukupolvien menetelmistä.

Tässä tutkielmassa on perehdytty projektinhallintamenetelmien eri sukupolvien piirteisiin ja niihin liittyviin käytäntöihin. Erityisesti on kiinnitetty huomiota piirteiden tunnistamiseen ja siihen, miten eri projektinhallintamenetelmien ja toisaalta myös niiden eri sukupolvien piirteet sekoittuvat käytettäessä projektissa eri menetelmistä yhteen nivoutunutta kokonaisuutta. Kirjallisuuteen pohjautuvan tarkastelun lisäksi on tutkittu myös yhtä case-projektia. Siinä projektin hallintakokonaisuus on purettu osiin ja tutkittu miten osat vastaavat eri projektinhallintamenetelmien piirteitä ja millä menetelmillä on vastattu mihinkin projektin hallinnolliseen haasteeseen. Tämän pohjalta on pystytty toteamaan, miten projektinhallintamenetelmien yhteen nivoutuminen ilmenee käytännössä.

Avainsanat: ohjelmistotuotanto, projektinhallinta, agile, scrum, kanban

1 Johdanto

Ohjelmistoprojektien hallinta on ollut suuri haaste jo ohjelmistoalan alkuajoista asti. Ohjelmistot ovat luonteeltaan abstrakteja rakennelmia, ja lisäksi tilaajilla tai asiakkailla saattaa olla puutteellisia käsityksiä ohjelmistojen toteuttamisesta ja tarvittavasta määrittelyn tasosta. Haikalan ja Mikkosen [2011] mukaan näihin haasteisiin on vastattu kehittyvällä ohjelmistotuotannon teollisuuden- ja tieteenalalla ja niiden osana kehitetyillä ohjelmistoprojektien hallintamenetelmillä. Varhaisia menetelmiä, kuten vesiputous tai V-malli, on otettu käyttöön ohjelmistoprojekteissa soveltamalla ja mukauttamalla muilla teollisuudenaloilla hyväksi havaittuja menetelmiä [Haikala ja Mikkonen 2011]. Näitä ensimmäisen aallon menetelmiä on seurannut uudentyyppisiä ratkaisuja, kuten Scrum tai Kanban, joita kutsutaan yleisesti *ketteriksi menetelmiksi* (Agile methods). Näissä menetelmissä on hyödynnetty ohjelmistoprojektien hallinnasta kerääntynyttä kokemusta, jonka avulla niitä on muokattu aikaisempia ohjelmistoalalle paremmin sopiviksi, perusajatuksen ollessa tekemisen, tulosten ja ihmisten painottaminen prosessien ja rakenteiden sijaan [The Agile Alliance 2001].

Ketteriä menetelmiä ovat esimerkiksi Scrum, Kanban ja XP (Extreme Programming) [Cockburn 2001]. Scrum määrittelee tiettyjä tehtäviä, joihin projektinhallinnan pitää vastata sekä työskentelytapoja, kuten työskentelyn pyrähdysmäisinä Sprint-pätkinä, joiden sisältöön ohjelmiston toteuttavalla ryhmällä on paljon päätäntävaltaa [Schwaber and Beedle 2002]. Kanban puolestaan perustuu jatkuvan työskentelyn malliin ilman työn jaksotusta, jolloin uusia kehitystehtäviä priorisoidaan ja otetaan työn alle sitä mukaan, kun niitä havaitaan tai halutaan ottaa mukaan toteutettavaan ohjelmistoon [Anderson 2010]. Extreme Programming on enemmän pienten tiimien ohjelmointitekniikoihin ja työskentelyyn keskittyvä ohjeisto [Cockburn 2001]. Yhteistä näille menetelmille verrattuna niitä edeltävien sukupolvien menetelmiin on etenkin ketteryys. Menetelmän ketteryydellä tarkoitetaan etukäteen tehtävän suunnittelun pitämistä mahdollisimman kevyenä ja muutoksien tekemistä mahdollisimman helppona, koska kokemus on osoittanut ohjelmistojen laajan etukäteen tehtävän määrittelyn olevan hyvin hankalaa [Haikala ja Mikkonen 2011]. Käytännössä nämä ketterät menetelmät ovat usein sekoittuneet keskenään, kun useammasta menetelmästä on valittu käytäntöjä vastaamaan projektin kohtaamiin projektinhallintahaasteisiin [Terlecka 2012]. Näin ollen projektin hallintamenetelmäksi on nivoutunut yhteen osia erilaisista ketteristä ja myös niitä edeltäneistä menetelmistä.

Tämän tutkielman tutkimuskysymys rakentuu erilaisien ohjelmistotuotannon projektinhallintamenetelmien ja -järjestelmien ja niiden ominaisuuksien läpikäynnin ja vertailun perustalle. Näistä tunnistetaan niitä määrittävien piirteiden lisäksi sitä, mihin käyttötarkoituksiin ja projektinhallinnan haasteisiin mitkäkin menetelmien osat vastaavat. Yksittäisten projektinhallintamenetelmien käsittelemisen lisäksi tutkitaan erikoistapauksia, joissa samassa projektissa käytetään rinnan tai päällekkäin eri projektinhallintamenetelmien käytäntöjä. Näitä tapauksia eritellään ja arvioidaan kirjallisuudesta löytyvien esimerkkien avulla. Modernit ketterät menetelmät, erityisesti Agile Manifesto -pohjaiset, ovat lähtöisin teollisuuden kokeellisesta toiminnasta tieteellisen tutkimuksen sijaan. Tämä näkyy myös tutkielmaan valitussa kirjallisuudessa, joka näiden menetelmien ominaisuuksien ja piirteiden tunnistamisen osalta painottuu tieteellisen kirjallisuuden rinnalla menetelmien kehittäjien teoksiin. Koska kyseessä on menetelmien perusominaisuuksien tunnistaminen, pidin niiden alkuperäisten kehittäjien näkemyksiä tarkoituksenmukaisimpina kuin muiden niistä myöhemmin tekemää tutkimusta. Perusteoksiin ja oppikirjoihin pohjautuvat projektinhallintamenetelmien ja niiden piirteiden kuvaukset on esitetty juuri siinä muodossa, missä alalla työskentelevä on ne tottunut käytännössä tunnistamaan. Ketterien menetelmien sekoittumisen ja niiden arvioinnin ja vertailun osalta lähteet painottuvat tutkimuskirjallisuuteen, mikä myös monipuolistaa menetelmien kehittäjien ehkä hieman yksipuolista lähestymistapaa. Tämä jako on myös perusteltua, koska työn varsinainen tarkoitus on tutkia juuri menetelmien sekoittumista, ei niinkään itsenäisiä menetelmiä. Tästä lähtökohdasta voidaan tunnistaa ja arvioida, miten ohjelmistoprojektien hallintajärjestelmät nivoutuvat yhteen useiden erilaisten projektinhallintamenetelmien piirteistä.

Erilaisten projektihallintamenetelmien erottuminen samassa projektissa on esillä tutkielmassa käsitellyssä case-esimerkissä. Esimerkkiprojektin hallintamenetelmät on jaettu osiin ja tutkittu mistä projektinhallintamenetelmästä mikäkin osa on lähtöisin käyttäen hyväksi erilaisten projektinhallintamenetelmien näkyviä piirteitä ja prosesseja. Tässä tutkimuksessa käytetty case-projekti on varsin tyypillinen verkossa toimiva sovellus, sähköinen asiointijärjestelmä, joka kattaa front- ja backendin. Erityspiirteenä osa projektista siirtyi kehitystiimille toiselta yritykseltä kilpailutuksen tuloksena, ja sen osalta kyse oli valmiin ohjelmiston jatkokehityksestä, joidenkin muiden osien kohdalla tiimi aloitti työn puhtaalta pöydältä. Projektin piirteitä ja menetelmiä tutkimalla oli mahdollista löytää yhteyksiä eri projektinhallintamenetelmiin, joten tässä kyse oli selvästä tapauksesta, jossa hallintamenetelmä on nivoutunut yhteen eri projektinhallintamenetelmistä.

Tutkielman luvussa 2 käsitellään ohjelmistotuotannon kehitystä ja projektinhallintamenetelmien vaiheita sekä perinteisiä menetelmiä. Ketteriä kehitysmenetelmiä ja niiden piirteitä eritellään ja vertaillaan luvussa 3. Case-projekti ja sen projektinhallintamenetelmän yhteen nivoutuminen esitellään luvussa 4. Luku 5 sisältää loppupäätelmät.

2 Ohjelmistotuotanto ja projektinhallintamenetelmät

Ohjelmistotuotanto syntyi vastaamaan ohjelmistoteollisuudelle ominaisiin projektinhallinnan haasteisiin. Tässä luvussa käydään lyhyesti läpi ohjelmistotuotanto ja -projektienhallinta. Luvussa esitellään myös perinteisiä ohjelmistoprojektien hallintamenetelmiä.

2.1 Projektinhallinta ohjelmistotuotannossa

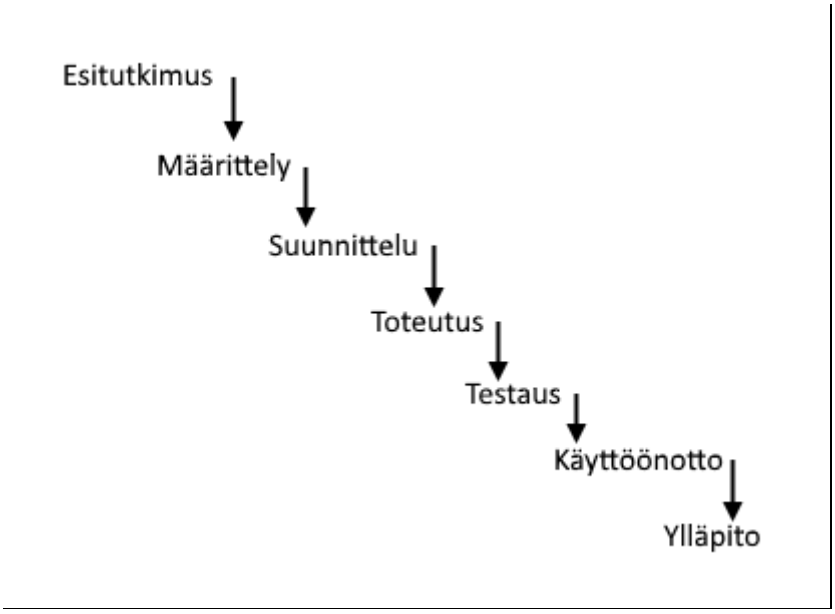
Haasteet suurien ohjelmistojärjestelmien toteuttamisen hallinnassa ovat olleet nähtävissä heti niistä ajoista lähtien, kun näiden järjestelmien valmistaminen on alkanut, erityisesti suurien ohjelmistohankkeiden ongelmissa. Näitä haasteita pyrittiin aluksi hallitsemaan muualta prosessiteollisuudesta omaksutuilla menetelmillä, joita ovat esimerkiksi IPMA tai PMI. Abstraktina työnä ohjelmistokehitys kuitenkin on luonteeltaan erilaista tavanomaiseen teollisuuteen nähden niin työn määrittelyn ja tilaamisen kuin toteuttamisenkin puolesta. Ratkaisuna tähän projektinhallintamenetelmiä kehitettiin esimerkiksi tuomalla niihin paremmin ohjelmistotuotannon tarpeisiin sopiva jaksotus. Siinä varsinainen työ on erotettu määrittely- ja suunnitteluvaiheista. [Haikala ja Mikkonen 2011]

Vaiheet ovat oleellinen osa ohjelmistotuotannon ensimmäisen sukupolven projektinhallintamenetelmiä. Niissä projekti etenee vaihe kerrallaan, vaiheen tuottama tieto tai dokumentaatio toimii aina seuraavan vaiheen syötteenä. Perusajatuksena on saada seuraavalle vaiheelle selkeä tieto siitä mitä pitää tehdä. Tämä tiedon ja määrittelyn epämääräisyys oli suuri ongelma aikaisemmissa menetelmissä. Vaiheistetuissa menetelmissä ongelmana on kuitenkin muutoksenhallinta. Myöhäisessä vaiheessa havaittu virhe voi tarkoittaa palaamista monta vaihetta taaksepäin. Erityisesti jos virhe liittyy määrittelyyn tai tilaajan ja toteuttajan välisiin väärinkäsityksiin ohjelmiston toiminnasta, voi tämä ohjelmiston abstraktin luonteen takia johtaa suureen muutostyöhön myös muissa ohjelmiston osissa. [Haikala ja Mikkonen 2011]

Tässä luvussa esitellään yleisimpinä malleina vesiputous-, V-, inkrementaalinen ja iteratiivinen malli. Muina esimerkkeinä ohjelmistotuotannon projektinhallintamalleista Unnati [2016] mainitsee evolutionäärisen sekä spiraali-, prototyyppi- ja Rapid Development (RAD) -mallit.

2.2 Vesiputousmalli

Yksi laajasti käytössä oleva varhainen ohjelmistotuotannon projektinhallintamenetelmä on vesiputousmalli. Siinä ohjelmistotuotannon eri vaiheet on tarkasti määritelty ja eritelty toisistaan. Varsinaista suunnittelua ja toteutusta tuetaan useilla esi- ja jälkivaiheilla. Vesiputousmalli on esitetty kuvassa 1. Selkeän vaihejaon lisäksi oleellista on yhden vaiheen tekeminen loppuun ja sen hyväksyminen usein mittavan muodollisen hyväksymisprosessin kautta, ennen kuin seuraavaa vaihetta voidaan aloittaa sekä eri vaiheiden tuotokset tarkasti kuvaava kattava dokumentaatio. [Royce 1970]

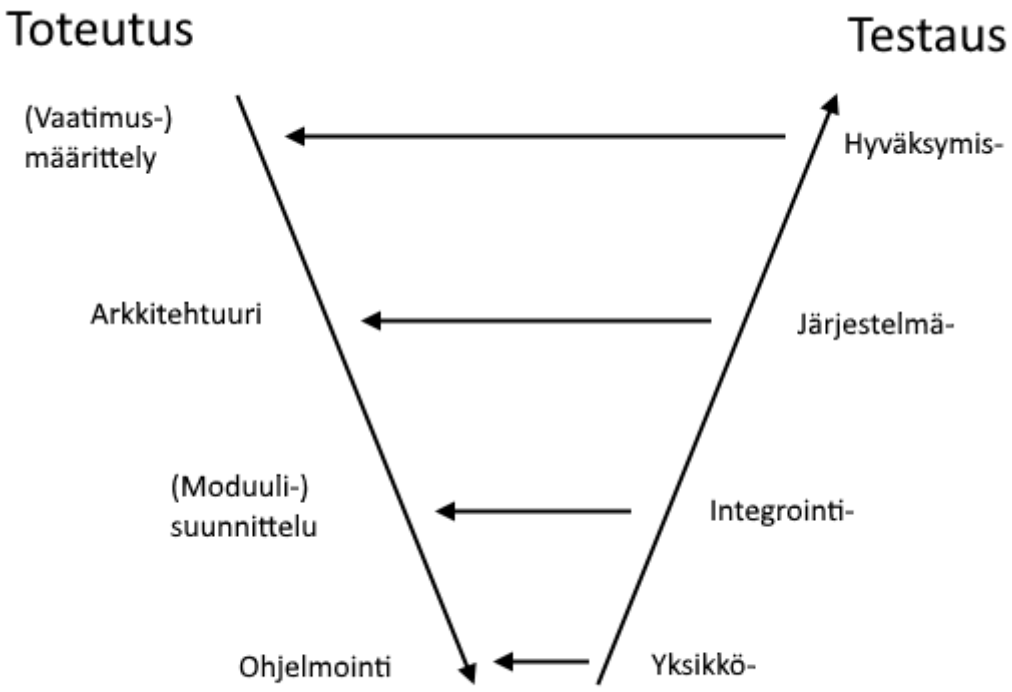


Kuva 1: Vesiputousmalli Haikalaa ja Mikkosta [2011] mukaillen.

Vaiheittaisen etenemisen tulisi johtaa virheiden aikaiseen huomioimiseen. Käytännössä tämä ei kuitenkaan ole toiminut ja virheet havaitaan vasta myöhemmissä vaiheissa. Kaikkea projektin hallintaan tarvittavaa tietoa ei ole sen alkuvaiheessa tiedossa tai tarpeet muuttuvat kesken projektin. Lisäksi vesiputousmalli toteutetaan käytännössä usein kankeampana ja lineaarisempana kuin se on alun perin esitelty. Vesiputousmallissa palaaminen aikaisempiin vaiheisiin on käytännössä kallista, hidasta ja hankalaa. Siitä huolimatta sen perusajatukset ohjatusta suunnittelu- ja projektinhallintamenetelmästä ovat osoittautuneet toimiviksi ja vesiputousmaisia piirteitä voi havaita uudemmissa projektinhallintamenetelmissä. [Haikala ja Mikkonen 2011]

2.3 V-malli

Kehitystarpeet vesiputousmallissa johtivat V-malliin. Siinä ohjelmiston testaus ja laajemmin laadunvarmistus- ja hyväksymisprosessi on myös vaiheistettu ja sidottu ohjelmiston suunnittelun ja toteuttamisen eri vaiheisiin. V-mallin vasen reuna on pitkälle vesiputousmallin mukainen, kuitenkin kunkin vaiheen yhteydessä suunnitellaan myös oikean reunan hyväksymis- ja varmistusmenettely. Tämän samanaikaisen suunnittelun etuna on, että oikean reunan tarkistukset todennäköisemmin vastaavat määrittelyn ajatusta. Vesiputousmaisesti projektipäällikön määräämien kontrolliporttien tahdissa etenevän projektisyklin lisäksi mallissa tehdään tämän ytimen ulkopuolisesti iteratiivista työtä seuraavan vaiheen tasolla. Tavoitteena on selvittää työn alla olevassa vaiheessa tehtävän määrittelyn toteutuskelpoisuus ja siihen liittyvät riskit teknisestä näkökulmasta. V-mallissa huomioidaan myös insinöörityön monialaisuus tekemällä eri alojen selvitystä rinnakkain riskien ja toteutuskelpoisuuden selvittämiseksi. Tämä teknisen suunnitteluprosessin varhaisempi mukaan tuominen on merkittävä ero vesiputoukseen. V-mallin vaiheet ja niiden vastaavuudet on esitetty kuvassa 2. Lisäksi kuvassa on otettu mukaan myös asiakkaan suorittama hyväksymistestaus. [Forsberg and Mooz 1991]

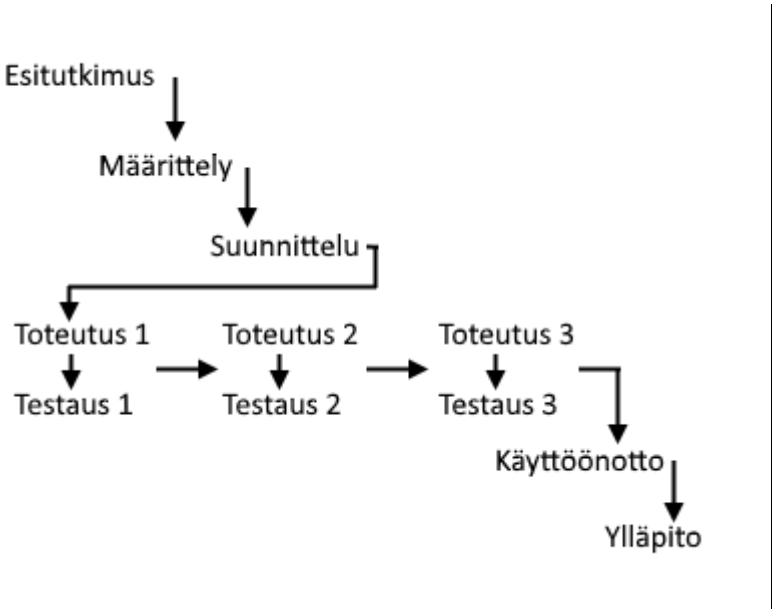


Kuva 2: V-malli Haikalaa ja Mikkosta [2011] mukaillen.

V-malliin sisältyvä testauksen jaottelu eri määrittely- ja suunnitteluvaiheita vastaavaksi auttaa myös testausvastuun jakamisessa. Alimmalla tasolla yksikkötestien parissa kehittäjät voivat itse suorittaa testausta, kun sieltä noustaan ylemmäs vastuu voi siirtyä erilliselle testausryhmälle, ulkopuolisille testauskonsulteille, tilaajan henkilökunnalle ja lopulta tilaajan laadunvarmistusorganisaatiolle. Ohjelmistotuotannossa yhtenä testauksen keskeisistä ajatuksista on nopea virheiden löytäminen kustannuksien rajaamiseksi. V-malli visualisoi myös tämän periaatteen osoittamalla, kuinka pitkälle taaksepäin suunnittelu- ja testausvaiheita on mentävä virheen havaitsemiseen johtaneen testausvaiheen perusteella. [Haikala ja Mikkonen 2011]

2.4 Inkrementaalinen kehitys

Projektin koostuessa useasta peräkkäisesti pienemmästä projektista puhutaan inkrementaalisesta kehitysmallista. Jokaisessa osaprojektissa eli inkrementissä tyypillisesti toteutetaan joitakin lisäominaisuuksia edellisessä inkrementissä tuotettuun ohjelmistoversioon. Eri ominaisuuksien toteuttamisen jakautuminen projektin inkrementtien välille, samoin kuin inkrementtien määrä ja aikataulu on suunniteltu projektin alkuvaiheessa, ennen varsinaisen toteutustyön aloittamista. Inkrementaalinen kehitysmalli on esitetty kuvassa 3. [Mills 1980]



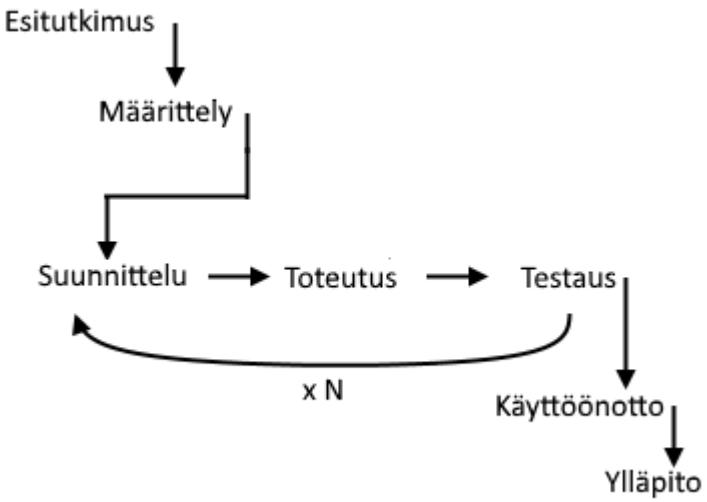
Kuva 3: Inkrementaalinen kehitys Haikalaa ja Mikkosta [2011] mukaillen.

Inkrementaalinen kehitysmalli vastaa vesiputoustyyppisten mallien ongelmiin. Pyrkimällä tuottamaan toimiva, joskin pienempi, kokonaisuus varhaisemmin, myös projektin loppuvaiheen riskit realisoituvat paljon aikaisemmin. Samoin testaus on aloitettava yhden

vesiputouksen mallia aiemmin. Nämä muutokset poistavat epävarmuutta ja toisaalta antavat asiakkaalle mahdollisuuden tutustua valmiisiin osakokonaisuuksiin. Asiakas pystyy vakuuttumaan projektin etenemisestä sekä mahdollisesti tarkentamaan vaatimuksiaan kokeiltuaan tuotetta vesiputousmallin projektia aikaisemmassa vaiheessa. Lyhyempi inkrementti antaa myös mahdollisuuden hyötyä organisaatiossa edellisten inkrementtien aikana tapahtuneesta oppimisesta. Periaatteessa inkrementaalinen malli mahdollistaa myös muutokset kesken projektin, jos aiempien inkrementtien kokemusten pohjalta päädytään muuttamaan tulevien inkrementtien tai ohjelmiston ominaisuuksien sisältöä. [Haikala ja Mikkonen 2011]

2.5 Iteratiivinen kehitys

Tietyn kehityssyklin toistuminen osana projektin rakennetta on iteratiivisten menetelmien ominaispiirre. Iteratiivinen malli on esitetty kuvassa 4. *Iteratiiviset ja inkrementaaliset kehitysmenetelmät* (IID) ovat olleet ohjelmistoprojekteissa käytössä ohjelmistotuotannon alkuajoista asti, kun esimerkiksi ilmailu- ja sotilasprojekteihin käytettyjä menetelmiä sovellettiin myös näiden projektien ohjelmisto-osiin. Kuitenkin varsinaisena ohjelmistoprojektien hallintamenetelmänä iteratiivinen kehitys formalisoitui vasta vastareaktiona korjaamaan määräävään asemaan päätyneen vesiputousmallin ongelmia, erityisesti vesiputousmallin heikkoutta ohjelmistoprojektien kompleksisen luonteen käsittelyssä. Iteratiivisissa menetelmissä hyväksytään laajan, ennen projektin toteutusta tapahtuvan määrittelyn mahdottomuus. Määrittelyä tehdään osana iteraatioita sitä mukaan, kun tietämys tarpeista kasvaa. Erityisesti vanhemmissa, vesiputouskaudella tai sitä ennen toteutetuissa iteratiivisissa projekteissa saattoi olla vesiputoustyylinen laaja, koko projektin kattava määrittelyvaihe, jota sitten tarkennettiin iteraatioissa. Nykyisissä ketterissä iteratiivisissa menetelmissä tämä osuus on kevyempi ja määrittelyn pääpaino on iteraatioissa. Vesiputousmalliin tehdyissä vertailuissa on havaittu, että kehityssyklin lyhyys ennustaa ohjelmistoprojektin onnistumista, näin ollen vesiputouksesta iteratiivisempaan malliin siirtyminen on usein perusteltua. [Larman and Basili 2003]



Kuva 4: Iteratiivinen kehitys Haikalaa ja Mikkosta [2011] mukaillen.

Inkrementaalinen malli on tässä erotettu iteratiivisesta kehityksestä iteraation sisäisen joustavuuden perusteella. Iteratiivisessa mallissa ei ole ennalta määrättyä eri ominaisuuksien jaottelua eri iteraatioihin ja iteraatioiden määräkin voi muuttua. Jokaiseen mallin iteraatioon otetaan mukaan uusia toteutettavia ominaisuuksia iteraation alussa vallitsevan tilanteen mukaisesti.

Mitä lyhyemmäksi iteraatiosykli muuttuu ja mitä enemmän määrittelyä ja suunnittelua tehdään osana jokaista iteraatiota, sitä lähemmäksi ketteriä kehitysmenetelmiä siirrytään. Useat ketterät menetelmät ovatkin iteratiivisia. Niissä korostuu seuraavan iteraation suunnittelu edellisen iteraation tuloksien pohjalta. Toisaalta iteratiivista kehitystä voidaan tehdä myös perinteisempien mallien, kuten vesiputouksen avulla. Iteraation pituus voi vaihdella huomattavasti eri tarpeiden mukaan. [Haikala ja Mikkonen 2011]

3 Ketterät projektinhallintamenetelmät

Perinteisiä luvussa 2 esiteltyjä ensimmäisen aallon ohjelmistoprojektin hallintamenetelmiä seurasivat ketterät menetelmät. Tässä luvussa esitellään yleisesti ketterät menetelmät sekä tarkemmin Scrum ja Kanban ja niiden vaikutuksia käytännön ohjelmistokehitystyöhön. Kyseisiin menetelmiin verrattuna esim. Extreme Programming (XP) on ennemmin pienille ohjelmistokehitystiimeille tarkoitettu kokoelma toimintaohjeita ja työskentely- ja

viestintätapoja kuin varsinainen projektinhallintamenetelmä [Cockburn 2001]. Luvun lopuksi vertaillaan myös tutkimuskirjallisuudessa esitettyjen havaintojen avulla Scrumin ja Kanbanin vahvuuksia ja heikkouksia sekä heikkouksien kiertämistä eri menetelmiä yhdistelemällä. Projektinhallintamenetelmien sekoittumista ja nivoutumista yhteen havainnollistetaan kirjallisuudessa esitetyillä käytännön esimerkeillä.

3.1 Ketterä ohjelmistonkehitys

Peruskivenä ketterän ohjelmistokehityksen aikakaudelle voidaan pitää ns. ketterän kehityksen manifestia. Joukko ohjelmistotuotannon ja projektinhallinnan asiantuntijoita pyrki määrittelemään uuden ja paremmin ohjelmistotuotannon tarpeisiin soveltuvan tavan hallita projekteja [The Agile Alliance 2001]. Manifestin eli ketterän ohjelmistokehityksen julistuksen perusajatuksena ovat seuraavat arvotukset:

- Yksilöitä ja kanssakäymistä arvostetaan enemmän kuin menetelmiä ja työkaluja.
- Toimivaa ohjelmistoa arvostetaan enemmän kuin kattavaa dokumentaatiota.
- Asiakasyhteistyötä arvostetaan enemmän kuin sopimusneuvotteluja.
- Vastaamista muutokseen arvostetaan enemmän kuin pitäytymistä suunnitelmassa.

Ketterien menetelmien käyttöön liittyy usein kehitettävän ohjelmiston pitäminen jatkuvasti toimintakuntoisena *jatkuvan integraation* (Continuous Integration) menetelmillä [Haikala ja Mikkonen 2011]. Keskeistä ketterille menetelmille on työn mahdollisimman kevyt ja myöhäinen määritteleminen. Dokumentaatio pidetään minimissä, joten määrittelyä tehdään etukäteen vain sen verran kuin on tarpeen, jotta päästään työssä seuraava askel eteenpäin. Näin pyritään pääsemään tasapainopisteeseen, missä tekemisen helppous on mahdollisemman suuri mahdollisimman pienellä dokumentointiin ja hallinnointiin käytetyllä ajalla. Kun ohjelmiston rakennetta ei tarpeettomasti lukita etukäteen, on myöhemmin tulevien muutostarpeiden huomioonottaminen helpompaa, näin ollen näitä menetelmiä pidetään ketterinä. Ohjelmistojen abstraktin luonteen vuoksi on kokemuksen pohjalta havaittu, että hyvinkään perusteellinen etukäteen suunnittelut ja määrittely ei välttämättä johda myöhempien muutostarpeiden vähenemiseen, vaan niiden toteuttamisen hankaloitumiseen. Ketterissä menetelmissä yleisesti myös korostetaan kehitystiimin roolia ja vastuuta päättää ja arvioida asioita. Tämän mahdollistamiseksi ketterät menetelmät antavat paljon painoarvoa tehokkaalle ja monimuotoiselle tiimin sisäiselle viestinnälle [Cockburn 2001].

3.2 Scrum

Yksi varhaisista ketteristä ohjelmistonkehitysmenetelmistä on Scrum. Sen suunnittelijat Ken Schwaber ja Mike Beedle olivat myös Agile Manifeston tekijöitä. Scrum-nimitys tulee rugby-urheilulajin toimintatavasta, jossa pelaajat kokoontuvat yhteen kasaan pelitilanteiden alkaessa. Tämä yhteinen hetki yhdistettyinä lyhyisiin *kehityspyrähdyksiin* (Sprint) muodostaa Scrum-menetelmän perustan. Oleellista on myös jatkuva oman toiminnan kriittinen arviointi ja kehittäminen jokaisen pyrähdyksen päätteeksi. Scrum-menetelmässä kehitystiimillä on paljon valtaa arvioida ja määritellä tehtävää työtä itsenäisesti. [Schwaber and Beedle 2002]

Kirjassaan Schwaber ja Beedle [2002] käyvät läpi esimerkin Scrum-menetelmän osista ja siten tunnistettavista piirteistä, joista käytetään tässä tutkielmassa Haikalan ja Mikkosen [2011] suomenkielisiä käsitteitä. Perustyöskentelytapana on kehityksen pyrähdys, noin kahdesta neljään viikkoa kestävä kehitysjakso. Tilaajan tai liiketoimintaosaamisen edustaja eli *tuotteen omistaja* (Product Owner) määrittelee ja priorisoi *tehtäviä* (Task), joita hallinnoidaan *tuotteen työlistalla* (Product Backlog). Pyrähdyksen alkaessa kehitystiimi arvioi määriteltyjen tehtävien työmäärää ja ottaa siihen mukaan sen verran työtä, minkä ryhmän arvioidaan pystyvän saamaan valmiiksi pyrähdyksen aikana. Pyrähdyksen kuluessa tiimi itsenäisesti jakaa ja suorittaa työt. Scrumiin kuuluu päivittäinen hyvin lyhyt tiimin tilannekatsaus eli *päivän Scrum* -palaveri (Daily Scrum), jossa muutamalla lauseella käydään läpi jokaisen tiimiläisen työn tilanne. Lopuksi pyrähdykseen varatun ajan kuluttua tiimi katselmoi yhdessä tuotteen omistajan kanssa valmiiksi saadut työtehtävät ja arvioi kriittisesti tiimin työskentelyä kokonaisuutena parannuskohteiden löytämiseksi pyrähdyksen *katselmointikokouksessa* (Sprint Review) ja pyrähdyksen *arviointipalaverissa* (Retrospective). Pyrähdyksen aikana syntynyttä lisäarvoa kutsutaan inkrementiksi, mikä eroaa kohdassa 2.4 esitetyn inkrementin käsitteestä siten, että Scrumissa inkrementtien sisältöä ei suunnitella projektin alussa tai muutenkaan etukäteen, vaan projektiryhmä valitsee pyrähdyksen ja siten inkrementin tavoitesisällön aina uuden pyrähdyksen aluksi. Menetelmänä Scrumia voidaan myös pitää kohdassa 2.5 esiteltynä iteratiivisena mallina toistettavan pyrähdyksen käsitteen perusteella. Scrumissa pyritään minimoimaan kankeat hallinnointi- ja johtamiskäytännöt. Tiimin sisäistä toimintaa koordinoi *Scrum-mestari* (Scrum Master), joka pitää huolen siitä, että Scrumin käytäntöjä noudatetaan, auttaa saamaan tarvittavat päätökset aikaiseksi, poistaa työnteon edistymisen *esteet* (Impediments) ja vastaa myös sidosryhmien kanssa viestinnästä ja siten tiimin eristämisestä näiden yhteydenottojen työtä häiritsevältä vaikutukselta.

Toisaalta vaikka Scrum-menetelmän yhteydessä usein esitellään valmis työkalupakki kaikkien projektinhallintatehtävien suorittamiseksi, Scrum ei kuitenkaan suoranaisesti määritä mitään käytettävää työkalua, ainoastaan tehtävät ja tavoitteet, jotka työkaluilla täytyy tehdä [Cockburn 2001]. Taulukossa 1 on kuvattu Scrumin projektinhallintatehtävät ja esimerkkejä niihin käytettävistä työkaluista ja rooleista.

Tehtävä	Työkalu, rooli tai menetelmä
Työmäärän arviointi	*Suunnittelupokeri* (Planning Poker), Backlog refinement, koko kehitystiimi
Työmäärän määrittely	Henkilötyöpäivä, Story Point, Backlog refinement
Työlista	Tuotteen työlista, pyrähdyksen työlista, tehtävät
Sprintin tavoite	Sprint Goal, pyrähdyksen suunnittelupalaveri, *tiimin nopeus* (Velocity) kehitystiimi
Jatkuva parantaminen	Pyrähdyksen arviointipalaveri ja katselmointikokous
Sprintin sisäinen viestintä	Päivän Scrum, Standup Meeting
Tehdyn työn arviointi	Inkrementti, pyrähdyksen katselmointipalaveri, pyrähdyksen edistymiskäyrä, tehtävän valmistumisen ehdot
Käytäntöjen ja sääntöjen noudattaminen, päätöksenteko ja ongelmien poistaminen	Scrum-mestari
Uusin asioiden hyväksyminen työlistalle, tehtävien priorisointi	Tuotteen omistaja

Taulukko 1: Scrum-menetelmän välineitä [Schwaber and Beedle 2002; Schwaber and Sutherland 2017].

Scrum-menetelmässä tehtävät jaetaan tuote-, pyrähdys- ja päivätasolle. Kaikki tuotteeseen tulevat ominaisuudet kerätään tuotteen työlistaan. Siitä töitä arvioidaan ja otetaan tehtäväksi johonkin pyrähdykseen sijoittamalla ne *pyrähdyksen suunnittelukokouksessa* (Sprint Planning Meeting) jonkun tietyn *pyrähdyksen työlistaan* (Sprint Backlog). Päivän Scrum –palaverissa tiimi kokoontuu päivittäin jakamaan tehtävät seuraavalle vuorokaudelle ja synkronoimaan toimintansa. Tehtävien etenemistä seurataan *tehtävän valmistumisen ehdoilla* (Definition of Done) ja pyrähdyksen etenemistä esimerkiksi jäljellä olevan työmäärän päivittäistä kehittymistä kuvaavalla *pyrähdyksen edistymiskäyrällä* (Burndown Chart). [Schwaber and Beedle 2002]

3.3 Kanban

Ketterien ohjelmistonkehitysmenetelmien yksi suuntaus ovat ns. Lean-ajattelusta vaikutteita saaneet järjestelmät, kuten Kanban. Alun perin Lean-menetelmät kehitettiin teollisuuden ja erityisesti Japanin autoteollisuuden tarpeisiin 1980-luvulla, tavoitteena arvoketjun tiivistäminen tuotantovaiheita nopeuttamalla ja vähentämällä hukkaan meneviä resursseja ja aikaa [Poppendieck 2002]. Lean-menetelmissä korostuu työn tekeminen pyrähdyksien sijaan jatkuvana virtana, jossa keskeinen tekijä on aiemman vaiheen tuotantomäärän määrittäminen suoraan sitä seuraavan vaiheen tarpeella ja näin ylituotannon ja välivarastojen välttäminen [Gross and McInnis 2003].

Kanban-projektinhallinnassa tehtävät, joiden japaninkielisestä nimestä menetelmä on saanut nimensä ja joita projekteissa usein kutsutaan jollain tietyllä termillä, kuten tiketti, ovat keskeisessä roolissa tilaajan tai asiakkaan edustajan määritellessä ja priorisoidessa niitä. Tästä tehtäväkentästä projektitiimin jäsenet ottavat tehtäviä työn alle sitä mukaa, kun he saavat tehtäviä valmiiksi. Kanbanissa keskeisessä roolissa onkin näiden tehtävien elinkaari. Usein tehtävälle on määritelty eri vaiheita kuvaamaan sen tilan edistymistä. Taulukossa 2 on esitetty yksi tapa mallintaa tehtävän elinkaari, tämän esimerkin perustuessa luvussa 4 käsiteltävän case-projektin malliin.

Tila	Kuvaus
Todo	Odottaa työ alle ottamista
In Progress	Joku projektitiimistä työstää tehtävää
Ready for review	Tehtävä on valmis ja odottaa katselmointia
Done	Tehtävä on katselmoitu ja valmis
Impeded	Tehtävä on keskeytynyt ulkoisesta syystä

Taulukko 2: Kanban-tehtävän tilat.

Tehtävien tilat voivat vaihtua reaaliajassa projektitiimin niin halutessa. Kanbanissa tärkeimpänä suunnittelu- ja viestintätyökaluna on *tehtävätaulu* (Kanban Board), jolla seurataan tehtävien etenemistä niiden elinkaaren vaiheesta toiseen [Alqudah *et al.* 2017]. Tätä kautta koko tiimi on jatkuvasti tietoinen projektin ja sen yksittäisten tehtävien valmistumisesta. Tehtävätaululla toteutetaan ja visualisoidaan myös toinen Kanbanin tärkeä piirre ja ero perinteiseen Scrumiin, eli tietyssä vaiheessa olevien Kanban-tehtävien määrän yläraja eli kaistanrajoitus. Ladas [2008] vie tätä ajattelua vielä pidemmälle vertaamalla Kanban-projektia suljettuun talouteen, missä Kanban-korttien määrä ja eri työvaiheiden kaistanrajoitukset muodostavat rahan tarjontaa vastaavan vaikutuksen resurssien jakamiseen ja talouden kiertonopeuteen, joita voidaan optimoida näitä määriä säätämällä.

Taulukossa 2 esitettyä elinkaarimallia noudattavan projektin Kanban-taulu voi näyttää esimerkiksi taulukossa 3 esitetyn kaltaiselta.

Tekijä	Todo	In Progress	Ready for review	Done	Impeded
Matti	#15 client-listan alustus	#22 päivitä paikkataulu			
Maija		#10 valmistele 2.4 julkaisu		#6 muuta resurssisivun nappitekstit	
Kalle	#24 etsi ja tuhoa havainto bug-442	#19 päivitä tietokanta-ajurien versio	#17 lisää karttasivun hakutoiminto		#12 lisää liityntä ulkoiseen karttajärjestelmään

Taulukko 3: Esimerkki Kanban-taulusta.

Kanban soveltuu projekteihin, joissa tulee paljon ja joskus hyvinkin nopealla aikataululla uusia tehtäviä. Näin tilaaja ei joudu odottamaan esimerkiksi Scrum-tyyppisen projektin tapaan seuraavan kehitysjakson alkamista saadakseen jonkun tehtävän työn alle. Myös uusien vaatimuksien ja siten tehtävien mukaan ottamisen vaatima hallinnollinen työmäärä on hyvin pieni, koska asiakkaan edustaja voi itse luoda projektitiimille suoraan tehtäviä siten, että ne ovat välittömästi tiimin työstettävissä. [Alqudah *et al.* 2017]

3.4 Scrumin ja Kanbanin eroja sekä yhteiskäyttö samassa projektissa

Käytännön ohjelmistokehitystyössä Scrum-menetelmää on tavanomaista muokata projektin tarpeisiin tai siksi ettei organisaatio ole valmis toteuttamaan menetelmää sellaisenaan. Ilmiö on niin yleinen, että sitä kuvaamaan on vakiintunut ammattisanastoa. Osittaista Scrum-menetelmän käyttöä kutsutaan yleisesti Scrumbut-termillä [Haikala ja Mikkonen 2011]. Jos Scrumia on yhdistetty Kanbaniin, puhutaan Scrumbanista [Ladas 2008]. Erityisesti vaikka suurin osa varsinkin suurien projektien ongelmista johtuu virheistä vaatimuksissa, ei Scrum ota kantaa vaatimustenmäärittelyyn [Haikala ja Mikkonen 2011]. Suunnittelun ja ohjelmistoarkkitehtuurin kevyt rooli on yksi ketterien menetelmien heikkouksista [Dybå and Dingsøyr 2007]. Scrum ei sisällä tuotteen työlistaa kehittyneempiä vaatimustenhallintatyökaluja ja jättää koko aihepiirin tuotteen omistajan murheeksi tai ylemmän tason sovelluskehyksen tai muun tukirakenteen määrittelemäksi [Haikala ja

Mikkonen 2011]. Ketterien menetelmien vaikutusta tieteellisten ohjelmistoprojektien testaukseen ja vaatimusten käsittelyyn selvittänyt tutkimus päätyi samaan tulokseen, sillä vaikka Scrumilla oli muuten myönteinen vaikutus tutkittuihin projekteihin, ei vaatimusten käsittelyn osalta vastaavaa parannusta havaittu [Sletholt *et al.* 2011].

Vaikka Scrum nähdään yleisesti yhtenä merkittävänä ketteränä menetelmänä, voivat jotkut sen rakenteet, kuten timebox-aikarajoituskäytäntö kehityspyrähdyksissä, johtaa ketteryyden sijasta kangistumiseen [Haikala ja Mikkonen 2011]. Ladas [2008] esittelee menetelmän, jossa yhdistämällä Scrumiin tiettyjä Kanban-piirteitä tästäkin puolesta voidaan tehdä ketterämpi. Hän esittää Kanbanin pull-ominaisuuksien tuomista Scrumiin tehtävien tiloja ja tehtävätaulun jaottelua lisäämällä sekä kaistanrajoittimia optimoimalla, jolloin pyrähdyksen sisällä työn tekeminen varastoon ja siten hukka pienenee ja tehtävien läpimenoaika tehtävätaululla paranee. Esimerkkiä Scrumin laajentamisesta ja käyttämisestä hyvin monenlaisissa projekteissa ohjelmistokehityksen lisäksi käsitellään Terleckan [2012] artikkelissa. Vaikka sen näkökulma on tietojärjestelmien ylläpitotoiminnassa, voidaan siitä nähdä, miten projektin erityistarpeiden pohjalta yhdistetään erilaisia hallintamenetelmiä.

Laaja tilastollinen tutkimus Scrumin ja Kanbanin vaikutuksista ohjelmistoprojektien hallinnan eri ulottuvuuksien onnistumiseen ei tuonut tilastollisesti merkittävää etua kummallekaan järjestelmälle, vaikka joitakin viitteitä Kanbanin edusta projektin aikataulunhallinnassa tulikin esille [Lei *et al.* 2017]. Tutkitut ketterät menetelmät vaikuttivat kuitenkin yleisesti auttavan projektien onnistumista, joten menetelmä voidaan valita projektin käytännön tarpeiden mukaan kummankin menetelmän hyödyllisyyden ollessa yhtä hyvä [Lei *et al.* 2017]. Ketterien menetelmien valintakriteereitä käsittelevässä tutkimuksessaan Alqudah ja muut [2017] eivät myöskään löytäneet selkeää voittajaa, joten tiimeille on suositeltavaa tehdä valinta sen mukaan, pidetäänkö tärkeämpänä Scrumin rooleja, vastuita ja kehityspyrähdyksien ennustettavuutta vai Kanbanin joustavampaa tiimin kokoa, nopeampaa tehtävien toteuttamista pienempinä kokonaisuuksina ja priorisointisykliä. Näiden tuloksien perusteella myös Scrumin ja Kanbanin osien yhdisteleminen projektin tarpeiden mukaisesti räätälöidyksi hallintamenetelmäksi vaikuttaa hyödylliseltä. Ketterien ja Lean-menetelmien suoraa kvantitatiivista vertailua ei ole juuri tehty muille kuin XP:lle tai yleisesti ketterille menetelmille, minkä lisäksi tutkimuksissa oli puutteita ja ne keskittyivät usein arvioimaan ketterän menetelmän käyttöönottoa tavallisen käytön sijaan [Dybå and Dingsøyr 2007].

Esimerkki yllä kuvatun tyyppisestä muutostilanteen tutkimuksesta on Sjøbergin ja muiden [2012] artikkeli. Siinä tutkitaan läpimenoaikoja, virheiden määriä ja muita tavanomaisia metriikoita projektissa, joka siirtyy ensin vesiputouksesta Scrumiin ja siitä Kanbaniin. Tuloksista havaitaan useimpien mitattavien arvojen paranevan projektin siirtyessä

Scrumista Kanbaniin. Tämän tutkielman kannalta mielenkiintoista on näiden vertailukohtien taustatilanteet sekä niiden vaikutus mittaustuloksiin, mistä voidaan päätellä eri menetelmien soveltuvuutta erilaisiin tilanteisiin. Kuten kirjoittajat huomauttavat, Scrum-jakson aikana saavutettu kokemus ketteristä menetelmistä saattoi vaikuttaa Kanbanin Scrumia sujuvampaan käyttöönottoon. Erityisesti Scrum-vaiheen lopun mittaustulokset olivat pääsääntöisesti Kanban-jakson tasolla, jolloin Scrumin rinnalle oli jo otettu käyttöön joitain pull-tyyppisiä Kanbanista lähtöisin olevia käytäntöjä. Tästä voidaan päätellä, että tässä projektissa tehokkuutta ei parantanut varsinaisesti Kanban menetelmänä, vaan joidenkin sen piirteiden tuominen projektiin. Projektissa oli paljon keskeytyksiä, ylimääräisiä tehtäviä, virhekorjauksia ja muuta perinteisestä ohjelmistokehityksestä poikkeavaa työtä. Näin ollen vaikuttaa siltä, että Scrumille ominainen pyrähdyksen rauhoitus ei onnistunut, mutta puolestaan Kanbanin piirteet auttoivat. [Sjøberg *et al.* 2012]

Terlecka [2012] kuvailee yhden esimerkin Scrumin ja Kanbanin yhteiskäytöstä projektissa. Vaikka hänen kuvaamansa projektinhallintamenetelmä kehitettiinkin tietojärjestelmän ylläpitoprojektissa, Terleckan mukaan sitä on käytetty myöhemmin myös ohjelmistoprojekteissa. Sjøbergin ja muiden [2012] tapaan Terlecka esittää havainnon, että Scrum soveltuu paremmin kehitysprojekteihin ja esimerkkiprojektin ylläpidon suuren suhteellisen työmäärän takia tämä johtikin Kanban-tyyppisten pull-ratkaisujen ottamiseen projektin. Tässä esimerkissä on erityistä myös huomio siitä, että pelkän erillisten projektinhallintatyökalujen mukaan ottamisen lisäksi näistä voi yhdistellä uudenlaisia menetelmiä, tässä tapauksessa päivän Scrum ja pyrähdyksen suunnittelu oli yhdistetty päivän Scrumia laajemmaksi päivittäiseksi suunnittelupalaveriksi. Näin pyrähdyksettömään Kanbaniin huonosti sopiva, mutta kehitysryhmän tarpeelliseksi kokema pyrähdyksen suunnittelu on saatu mukaan vastaamaan työmäärien ja lähitulevaisuuden työn arvioinnista. Toinen vastaava esimerkki on Kanbanissa koettu etenemisen tunteen puuttuminen ja siihen vastaaminen Scrumin pyrähdyksen katselmoinneilla. Samoin kuin edellä suunnittelun kohdalla, tässäkin työkalu on mukautettu Kanbanin pyrähdyksettömyyteen sitomalla katselmoinnit Scrum-tyylisen timeboxingin sijaan tuoteomistajan päättämien suurempien kokonaisuuksien valmistumiseen. Pyrähdyksettömyyden aiheuttaman aikataulupaineen puuttumisen kehitysryhmä ratkaisi lisäämällä joillekin tehtävillä aikarajoja. Muuten esimerkkiprojektissa on tavanomaisemmin poimittu tiettyjä Scrumin piirteitä vastaamaan kehitysryhmän Kanbanissa kokemiin heikkouksiin. Työmäärien arvioinnista oli sen hankaluuden takia tässä projektissa luovuttu, tuoteomistajan ohjatessa projektia pelkästään tehtäväkorttien määrän perusteella. Kanban on kuitenkin projektin ytimenä ylläpidon ja yllättävien töiden suuren osuuden johdosta, erityisesti työn virta pysyy sujuvana kaistanrajoituksien takia. Scrumista mukaan tuotuja piirteitä on mukautettu, jotta ne soveltuvat Kanban-maailmaan, kuitenkin säilyttäen alkuperäisen tarkoituksensa. [Terlecka 2012]

Yllä esitetyn pohjalta voidaan tehdä muutamia havaintoja Scrumin ja Kanbanin yhdistelemisestä. Scrum nähdään usein hyvin tuotekehitykseen soveltuvana, kun vaatimushallinta ei ole monimutkaista ja erityisesti, jos työtä voidaan tehdä rauhassa ulkoisilta häiriöiltä ja vaikuttamisyrityksiltä, minkä vähimmäisvaatimuksena voidaan pitää kehityspyrähdyksen rauhoittamista. Kanbanin vahvuutena on nopea reagoiminen muuttuviin tai uusiin vaatimuksiin ja käytännössä usein myös nopeampi läpimenoaika. Scrumin parantelussa Scrumban-suuntaan on usein kyse juuri joidenkin Kanbanin pull-ominaisuuksien tuomisesta mukaan projektinhallintaan. Toisaalta Kanbaniin on Scrumista tuotu työn jaksottamiseen ja osakokonaisuuksien valmistumiseen ja esittelyyn liittyviä ominaisuuksia.

4 Case-esimerkki projektinhallintamenetelmien nivoutumisesta

Ohjelmistoprojektissa käytössä olevassa hallintamenetelmässä voi olla piirteitä useista erilaisista projektinhallintajärjestelmistä, kuten aiemmissa luvuissa esitettiin. Tätä eri projektinhallintamenetelmien yhteen nivoutumista tarkastellaan tässä luvussa käytännön tasolla case-esimerkin avulla. Esimerkkiprojektin esittelyn pohjalta on tehty tarkempi kuvaus projektinhallintamenetelmistä sekä arvioitu menetelmiä ja niiden yhteen nivoutumista aiemmissa luvuissa tehtyjen havaintojen perusteella.

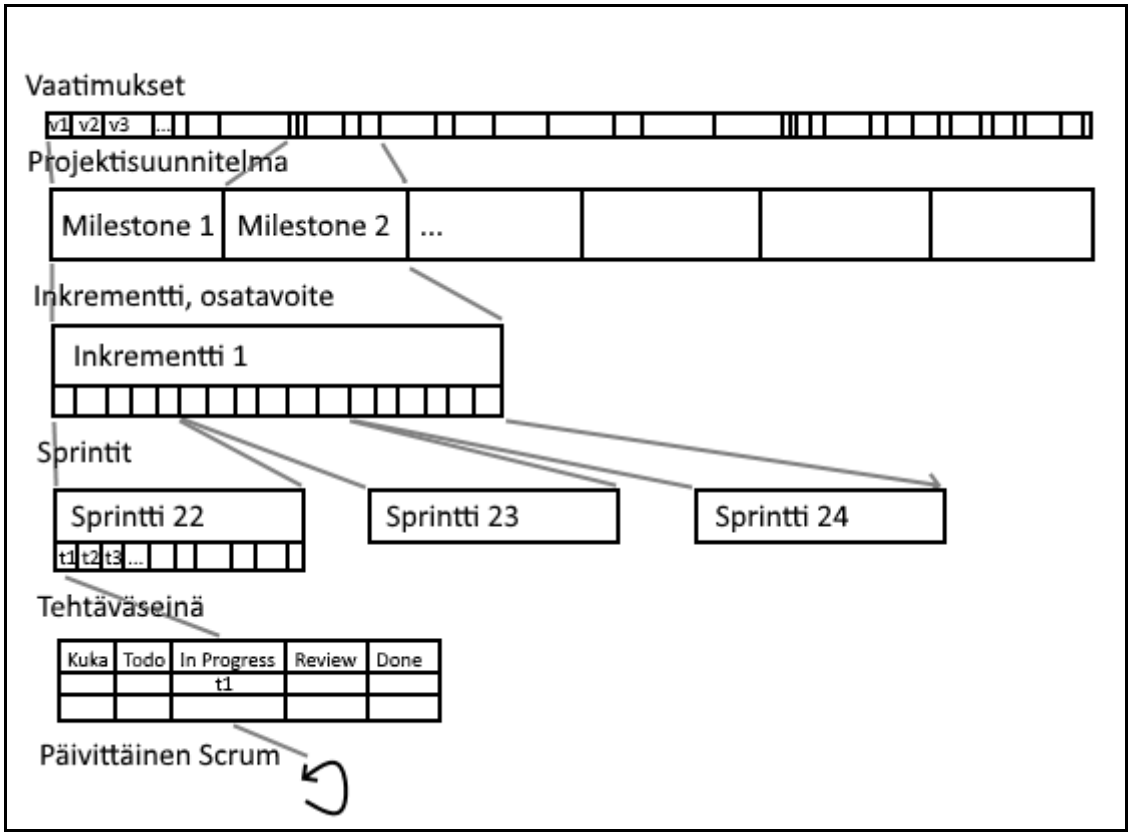
4.1 Esimerkkiprojektin kuvaus

Esimerkkiprojektissa suurehko suomalainen ohjelmistotalo toimitti laajan verkossa toimivan asiointijärjestelmän julkisen sektorin tilaajalle. Projekti sisälsi sekä front- että backendkehitystyötä ja sitä voidaan pitää tavanomaisena ohjelmistoprojektina. Toteutusryhmän koko vaihteli hieman, mutta oli alle 10 henkilöä. Projektissa sekä laajennettiin aiempaa asiakkaan järjestelmää että tehtiin kokonaan uusia asiointimahdollisuuksia. Kehitettävä järjestelmä oli ollut pitkään käytössä ja siihen oli vuosien varrella lisätty ominaisuuksia erilaisilla toteutusmenetelmillä. Osittain tämä johtui ohjelmistoon kohdistuneesta vaatimuksesta noudattaa lukuisia sovellusaluetta koskevia lakeja, niiden muutoksia sekä muuttuvaa oikeuskäytäntöä tuomioistuimien ennakkotapausten pohjalta. Tämä toi projektin kehitykseen myös aikataulupaineen, sillä ohjelmiston muutoksien oli oltava toiminnassa samaan aikaan lakien muutoksien kanssa. Kokonaisuuden toimivuutta ja yhtenäisyyttä puolestaan lisäsi korkean tason järjestelmän arkkitehtuuridokumentaatio ja sovelluskehys.

Teknisesti asiointisovellukset toimivat Oraclen Weblogic-palveluun rakennetussa tuotantoympäristössä. Järjestelmä sisälsi useita erilaisia asiointimahdollisuuksia, joille jokaisella oli oma käyttöliittymäsovelluksensa. Järjestelmästä oli liittymiä useisiin ulkoisiin järjestelmiin, joiden avulla tietoa sekä haettiin että jaettiin eteenpäin. Ulkoisten järjestelmien liittymät oli toteutettu hyvin vaihtelevilla menetelmillä.

4.2 Projektinhallintajärjestelmien yhteen nivoutuminen esimerkkiprojektissa

Järjestelmän kehitystyötä hallittiin useilla erilaisilla työkaluilla ja prosesseilla. Karkeasti nämä voidaan jakaa tilaajan hallinnointia tukeviin järjestelmiin, toimittajan hallintaprosessiin ja tuotantotiimin oman toiminnan hallintaan. Projektinhallinnan eri tasot on esitetty kuvassa 5.



Kuva 5: Case-projektin hallintamenetelmät.

Tässä projektissa on tunnistettu kolmenlaisia järjestelmiä. Ylimpänä tasona on tilaajan suunnitteluun ja määrittelyyn käyttämät menetelmät. Vaatimusten määrittely toteuttaa projektissa tarpeen sopia tarkasti toteutettavista asioista tilaajan ja toimittajan kesken, niiden jaksotuksesta sovitaan puolestaan projektisuunnitelmassa. Nämä ovat selkeästi perinteisiä, vesiputousmallisia hallintamenetelmiä, vaikkakin yhden vesiputouksen sijaan te-

kemistä onkin jaksotettu. Oikeastaan osa projektin määrittelystä voidaan katsoa, sen sovellusala huomioiden, tulleen tehdyksi jo ministeriön ja lainsäätäjän työnä. Askel ketterämpiä menetelmiä kohti tehdään inkrementtijaossa. Vaikka niistäkin on periaatteessa tehty suunnitelma projektin aluksi, voi toimittaja järjestellä niiden sisältöä toteutuksen ja kehitystiimin tarpeita ja vahvuuksia huomioiden. Varsinaista V-mallin mukaista formaalia eri testausvaiheiden takaisinkytkentää ei kuitenkaan ole. Alemman tason menetelmissä ketteryys korostuu. Projekti etenee Scrum-maisesti kehityspyrähdyksien ja päivittäisen Scrum-palaverin tahdittamana, mutta työtä hallinnoidaan Kanban-tyylisellä tehtäväseinällä, jolle asiakas voi myös tuoda uusia tehtäviä. Näin ollen puhtaan Scrumin yksi peruspilari, eli pyrähdyksen rauhoittaminen puuttuu ja on korvattu Kanbanille tunnusomaisella toimintatavalla.

4.3 Käytettyjen hallintamenetelmien arviointi

Hallintamenetelmät vastaavat johonkin projektinhallintatarpeeseen. Arvioitaessa niitä tästä näkökulmasta nähdään, että sekä ylimmän että alimman tason hallintamenetelmiä eli vaatimusmäärittely, projektisuunnitelma, inkrementit ja Kanban-tehtäväseinä on lähtöisin enemmän projektin asiakkaan tarpeista. Toisaalta jako kehityspyrähdyksiin ja päivittäinen Scrum-palaveri ovat projektin kehitystiimin ajankäytön ja työnohjauksen tarpeista lähtöisin. Tämä sopii sinänsä Scrumista aiemmissa luvuissa tehtyihin havaintoihin, joiden mukaan Scrum-tyyppisiä hallintamenetelmiä pidetään juuri kehitystiimien itsensä toivomina ratkaisuina.

Projektin hallintamenetelmien valinnoissa näkyvät myös eri menetelmien heikkoudet ja pyrkimykset ratkaista ne. Scrum-tyylisen kehityksen heikkoutena on vaatimusten käsittelyn puute. Tähän haasteeseen vastasi projektin tilaajan käytössä olevat korkean tason järjestelmäarkkitehtuurimäärittely ja sovelluskehys. Lisäksi asiakas ja toimittaja sopivat vaatimuksista hyvin varhaisessa vaiheessa ja näin ollen varsinaista tuoteomistajan tekemää Scrumille tyypillistä vaatimusten parantamista ja tarkentamista ei ollut laajasti käytössä. Tässä näkyi myös Scrumin kritiikki sen ketteryyttä heikentävistä puolista, sillä projektissa vaatimukset ja osittain inkrementit ja sprintitkin olivat asiakkaan korkealla tasolla määrittelemiä, ei toteutustiimin valittavissa.

Yhteenvetona kehitystiimin toiminta oli ketterien periaatteiden mukaista, sekoittaen tiettyjä Kanbanin piirteitä Scrumiin. Tilaaja puolestaan toisaalta määritteli asioita mahdollisimman paljon ja tarkasti etukäteen, mutta toisaalta myös halusi vaikuttaa suoraan Sprinttien sisällä kehitystyöhön käyttämällä hyväksi Kanban-tyyppistä tehtäväseinää. Projektin tilaaja pyrki saamaan sekä klassisen vesiputousmallin että ketterämmän Kanban-kehityksen hyödyt, kun kehitystiimi puolestaan pyrki toimimaan tuotekehityksen kannalta mukavassa Scrum-henkisessä projektissa.

5 Yhteenveto

Luvuissa 2 ja 3 tunnistettiin ja esiteltiin erilaisia projektinhallintamenetelmiä ja niiden piirteitä sekä avattiin projektien sisällä tapahtuvaa hallintamenetelmien yhteen nivoutumista käsittelevää tutkimusta. Työssä pitäydyttiin lähinnä kirjallisuuden pohjalta toteutetussa eri menetelmien piirteiden kvalitatiivisessa arvioinnissa, sillä menetelmien kvantitatiivisesta vertailusta materiaalia oli heikommin saatavilla ja tutkielman kokoa oli syytä rajata. Hallintamenetelmien piirteitä kartoitettiin myös alan perusteoksista ja oppikirjoista, eri menetelmien sekoittumista ja vertailua tutkimuskirjallisuudesta.

Arvioitaessa esimerkkiprojektia vertaamalla sitä aiemmin tunnistettuihin ominaisuuksiin ja kirjallisuudessa esitettyihin esimerkkeihin voitiin luvussa 4 havaita sen sisältävän selvästi piirteitä useammasta eri projektinhallintamenetelmästä sekä niihin kirjallisuudessa liitetyistä ilmiöistä. Esimerkkiprojektissa tuli esille myös se, miten erilaiset asiakkaan ja projektitiimin tarpeet johtivat erilaisten projektinhallintamenetelmien piirteiden esiintymiseen projektissa, johtaen menetelmien yhteen nivoutumiseen uudeksi kokonaisuudeksi. Näin ollen havainnot esimerkkiprojektista vastasivat kirjallisuudessa esitettyjä hallintamenetelmien piirteitä sekä niiden sekoittumiseen johtavia syitä, samoin menetelmien yhdistymisen ratkaisut ja seuraukset olivat aiemmissa luvuissa esitettyjen kaltaisia. Työssä oli päätarkoituksena tutkia ohjelmistoprojektin hallintamenetelmien sekoittumista projektissa, joten koska esimerkkiprojektissa ilmiö havaittiin kirjallisuudessa kuvailtua vastaavasti, voidaan tätä osaa työstä pitää onnistuneena. Toisaalta esimerkkiprojekti on hyvin tavanomainen ohjelmistoprojekti, ja koska monet ketterien menetelmien puutteet ja rajoitteet ovat laajasti tiedossa, on tämä voinut johtaa kirjallisuudessa esitettyjen havaintojen mukaisiin ratkaisuihin myös tässä projektissa. Koska asiaa tutkittiin vain ulkoisten piirteiden avulla eikä esimerkiksi haastattelemalla avainhenkilöitä, emme myöskään voi tietää oikeita syitä esimerkiksi Scrum-piirteiden valinnalle projektin hallintamenetelmään. Vaikka näille Scrum-piirteille on myös tässä tutkielmassa esitetyt kehitystarpeiden perustelemat syyt, on niihin saatettu päätyä myös esimerkiksi projektitiimin autonomian ja yleisen ohjelmistokehityspiireissä vallitsevan Scrum-myönteisyyden ja Scrumin koetun miellyttävyyden takia.

Mielenkiintoinen jatko tutkimukselle olisi lisätä kvantitatiivinen lähestymistapa eri menetelmien ja niiden yhdistelmien vertailuun. Se tosin vaatisi myös laajempaa saatavilla olevaa tutkimusaineistoa ketterien menetelmien käytöstä ja sekoittumisesta ohjelmistoprojekteissa. Tässä työssä niin projektinhallintamenetelmien kuin esimerkkiprojektinkin piirteitä tutkittiin vain pinnallisesti. Näiden ulkoisten tunnusmerkkien perusteella tehtiin päätelmiä projektin tarpeista ja siitä, miten eri projektinhallintamenetelmät vastaavat nii-

hin. Tutkimusta olisi luontevaa jatkaa askelta syvemmälle selvittämällä eri projektinhallintamenetelmien piirteisiin johtaneet syyt haastattelemalla niiden tekijöitä tai kirjallisuudesta sekä vastaavasti selvittämällä case-projektin vastuuhenkilöiltä, mitkä syyt johtivat minkäkin projektinhallintamenetelmän piirteen valikoitumiseen projektiin. Olisi myös mielenkiintoista laajentaa tutkimusta ajallisesti projektien sisällä ja selvittää, muuttuvatko projektinhallinnan eri haasteiden ratkaisemiseen valitut menetelmät projektin elinkaaren vaiheen mukaan ja mitkä paineet näihin muutoksiin johtavat.

6 Lähdeluettelo

Alqudah, Mashal and Rozilawati Razali. 2017. A comparison of scrum and Kanban for identifying their selection factors. In*: Proc. of the 2017 6th International Conference on Electrical Engineering and Informatics*, 1-6

Anderson, David. 2010. *Kanban, Successful Evolutionary Change for Your Technology Business.* Blue Hole Press.

Cockburn, Alistair. 2001. *Agile Software Development: The Cooperative Game.* Addison-Wesley Professional.

Dybå, Tore and Torgeir Dingsøyr. 2008. Empirical studies of agile software development: A systematic review. *Information and Software Technology* 50, 9, 833-859.

Forsberg, Kevin and Harold Mooz. 1991. The Relationship of System Engineering to the Project Cycle. In: *Proceedings of the First Annual Symposium of National Council on System Engineering*, 57–65.

Gross, John and Kenneth McInnis. 2003. *Kanban Made Simple: Demystifying and Applying Toyota's Legendary Manufacturing Process*. AMACOM.

Haikala, Ilkka ja Tommi Mikkonen. 2011. *Ohjelmistotuotannon käytännöt.* Alma Talent.

Ladas, Corey. 2008. Scrum-ban. *http://leansoftwareengineering.com/ksse/scrum-ban. Luettu 1.4.2019*

Larman, Craig and Victor R. Basili. 2003. Iterative and incremental development: A brief history. *Computer*, 6, 47-56.

Lei, Howard, Farnaz Ganjeizadeh, Pradeep K. Jayachandran and Pinar Ozcan. 2017. A statistical analysis of the effects of Scrum and Kanban on software development projects. *Robotics and Computer-Integrated Manufacturing* 43, 59-67.

Mills, H.D. 1980. Incremental software development, *IBM Syst. J.*, 19, 4, 415–420.

Poppendieck, Mary. 2002. Principles of lean thinking.

http://www.leanessays.com/2002/11/principles-of-lean-thinking.html. Luettu 15.4.2019

Royce, Winston W. 1970. Managing the development of large software systems: concepts and techniques. In: *Proc. Of the IEEE WESCON.*

Schwaber, Ken and Jeff Sutherland. 2017. The Scrum Guide. https://scrumguides.org/scrum-guide.html. Luettu 15.4.2019.

Schwaber, Ken and Mike Beedle. 2002. *Agile Software Development with Scrum.* Prentice Hall.

Sjøberg, Dag I.K., Anders Johnsen and Jørgen Solberg. 2012. Quantifying the Effect of Using Kanban versus Scrum: A Case Study. *IEEE Software* 29, 5, 47 - 53.

Sletholt, Magnus, Jo Hannay, Dietmar Pfahl, Hans Benestad and Hans Langtangen. 2011. A Literature Review of Agile Practices and Their Effects in Scientific Software Development. In: *Proc. of the 4th International Workshop on Software Engineering for Computational Science and Engineering*, 1-9.

Terlecka, Katarzyna. 2012. Combining Kanban and Scrum - lessons from a team of sysadmins. In: *Proc. of Agile Conference (AGILE)*, 99-102.

The Agile Alliance. 2001. Manifesto for Agile Software Development. http://agilemanifesto.org. Luettu 1.2.2019

Unnati, Shah S. 2016. An Excursion to Software Development Life Cycle Models: An Old to Ever-growing Models. *ACM SIGSOFT Software Engineering Notes*, 41, 1