Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

MIKKO SYVÄLUOTO
AME14SR

Ohjelmisto- ja sovelluskehitysmetodit

Ohjelmisto- ja sovelluskehityshankkeiden projektinhallintamenetelmiä konepajaliiketoiminnassa

KONE- JA TUOTANTOTEKNIIKAN KOULUTUSOHJELMA
2014

Tekijä(t)
Syväluoto, Mikko Tapio

Julkaisun laji
Opinnäytetyö, AMK

Heinäkuu 2020

Sivumäärä
57

Julkaisun kieli
Suomi

Julkaisun nimi
Ohjelmisto- ja sovelluskehitysmetodit

Tutkinto-ohjelma
Kone- ja tuotantotekniikan koulutusohjelma

Opinnäytetyö tehtiin Alfa Laval Aalborg Oy:n tarjoamasta aiheesta tutkia ohjelmisto- ja sovelluskehityshankkeiden haasteellisuutta ja moderneja menetelmiä tuotekehityksessä. Tavoitteena oli saada läpileikkaus perinteisen suunnittelukeskeisen projektihallinnan ja ketterien menetelmien eroista. Menetelmiksi valittiin perinteinen projektikolmio ja vesiputousmalli, ketteriksi menetelmiksi valittiin Scrum-viitekehys ja Kanban-menetelmä niiden ollessa jo yleisestikäytössä teknologia-aloilla. Modernit menetelmät Scrum ja Kanban kuvataan teoreettisella tasolla ja niitä ei opinnäytetyön aikana empiirisesti kokeiltu.

Ohjelmistokehityksessä haasteena koettiin, miten päästäisiin laadukkaaseen lopputulokseen ja miten perinteisen konepajateollisuusyrityksen tulisi mukautua informaatioteknologian muutoskaaressa. Ratkaisutyökaluiksi valittiin työhön kaksi menetelmää, MVP (Eng. Minimum Viable Product) ja tekninen velka päätöksenteon tukena.

Lopputuloksena saatiin kirjallinen tutkielma, jossa esitetään perinteisen projektihallinnan haasteet ohjelmisto- ja sovelluskehityshankkeissa ja ketterien menetelmien teoreettinen kuvaus. Ketterien menetelmien Scrum ja Kanban viitekehyksistä ja käytännöistä esitettiin lopuksi vertailutaulukko. Johtopäätöksessä sekä pohdinnoissa esitetään menetelmien mukautuvuutta organisaatiossa, sekä ohjelmistojen projektiluonteisten haasteiden ratkaisemista ja ylittämistä asiakasyritysympäristössä.

Asiasanat
Projektikolmio, vesiputousmalli, tekninen velka, Scrum, Kanban, MVP

Author(s)
Syväluoto, Mikko Tapio

Type of Publication
Bachelor's thesis / Master's thesis

July 2020

Number of pages
57

Language of publication:
Finnish

Title of publication
Software development methods

Degree program
Bachelor of mechanical engineering and production technology

The topic of this bachelor thesis work was given by Alfa Laval Aalborg Oy, to study the challenges in modern software development projects. Aim of thesis was to have some common understanding of differences between traditional plan driven methods and modern agile methods. Selected methods consisted traditional Iron Triangle and waterfall models, for modern methods Scrum and Kanban were selected due their common usage in technology sector. Both modern methods Scrum and Kanban are presented only in theoretical level.

Challenges in software development projects from customer side is, how to reach preferable goal. How should traditional machine centric company align itself in modern technology change curve. Couple solutions are presented in thesis work to support decision making, these solutions includes technical debt and MVP-model Minimum Viable Product.

The conclusion of the thesis is to consider challenges in traditional project landscape and theoretical framework of modern agile methods. Result of agile methods are shown in table view to point out major differences between two agile project models. Considerations at the end shows adaptation of modern methods in customer company and introduces few solutions regarding software development project challenges.

Key words
Iron triangle, waterfall model, technical debt, Scrum, Kanban, MVP

SISÄLLYS

1 JOHDANTO 5
1.1 Asiakasyritys 5
1.2 Ohjelmistokehittämisen haasteet, tutkielman tavoitteet ja lopputulos 6
2 METODIT, MATERIAALIT JA MÄÄRITELMÄT 8
2.1 Tutkimusaineisto ja lähestyminen 8
2.2 Esimerkkiohjelmisto *”Site Audit Tool”* 8
2.3 Ohjelma, ohjelmisto vai sovellus 9
3 OHJELMISTON OSTAMISEN HAASTEITA 10
3.1 Kaikki muu on monimutkaista, paitsi koodaus 10
3.2 Ohjelmisto projektina 11
3.3 Perinteinen Projektikolmio 13
3.4 Perinteinen vesiputousmalli ohjelmistokehityksessä 14
3.4.1 Vesiputousmallin yhteenveto 19
3.4.2 Esimerkkiohjelmisto *”Site Audit Tool”* - Projektin alku 19
3.5 Ketterä malli ohjelmistokehityksessä 20
3.5.1 Esimerkki ohjelmisto *”Site Audit Tool”* - Kehitysvaihe 22
3.6 Tekninen velka 23
3.6.1 Esimerkkiohjelmisto *”Site Audit Tool”*- Tekninen velka 26
3.7 Ohjelmistoprojektin epäonnistuminen - onnistuminen 26
3.7.1 Esimerkkiohjelmisto *”Site Audit Tool”*- Tulos tai ulos 29
3.8 Yhteenveto 29
4 NYKYISET KETTERÄT MENETELMÄT 30
4.1 Agile-menetelmä 31
4.2 Agile-manifesti 32
4.3 Scrum menetelmä 33
4.4 Scrum-yhteenveto 37
4.5 Kanban-menetelmä 37
4.6 Kanban-yhteenveto 42
4.7 Scrum ja Kanban vertailutaulukko 44
4.8 MVP-malli (Minimum Viable Product) 45
4.9 MVP yhteenveto 48
5 LOPPUPÄÄTELMÄ 49
6 LÄHTEET 53
7 LIITTEET 55

1 JOHDANTO

Jossain vaiheessa enemmän tai vähemmän kaikki törmäävät digitaalisuuteen ja järjestelmien modernisointiin. Ohjelmisto- ja sovelluskehitys onkin yhteiskunnan kannalta tärkeää koska sen tulisi parantaa tehollisuuteen liittyviä alan tuotantomenetelmiä, työvälineitä, ympäristöjä ja teknologioita. Jos liiketoimintasi ei ole digitaalistenpalveluiden tuottamista kuten asiakasyrityksessä, nouseekin esille aivan uudenlaisia kysymyksiä. *Miksi* se koskettaa meitä, ja *miten* pääsemme haluttuun kokonaisuuteen tai ratkaisuun.

Miksi pitäisi jotain tehdä jätetään vähemmälle tarkastelulle aineistossa. Kysymys on enemmän liiketoiminnallinen asia, jolla haettaisiin jotain selkeää arvoa yritykselle.

Miten päästään arvoa lisäävään lopputulokseen, onkin paljon haastavampi ja monimutkaisempi kysymys. Tässä aineistossa pyritään konkretisoimaan asioita immateriaalisten ilmiöiden ympärillä ohjelmisto- ja sovelluskehityshankkeissa.

1.1 Asiakasyritys

Asiakasyritys on keskikokoinen liiketoimiyksikkö Suomessa, joka on osana suurempaa globaalia konsernia. Markkinasektoreina toimii pääasiassa meriteollisuus ja moottorivalmistajien maavoimalaitokset. Yrityksessä on kaikki tarvittavat toiminnot myynnistä käyttöönottoon ja huoltopalveluihin.

Tarkemmin kuvattuna asiakasyritys suunnittelee, valmistaa ja toimittaa asiakkaille lämmöntalteenottolaitteistoja kuten Höyry- ja kuumavesikattiloita, tankkeja, koneikkoja ja siirtoputkistoja. Projektit ovat systeemitoimituksia, joissa laitteet räätälöidään asiakkaan tarpeisiin sopiviksi. Laitteiden mitoitukset ja rakenneratkaisut tukeutuvat termodynamiikan perusteisiin sekä painelaitteiden suunnittelua ja valmistusta ohjaaviin standardeihin. Valmistusmenetelmät ja materiaalit ovat kehittyneet vuosien

saatossa, jos kohta liitäntämenetelmänä hitsaus on edelleen yksi eniten käytettyjä. Materiaalit ovat pääasiassa hiiliteräspohjaisia EN normien mukaan. Tuotteiden valmistusketju on hyvin perinteinen; suunnittelu tuottaa tarvittavat valmistuspiirustukset 2d muodossa, työnsuunnittelu osittaa laitteen osavalmistuksen työvaiheisiin, tilaa materiaalit ja varmistaa tarvittavan tuotantokapasiteetin.

Laitteiden suunnittelu tapahtuu Suomessa ja valmistus pääasiassa Euroopassa tai Aasiassa. Suunnittelu perustuu vakiintuneisiin, hyviksi koettuihin teknisiin ratkaisuihin sekä alihankintaketjun olemassa oleviin valmistusmenetelmiin. Laitteiden valmistusaika vaihtelee toimitettavan laitekokonaisuuden laajuudesta, materiaalien saatavuudesta, tuotantolinjan kapasiteetista noin yhdestä kahdeksankuukautta.

1.2 Ohjelmistokehittämisen haasteet, tutkielman tavoitteet ja lopputulos

Haasteet:

Perinteisen konepajaliiketoiminnan haasteena on ymmärtää ja pysyä mukana nykyisissä moderneissa ohjelmistokehitys menetelmissä. Ohjelmiston ostaminen ulkoa ja sen abstraktisuuden käsittäminen olisi helpommin hallittavissa, kun valmistusmenetelmiin ja mahdollisuuksiin olisi parempi läpinäkyvyys ja ymmärrys. Uusien perinteisten tuotteiden suunnittelu, on aiheuttanut painetta tuottaa myös digitaalisia sovellutuksia tukemaan perinteistä mekaanista ratkaisuja.

Ohjelmistojen ja sovelluskehityksen monimuotoisuuden vuoksi olisikin tärkeää tarkastella mitä ohjelmistokehittäminen tarkoittaa asiakasyritykselle.
Liiketoimintayksikkö keskittyy kuitenkin tuottamaan parhaan mahdollisen tuloksen tuotteilla ja palveluilla johon yrityksen ydinliiketoiminta on rakentunut.

Tavoitteet:

Opinnäytetyössä ei niinkään keskitytä mitä ohjelmistoja olisi tarjolla tai minkälaisilla ohjelmointi työkaluilla teollisuusyrityksen modernisointia pitäisi tuottaa.

Ohjelmistojen ja sovellusten kehityshankkeita tarkastellaan niiden projektimaisen uniikinluonteen kautta ja sekä sitä, miten päästäisiin laadukkaaseen lopputulokseen asiakasyrityksen tuotekehitysympäristössä.

Työssä kartoitetaan joitakin tunnettuja ja hyvin kirjallisesti dokumentoitua nykypäivän ilmiöitä ohjelmisto- ja sovelluskehitysalalla. Niiden pohjalta löydettyjä keskeisiä piirteitä tarkastellaan ja verrataan menetelmien kuvauksissa ja toteutuneesta asiakasyrityksen sovelluskehityshankkeesta *Site Audit Tool*.

Lopputulos:
Lopputuloksena on kirjallinen tutkimus, jossa pohditaan ohjelmisto- ja sovelluskehityshankkeiden päätöksentekoon ja projektihallintaan vaikuttavia syyperäisiä ilmiöitä ja ratkaisuja asiakasyritys ympäristössä.

Työn laajuus on rajoitettu vain asiakasyrityksen järjestelmien ja organisaation kykyyn toimia digitaalisissa hankkeissa. Rajoitteet johtuvat järjestelmien monimutkaisuuksista ja niiden yleistäminen vastaamaan kaikkia asiakasympäristöjä olisi melkein mahdotonta. Työssä ei myöskään tarkastella itse tehdyn tai ulkopuolisen toimittajan ohjelmistojen eroja ja käytäntöjä.

"*A Complex System is an organization, or a system in, composed of a set of heterogeneous elements whose local interactions are diverse, non-linear and are independent of centralized control or synchronization*" (Massotte, 2017)

Tutkimusaineisto on kirjoitettu viiteen eriosioon. Ensimmäinen osa-alue on käytännössä johdanto ja opinnäytetyön kuvaus. Toisessa osa-alueessa tarkastellaan miten tutkimusta lähestyttiin ja miten olemassa olevaa aineistoa hyödynnettiin.
Toiminnallisuudeltaan valmiista sovelluskehityshankkeesta asiakasyritykselle (*Site Audit Tool*), otettiin toteutuneita projektinhallinta ja tuotekehitys tapahtumia kirjallisen tutkimusaineiston tueksi.

Kolmannessa osiossa käsitellään ohjelmistojen hankintaan liittyviä asioita ja ilmiöitä. Erityisesti tarkasteltiin miten, monimutkaiset ohjelmisto ja sovelluskehitysprojektit sopivat perinteisiin asiakasyrityksen toimintamalleihin ja käytäntöihin.

Neljännessä osassa esitellään ketteriä projektimenetelmiä ja niiden vuorovaikutusta projektihallinta viitekehyksessä. Viidennessä ja viimeisessä osassa tehdään loppupäätelmä ja pohdintoja tutkimusaineiston tuloksien hyödyntämisestä asiakasympäristössä.

2 METODIT, MATERIAALIT JA MÄÄRITELMÄT

Opinnäytetyön kirjalliseen tutkimustyöhön on käytetty laadullista eli kvalitatiivista tutkimusmenetelmää. Teoreettisessa osuudessa pyritään hahmottamaan käsitteellisiä ilmiöitä, selityksiä ja rakenteita aiemman tutkimuskirjallisuuden pohjalta. Empiirisessä osuudessa pyritään esittämään konkreettisia havaintoja toteutuneesta ohjelmistokehitys hankkeesta.

2.1 Tutkimusaineisto ja lähestyminen

Laadullisiin osa-alueisiin kuuluu muun muassa, kirjallisen aineiston analysointi ja työn kirjoittajan havainnot työelämässä tapahtuneesta projektihallinnasta asiakasyrityksen ohjelmistohankkeiden parissa.
Havaintojen tueksi on kerätty konkreettisesti digitaalisia tallenteita kuten verkkopalaveri aineistot, muistiinpanot ja varsinaisen kehitysprojektin tuottama dokumentaatio. Laadullisen aineiston tulisi osoittaa, looginen ja yhtenäinen ilmiötä kuvaava tietokokonaisuus.

Tutkimustyön toisena aineistona on käytetty projektiosapuolta, joka asiakasyrityksessä on toteutettu varsinaisena sovelluskehityshankkeena.

2.2 Esimerkkiohjelmisto *”Site Audit Tool”*

Asiakasyritykselle tuotettiin toimittajan avulla puhtaasti räätälöity mobiilisovellus, jonka tarkoituksena oli standardisoida asiakasympäristössä tapahtuvaa laitteiden tai

järjestelemien tarkastamista. Kaupallisten ratkaisujen analysoinnin ja vertailun jälkeen todettiin, että omakustanteinen ja uniikkiratkaisu palvelisi kokonaisuutta tehokkaammin kuin muokata jo olemassa olevia järjestelmiä asiakasrajapinnan vaatimuksiin. Merkittäviä toiminnallisuuksia oli tietoa kuvaileva tieto (Eng. Meta data) kuvia ottaessa. Meta datan avulla raportoitavan työvaiheen tiedot esiintyisivät myös kuvien nimeämisformaatissa, jolloin tiedon tallentamiselle ja jatkojalostamiselle olisi järkevämpiä toteutusmahdollisuuksia moderneissa tietojärjestelmäratkaisuissa. Verkon ulkopuolella tapahtuva sovelluksen käyttö (Eng. Offline) oli merkittävässä roolissa, joten sovellusarkkitehtuurilliset ratkaisut perustuivat tähän käytäntöön.

Sovellus ja ohjelmisto itsessään oli hyvin perinteinen, helposti lähestyttävä ratkaisu pilvipalvelumallista Software as a Service (SaaS). Teemana oli lähtökohtaisesti tehdä taustapalvelin (Eng. Backend) ja päätelaitteen raportointi käyttöliittymä (Eng. Frontend). Erityispiirteenä ohjelmistossa oli asiakasyrityksen tapa tuottaa projekti ja tuotemateriaalia, jotka ovat hyvin yksilöllisiä ja seikkaperäisiä.

2.3 Ohjelma, ohjelmisto vai sovellus

Kyseisiä termejä käytetään kansankielellä melko vapaasti kuvailemaan erilaisia ilmiöitä tietojärjestelmäratkaisuissa tai laitteistojen toiminnallisuuksien kuvauksissa. Aihe alueiden yksinkertaistamiseksi tämän lopputyö sisältö viittaa seuraaviin, ei virallisesti dokumentoituihin selityksiin termien teknisestä erosta lukijalle.

Ohjelma – Ohjelmointikieltä, joka sisältää joukon tai yksittäisen ohjeen. Päätelaitteisto suorittaa ohjelman seuraamalla ohjeistusta ja tuottaa sen kuvailemat tapahtumat tai toiminnot.

Ohjelmisto - voi mahdollisesti koostua yhdestä tai useasta ohjelmasta, jotka suoritetaan päätelaitteistossa. Termillä kuvaillaan useasti joitain teknistä kokonaisuutta, jota jokin laitteisto tai erilliset laitteistot suorittavat.

Sovellus - on käyttäjille suunnattu ohjelmisto, joka useasti sisältää joukon koordinoituja ohjelmia. Näiden ohjelmien tarkoitus on suorittaa käyttäjän valitsemia toimintoja tai tehtäviä, sovellus normaalisti vaatii järjestelmäohjelmiston toimiakseen.
Tietojärjestelmissä käytettävät modernit konttiratkaisut (Eng. container, docker) tällöin sisältävät järjestelmäkehyksen ja rajapinnan, jossa sovellus ajetaan itsenäisesti riippumatta kohde järjestelmästä.

3 OHJELMISTON OSTAMISEN HAASTEITA

Osiossa käsitellään yleisestitunnistettuja ilmiöitä tai teemoja ohjelmisto- tai sovelluskehitysprojektien hankkimisesta. Useasti ne kulminoituvat tiettyihin lainalaisuuksiin ohjelmisto ja sovelluskehityshankkeissa, miten saataisiin aikaan laadukas toteutus.

3.1 Kaikki muu on monimutkaista, paitsi koodaus

Informaatiotekniikka ympäröi jo lähes kaikkea ihmisen joka päiväisessä arjessa niin vapaa-ajalla kuin työelämässä. Samalla kun tekniikka kehittyy harppauksin eteenpäin, luodaan tarve kehittää ja tuottaa entistä monimutkaisempia järjestelmiä, ohjelmistoja ja sovelluksia.
Suuntaus ei ole mitenkään tuore mutta kun asiaa tarkastellaan perinteisen konepajateollisuuden näkökannalta, se voi olla haastava.

Tarkasteltaessa ohjelmisto ja sovelluskehitysprojekteja asiakasyrityksen puolelta ne voivatkin vaikuttaa monimutkaisilta niiden epämääräisen luonteen ja abstraktisuuden vuoksi.

Haasteellisuutta voidaan tarkastella muun muassa kirjallisesta aineistosta (Xia, 2004) informaatioteknologian kehitysprojektit (Eng. Information System Development Projects, ISDP). Tutkimusaineistossa akateemikot kuvasivat kaksi ulottuvuutta monimutkaisuuden tulkitsemisessa.

- Organisaatiolliset muuttujat

- Teknologiset muuttujat
    - Teknologiset muuttujat jaettiin vielä erikseen kahteen eriluokkaan, rakenteelliset ja dynaamiset.

Keskittyessään pelkästään organisaation muuttujiin Maylor havaitsi hallinnollisen monimutkaisuuden sisältävän viisi peruselementtiä: tehtävä, organisaatio, toimitus, päätöksentekijät ja projektitiimi. (Maylor, 2008)

Informaatiotekniikan kehitysprojekteja ympäröikin siis muuttujien verkosto, ohjelmisto- ja sovelluskehityshankkeet ovatkin kärjistetysti tuomittu epäonnistumaan joko aikataulullisesti, kustannuksiltaan tai laadultaan. (Damasiotis, 2017)

3.2 Ohjelmisto projektina

Kaikki ohjelmisto- ja sovelluskehityshankkeet ovat asiakasyritykselle projekteja. Ohjelmistoprojekteissa esiintyy niille luontaisia erityispiirteitä, ja projektit ovatkin luonteeltaan uniikkeja. Niitä harvemmin tehdään kahta samanlaista tai tuotantolinjamaisesti useita peräkkäin. Luovassa työssä voidaankin samankaltainen toiminto luoda usealla eri tavalla, koska mitään konkreettista reunaehtoa ei välttämättä ole rajoitteena. Ohjelmisto tai sovellus ei konkretisoidu ja valmistu samalla tavalla kuin perinteinen tekninen tai mekaaninen järjestelmä.

Perinteisessä kurinalaisessa projektihallintomallissa useasti esiintyy selkeitä ongelmia tai haasteita. Ne eivät monestikaan ole linjassa asiakasyrityksen ja toimittajan toteutustapojen kanssa kun, puhutaan ohjelmisto- ja sovelluskehityshankkeista.

Haasteita:

- Projektissa lukitaan aika ja kustannus
- Jotta voidaan kontrolloida aikaa ja kiinteää projektikustannusta, on luovuttava osasta ominaisuuksista
- Asiakasyrityksen pitäisi pystyä esittämään tavoitteet ja ominaisuudet tarkalleen näkemättä niiden toimivuutta kokonaiosratkaisuissa

- Asiakasyrityksen perinteisten tietojärjestelmien pitäisi tukea mahdollisuutta kehittää niiden rinnalle ohjelmistoja tai sovelluksia

Yritettäessä mukauttaa ohjelmisto- ja sovellusprojektia perinteiseen projektihallintaan, johtaa se mahdollisesti käytettävyyden ja laatuarvojen heikkenemiseen.
Aikataululliset ja kustannukselliset paineet ovat niin vahvoja perinteisessä kiinteähintaisissa projekteissa, että määrittelyssä ja perussuunnittelussa useasti tehdään liiaksi kompromisseja ja oikaisuja. Todennäköistä on, ettei se vastaa käyttäjien tarpeita loppujen lopuksi lainkaan. (Cobb, 2014) (Wysocki, 2013)

Projektipäälliköt ovatkin puristuksissa ja tarvitsevat vaihtoehtoisia työskentelymalleja tai substanssinomaista kompetenssin lisäämistä. Merkittävät epäonnistumiset ohjelmisto- ja sovellusprojekteissa voivatkin johtua kyseenomaisesta vääränlaisesta projektinhallinta linjauksesta. Ongelmaksi nousee perinteisen projektihallinnan teema, jossa projektinhallinta on vasara ja projektit ja haasteet nauloja. Tallainen yksiulotteinen ratkaisumalli ei tutkitusti toimi moderneissa ohjelmisto- ja sovelluskehityshankkeissa. (Wysocki, 2013)
Projektien luonteen ja niiden tunnusomaisten piirteiden pitäisikin ohjata, minkälaisella tai minkä typpisellä projektinhallinnan viitekehyksellä sitä pitäisi lähestyä. Asiakasyrityksen olisikin hyvä tarkastella, onko organisaatiolla kykyä ja mahdollisuuksia mukautua uudenlaisiin toimintamalleihin projektihallinnassa.

Eräänlainen analogia uudenlaisesta projektinhallinnasta on perinteinen kokki vastaan ketterä keittiömestari (Wysocki, 2013)

- Hyvällä kokilla on kyky luoda erinomaisia ruoka-annoksia, mutta annokset ovat hyvin perinteisiä ja ne noudattelevat pääasiassa ennalta määriteltyjä raaka-aineita ja reseptejä.

- Hyvällä keittiömestarilla onkin kyky luoda erinomaisia ruoka-annoksia laajemmasta ja eksoottisemmasta raaka-aina valikoimasta. Keittiömestarilla on tarvittavat taidot ja kokemus luoda ja innovoida aivan uudenlaisia ruoka-annoksia, jotka sopisivat paremmin tapahtumahetkeen.

Puretaan haastetta perehtymällä ensin tarkemmin perinteiseen projektimalliin.

3.3 Perinteinen Projektikolmio

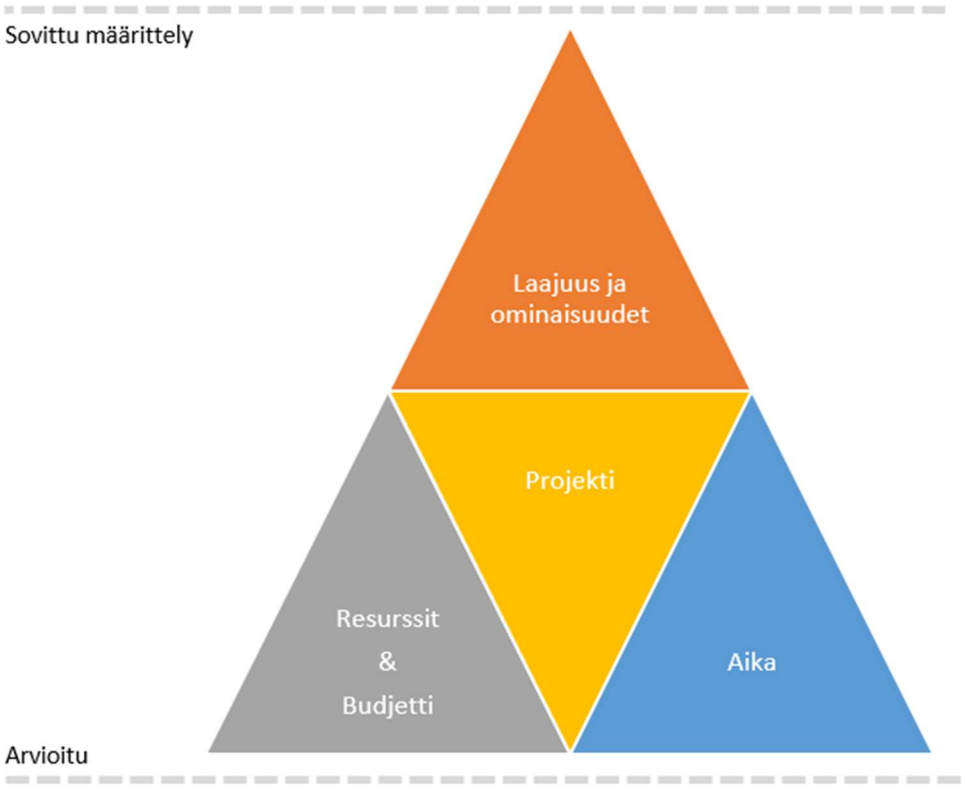
Projektikolmio (Eng. The Iron Triangle) on tohtori David Barnesin vuonna 1969 ehdottama projektimalli tuotekehityshankkeisiin. Malli noudattelee perinteistä vesiputousmallia ja onkin laajalti käytössä perinteisissä projektitoimitusyrityksissä. Vesiputousmallista on tarkempi kuvaus tämän kappaleen jälkeen.

Projektikolmiomallin tarkoitus on esittää projektitiimille tarvittavat projektinhallinnan peruselementit. Peruselementeillä on tunnistettu riippuvuussuhde keskenään ja siten voidaan esittää sellaiset muuttujat helpommin, jotka suoranaisesti vaikuttavat toisiinsa projektikokonaisuudessa. Kuvassa 1 on esitetty projektikolmion perusmalli.

Laajuus ja ominaisuudet
Sellaiset välttämättömät toiminnot ja määrittelyt, jotka tuottavat toimivan tuotteen tai järjestelmän.

Resurssit ja budjetti
Sisältää pääsääntöisesti kustannusarvion ja tarvittavat henkilöresurssit. Lisäksi resursseja voivat olla esimerkiksi, projektikohtaiset työkalut tai muut valmistusmenetelmät.

Aika
Projektin aikataulutus ja siihen liittyvät toiminnot esimerkiksi, työvaiheistukset ja ajalliset resurssoinnit alihankinnasta.



Kuva 1, *Projektikolmiomalli*

Ohjelmistoprojekteissa hyvin useasti koitetaan lukita kaikki kolme elementtiä. Lukitus voikin vaikuttaa selkeältä linjaukselta, koska perinteisessä mallissa ”tiedetään mitä ostetaan, millä hinnalla, ja milloin se toimitetaan”. Esimerkiksi, jos laajuus ja ominaisuudet on tilattu toimittajalta kiinteällä hinnalla, kesken projektin tehtävät ominaisuusmuutokset kohdistuvat joko resursseihin ja budjettiin tai aikaan. Muutoksen takia tuote tai järjestelmä maksaa enemmän kuin oli arvioitu tai sen käyttöönotto ja valmistelu viivästyy aikataulullisesti.

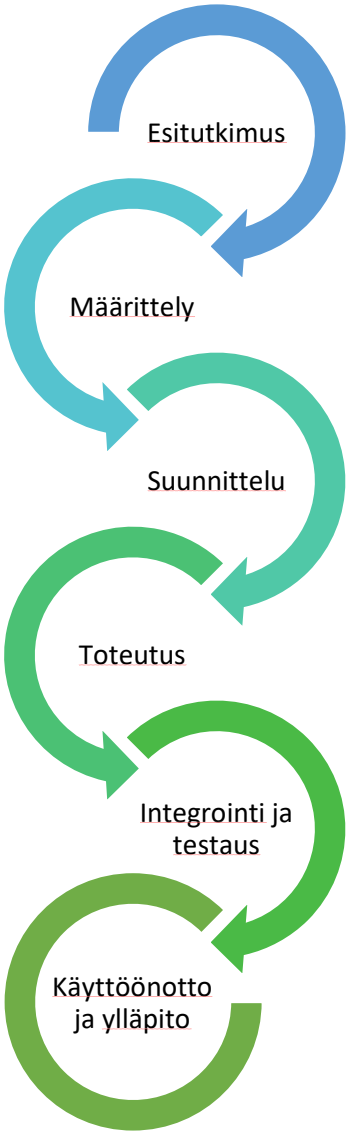
Perinteinen projektihallinta onkin johtanut tarpeeseen luoda erityyppisiä projektinhallintamenetelmiä. Varsinkin ohjelmisto- ja sovelluskehitys alalla, joka on varsin nuori verrattuna perinteisiin tuotannollisiin aloihin. (Cobb, 2014)

3.4 Perinteinen vesiputousmalli ohjelmistokehityksessä

Liikkeelle lähtiessään ohjelmistonkehitys tai hankintamalli noudattelee monesti perinteistä vesiputousmallia. Vesiputousmallista on tarjolla paljon erityyppisiä muunnelmia

ja variaatioita. Pääsääntöisesti siinä esiintyvät kuvan 2 tyyppiset vaiheet, esitutkimus, määrittely, suunnittelu, toteutus, integrointi/testaus ja käyttöönotto ja ylläpito. (Haikala&Merijärvi, 2004)

Vesiputousmallin roolit ovat hyvin vaihekohtaisia, projektipäällikköä lukuun ottamatta kussakin vaiheessa voi toimia useampia henkilöitä. Yksi henkilö voi myös toimia monessa toiminnossa ja vaiheessa, tämä onkin yleistä pienissä organisaatioissa. (Cobb, 2014)



Kuva 2, *Vesiputousmalli* (Haikala&Merijärvi, 2004)

Ensin on tehtävä tavalla tai toisella esitutkimusvaihe ja siihen liittyvät korkeantason vaatimusmäärittelyt ja tavoitteet. Kyseisessä vaiheessa selvitetään löytyisikö tarpeeksi

suuri ongelma tai hyvä idea, jota kannattaisi lähteä ratkaisemaan kehityshankkeessa. Samalla tutkitaan, onko siihen valmista markkinoilla olevaa ratkaisua olemassa.

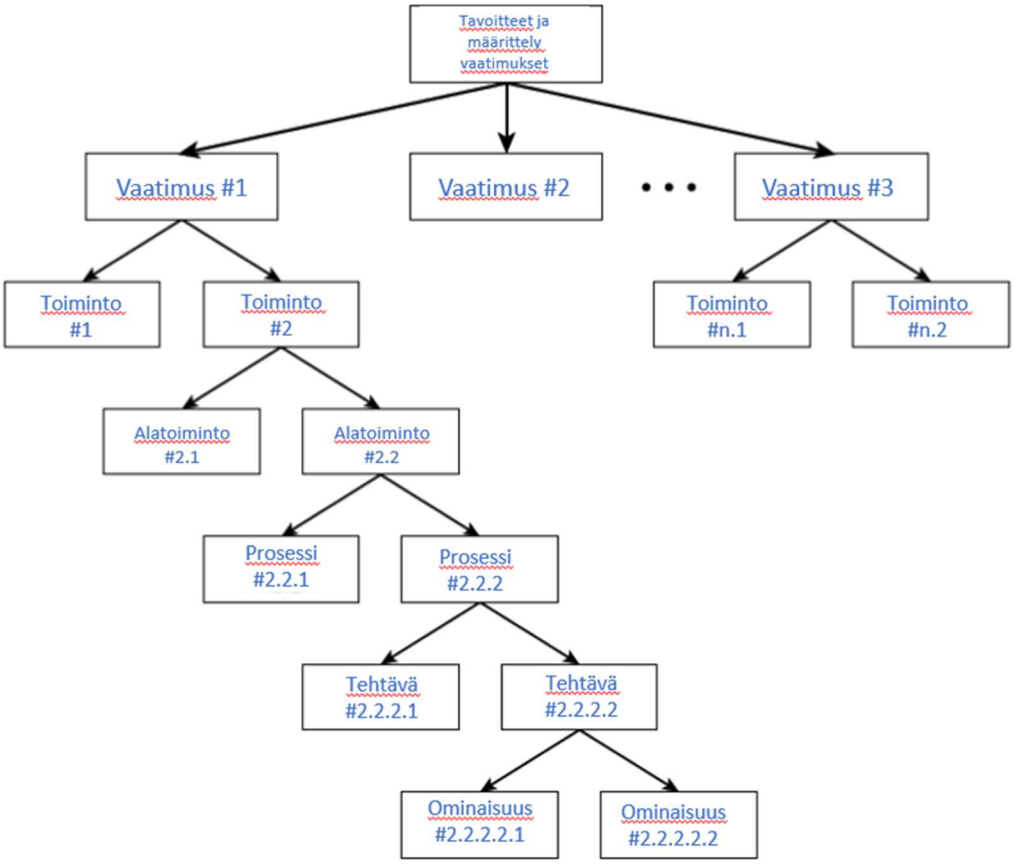
Vaatimusmäärittelyvaiheessa on erittäin haastava ilmiö, joka esiintyy määrittelyn ja tavoitteiden ongelmallisuutena. Ensinnäkin pitäisi ymmärtää, miksi ohjelmistoa tai sovellusta tarvitaan, ja mitä sisältöä käyttäjät siitä tarvitsevat ja tuottavat.
Määrittelyn vajavaisuus tai sen puutteellisuus projektin alkaessa tai kehitysvaiheessa johtaa vääjäämättä huonoon lopputulokseen. Päästäkseen tarpeeksi laadukkaaseen vaatimusmäärittelyyn perinteisessä vesiputousmallissa vaadittaisiin ajallisesti pidempää projektivaihetta ja enemmän resursseja, joita harvemmin on saatavilla.

Heikkolaatu tai arvo tulee esiin vahvemmin, jos kyseessä on isompi projekti tai laajempi järjestelmän toteutus. Tutkittuun tietoon perustuvien käytäntöjen ja ratkaisuiden löytäminen pidentäisi vielä lisää projektivaiheen ajankäyttöä, ja hankkeen toteuttamisvaiheen aloittamista. Tällöin helposti lipsutaan oikaisemaan ja tekemään kompromisseja omienkäsitysten perusteella vaatimusmäärittelyssä. (Wysocki, 2013)

Vaatimusmäärittely dokumentaatio toimiikin karttana ohjelmistokehityksen toteutukselle ja vaatii siten erityistä tarkkuutta. Valmiin tuotteen tulisikin sisältää määrittelyssä osoitetut ominaisuudet ja toiminnot tarpeeksi laadukkaalla tavalla, jolloin se vastaisi kysymykseen, miksi sitä tarvitaan ja mitä arvoa se tuottaisi.

Vaatimusmäärittelyssä suositellaan karsittavan tavoitteita niin, että ne voitaisiin holistisesti esittää koko projektitiimille. Ylätason tavoitteet tulisi purkaa pienempiin moduuleihin, jolloin niiden tuomat haasteet ja ongelmat olisivat helpommin tulkittavissa kokonaisuutta tarkasteltaessa. Kokonaisuus siis ratkaisee. Ei niinkään yksittäiset tavoitteet tai ominaisuudet, joihin monesti takerrutaan.

Kuvassa 3 on esitetty vaatimusmäärittelyn toiminnot erittelykuvana (Wysocki, 2013)



Kuva 3, *Vaatimusmäärittelyn erittelykuva* (Wysocki, 2013)

Hyvänä käytäntönä onkin lähteä määrittelyssä siitä missä mahdollisesti ohjelman tai sovelluksen isoimmat arvot ja riskit sijaitsevat. Siten arvojen ja riskien poistamien määrittelystä helpottuu. Kuvassa 4, vaatimusmäärittelyn leikkuri.

Kuva 4, *Vaatimusmäärittelyn leikkuri*

Täydellistä määrittelyä ei voida saavuttaa tai se voi käytännössä olla mahdotonta. Kuten yllä todettiin asiakasyrityksen organisaatiossa tai olemassa olevissa järjestelmissä voi olla niin paljon muuttujia, että niiden tarkkamäärittely voisi viedä mahdollisesti vuosia. (Wysocki, 2013)

Teknologian muutosvauhti hankaloittaa osaltaan vaatimuspäätöksien tekoa projektitiimissä tai hallinnollisessa osassa, jossa ohjelmistokehityshankkeen pitäisi noudatella muita liiketoimintayksikön strategisia linjauksia.

Suunnitteluvaiheessa linjataan ohjelmiston tai sovelluksen tekninen arkkitehtuurisuunnitelma määrittelydokumentaation ja tavoitteiden perusteella. Suunnitteluaiheessa myös analysoidaan vaatimusmäärittelyn tekniset reunaehdot ja tehdään valittujen vaatimusten tarkemmat kuvaukset.

Toteutusvaihe on suoraviivainen prosessi, jossa tuotetaan tarvittava ohjelmistomateriaali suunnitteludokumentaation perusteella. Kun vaatimukset ja ominaisuudet osoitetaan valmiiksi ja toimiviksi, voidaan siirtyä integrointi ja testausvaiheeseen.

Testausvaiheessa iteroidaan ohjelmistoa ja sovellusta siten että siitä löytyisivät mahdolliset virheet tai merkittävät puutteet toiminnallisuudessa.

Käyttöönotto- ja ylläpitovaiheessa ohjelmisto tai sovellus normaalisti siirtyy pois projektiorganisaatiolta normaalin ylläpidon piiriin. Kyseessä olevassa vaiheessa ei varsinaisesti kehitetä järjestelmää enempää, virheidenkorjaus ja asiakkaan ongelmien ratkaisut ovatkin keskeisempiä teemoja.

3.4.1 Vesiputousmallin yhteenveto

Vesiputousmalli onkin perinteisesti ollut ohjelmisto- ja kehityshankkeissa ensimmäisenä menetelmänä ja se voi vaikuttaa hyvältä kokonaisratkaisulta. (Cobb, 2014)
Käytännössä kiinteähintaisen ohjelmistoprojektin ostaminen on erittäin haastavaa, ja ei edes monesti suositeltavaa sen riippuvaisuudesta ohjelmiston tai sovelluksen määrittelyyn ja suunnitteluvaiheen oikeellisuuteen projektin alussa.

Kiinteähintainen malli myös ohjaa toimittajaa minimoimaan työaikaa, joka voi esiintyä rajoittamalla vaatimuksiin tai toiminnallisuksiin tehtäviä muutoksia. Aluksi tehdyt omien käsitysten perusteella muodostetut vaatimus- tai määrittelyominaisuudet voivatkin jäädä vajavaiseksi, Tästä johtuen projektille tarvittaisiin seuraava kehityssykli, joka nostaisi projektin kokonaiskustannuksia.

Vesiputousmallissa suunnitelmien muuttaminen jo aloitettuihin tai tehtyihin ratkaisuihin on hidasta ja aikaa vievää. Teknisestä laadusta yleisesti tingitään kiinteähintaisissa ohjelmisto- ja sovelluskehityshankkeissa useimmin kuin ketterän mallin hankkeissa. (Kerzner, 2010)

3.4.2 Esimerkkiohjelmisto *”Site Audit Tool”* - Projektin alku

Asiakasyrityksen ohjelmisto- ja sovelluskehitysprojekti ”*Site Audit Tool*”, käynnistyikin perinteistä vesiputousmallia noudattaen. Projektissa tehtiinkin hyvin kattava ja seikkaperäinen esitutkimus. Esitutkimuksessa kartoitettiin, löytyisikö markkinoilta

sopivaa tuotetta, ja olisiko toimittajat rakentaneet vastaavanlaisia tuotteita muille asiakkaille.

Projekti noudatteli hyvinkin tarkkaan asiakasyrityksen tapaa tehdä projekteja siihen tarkoitetulla projektihallintatyökalulla ja käytännöillä. Vaatimusmäärittely ja käyttötarkoitus ohjasi tuotteen arkkitehtuuria, mutta aikataulutus ja kiinteähintaisen projektin ostaminen toimittajalta ohjasi projektinhallintaa. Hankintakustannus ja budjetti eivät vastanneet vaatimusmäärittelyn tavoitteita, ja niistä jouduttiin leikkaamaan ydintoimintoja pois. Projektille saatiin lopuksi luotettava toimittaja useamman haastattelukierroksen jälkeen, ja projektin toteutusvaihe päästiin aloittamaan.

3.5 Ketterä malli ohjelmistokehityksessä

Agile - ketterä malli ei ole sitä että ’asioita *tehdään* ketterästi’, se on mielentila tai ajatusmalli ’jossa *ollaan* ketteriä’. Jotta ketteryys olisi onnistunutta tarvitaan siihen oikeanlainen asenne ja mielentila, mikään työkalu tai käytäntö ei itsessään tuo tarvittavaa ketteryyttä. Ketterä malli onkin ennemmin matka, kuin määränpää. (Measey, 2015)

Ketterä malli tuokin useasti mahdollisuuden reagoida muutoksiin tehokkaammin tai nopeammin. Menetelmällä saavutaan nopeampia tuloksia, koska ohjelmistokehittäjä voi keskittyä tuottamaan toiminnallisuuksia pala kerrallaan isompien kokonaisuuksien sijaan.

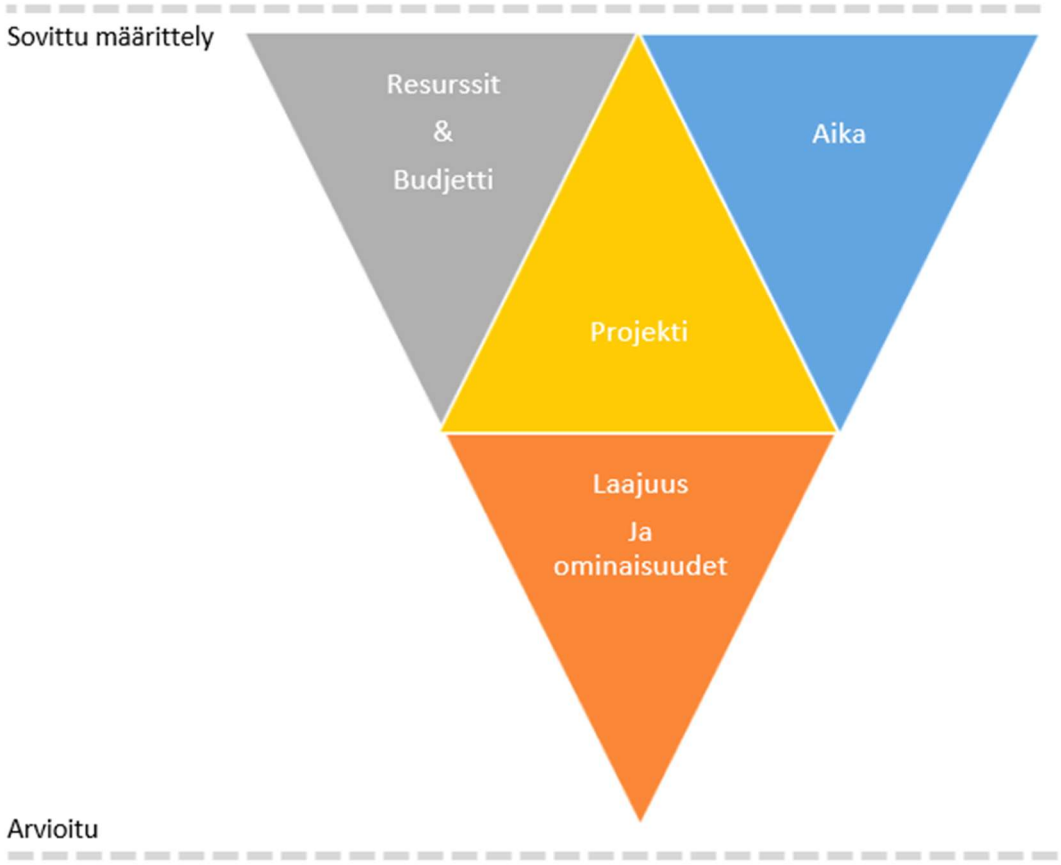
Käytännössä ohjelmisto- tai sovelluspalasia iteroidaan jatkuvasti projektin aikana. Menetelmällä saadaan toimittajalle tai kehittäjälle nopeasti palaute ohjelmiston toimivuudesta, jolloin voidaan reagoida toiminnon tai ominaisuuden muutokseen kehitystyön aikaisemmassa vaiheessa.

Tilaajan tai asiakasyrityksen pelkoa ”epämääräisestä” projektikustannuksesta lievittääkin ketteryyden tuoma kustannustenhallintamalli. Kun hankkeen aikana ymmärrys ja osaaminen kasvaa myös asiakasyrityksellä, kykenee asiakasyrityksen projektitiimi puuttumaan ominaisuuksien tärkeysjärjestykseen ja lisäämiseen ripeästi testaamisen

avulla. Toimittajan projektitiimin ajankäyttöä pystytään ohjaamaan tarkemmin tämän kaltaisella ennakoivalla ja palautemaisella lähestymisellä. Kustannustenhallinta säilyy näin omissa käsissä, eivätkä tehtävät jää suorittamatta tai karkaa tulevaisuuteen. (Cobb, 2014) Kiinteähintaista ketteryyttä ei siis oikeastaan ole, koska kustannus perustuu toteutuneeseen työhön eli kehitysjaksoihin. Asiakasyrityksen vastuulle jääkin sitoutua projektiin ja ymmärtää lyhyiden kehitysjaksojen tuottaman testauspaineen tarkoitus.

Yksinkertaistettuna ketteryys on ajatusmalli ja mielentila, jossa määrittely ja projektinlaajuus tehdään korkealla ylätasolla. Toteutus on inkrementaalista ja vaiheittaista, ja se perustuu nopeaan palautteeseen. Tutkitusti tämänkaltainen joustava lähestyminen ohjelmisto- ja sovelluskehitysprojekteissa luo arvollisesti ja laadullisesti parempia lopputuloksia koska kokonaisuuksien hallinta on systemaattisempaa kuin perinteisissä suunnittelukeskeisessäprojekteissa. (Cobb, 2014).

Kuvassa 5 on esitetty paradigman muutos perinteiseen projektikolmioon kuva 1, muutoksessa arvioidaan ominaisuudet ja laajuus samalla lukittaisiin kehitysjakson pituus ja hinta.



Kuva 5, *ketterän mallin muutos projektikolmiossa*

3.5.1 Esimerkki ohjelmisto *"Site Audit Tool"* - Kehitysvaihe

"*Site Audit Tool*" kehitysvaiheessa päädyttiin eräänlaiseen projektihallinnan hybridimalliin, jossa asiakasyritys perinteisesti toimi kolmen projektipyramidin säännön sisällä. Koska kiinteähinta ja tavoitteet olivat sopimusehdoissa, ainoaksi vaihtoehdoksi oli optimoida käytettävää aikaa. Tällä tarkoitettiin kehitysjaksojen pituuden ja ajan käytön lyhentämistä toimittajalla.

Toimittajan ja asiakasyrityksen välillä sovellettiin ketterän menetelmän perusteita, luovuttiin siis perinteisestä tavasta mallintaa kaikki projektinvaiheet asiakasyrityksen projektihallintatyökaluun. Ominaisuudet ja toiminnot lajiteltiin niiden tuottaman arvon mukaan, ja kehitysjaksoissa rakennettiin tärkeimmät toiminnot ensin. Kehitysjakson pituus oli noin 2-3 viikkoa, jonka jälkeen katselmoitiin saavutetut tavoitteet. Katselmointipalaverin jälkeen asiakasyritys testasi sovellusta loppukäyttäjän toimesta ja

antoi palautteen. Palautteen jälkeen toimittajan kanssa sovittiin yhdessä seuraavan kehitysjakson ominaisuudet tärkeysjärjestyksessä, ja aloitettiin prosessi uudelleen.

3.6 Tekninen velka

Tekninen velka on termi, jota esiintyy nykypäiväisissä keskusteluissa ohjelmistojen, sovellusten tai järjestelmienkehittämisessä. Termi mainittiin ensimmäisenkerran Ward Cunninhamin toimesta vuonna 1992.

Yksinkertaistetusti se tarkoittaa oikaisuja ohjelmisto- tai sovellusarkkitehtuurissa ja suunnitteluvaiheessa. Oikaisut voivatkin aiheuttaa kustannusten nousua ja ennakoinnin heikkenemistä. Tahallisesti tai tahottomasti tehtyjen päätösten vaikutukset heijastuvat useasti, jatkokehityksen tai ylläpidollisten teemojen ympärille ja konkretisoituvat vasta tulevaisuudessa. (McConnell, 2007)

Tekninen velka ilmiönä

Teknisen velan taustalla voikin olla ilmiöitä projektihallinnasta. Esimerkiksi aikataulupaine, rahoitus tai resurssoinnin vajaus johtaa useasti huonoihin käytäntöihin ja teknisen velan kasvattamiseen. Pitkittynyt viimeistely ohjelmisto- tai sovelluskehityshankkeessa voi olla oire korkeasta teknisestä velasta, joka on luotu joko projektin aikana tai projektin alussa tehdyistä suunnittelu ja määrittelyratkaisuista.

Tekninen velka onkin hyvin tuore tutkimustieteen ala ja siitä ei ole kovin paljoa tutkimustietoa, lähimmät kirjalliset tutkielmat ovat vuodelta 2010.
Vuonna 2014 joukko akateemisia tutkijoita valmistelivat aineiston, jossa pyrittiin osoittamaan konkreettisemmin minkä tyypistä teknistä velkaa esiintyy ja niiden pohjalta rakennettiin mittareita, joilla voidaan havainnollistaa syyperäisiä vaikutuksia. (Alves, 2014)

Tutkijat päätyivät tuloksissaan kolmeentoista teknisen velan tyyppiin, ja niiden syyperäiseen määrittelyyn:

- Arkkitehtuurillinen
- Koontiversio (Eng. Build)

- Koodipohjainen
- Virheellisyys rakenteessa
- Suunnittelu
- Dokumentointi
- Infrastruktuuri ja olemassa olevat järjestelmät
- Ihmiset
- Prosessit
- Määrittely ja tavoitteet
- Palvelut
- Testaus automaatio
- Testaus ja sen määrät

Käytännössä teknistä velkaa ei voida välttää niin, etteikö sitä jossain määrin syntyisi aina uusissa ohjelmisto- ja sovelluskehityshankkeissa.

Kyseisiä teknisen velantyyppejä hyödynnetäänkin nykyisissä ketterän projektin menetelmissä. Huomioimalla ne osaksi tuotekehitys strategiaa, saadaan kehityshankkeita nopeammin markkinoille. Samalla pyritään siirtämään velan takaisinmaksua mahdollisimman joustavaksi oikeilla syyperäisillä ratkaisuilla. (McCormick, 2017)
Kaikki tekninen velka ei siis ole pahasta, se onkin yksi työkalu ohjelmisto- ja sovelluskehitysprojekteissa nykypäivänä.

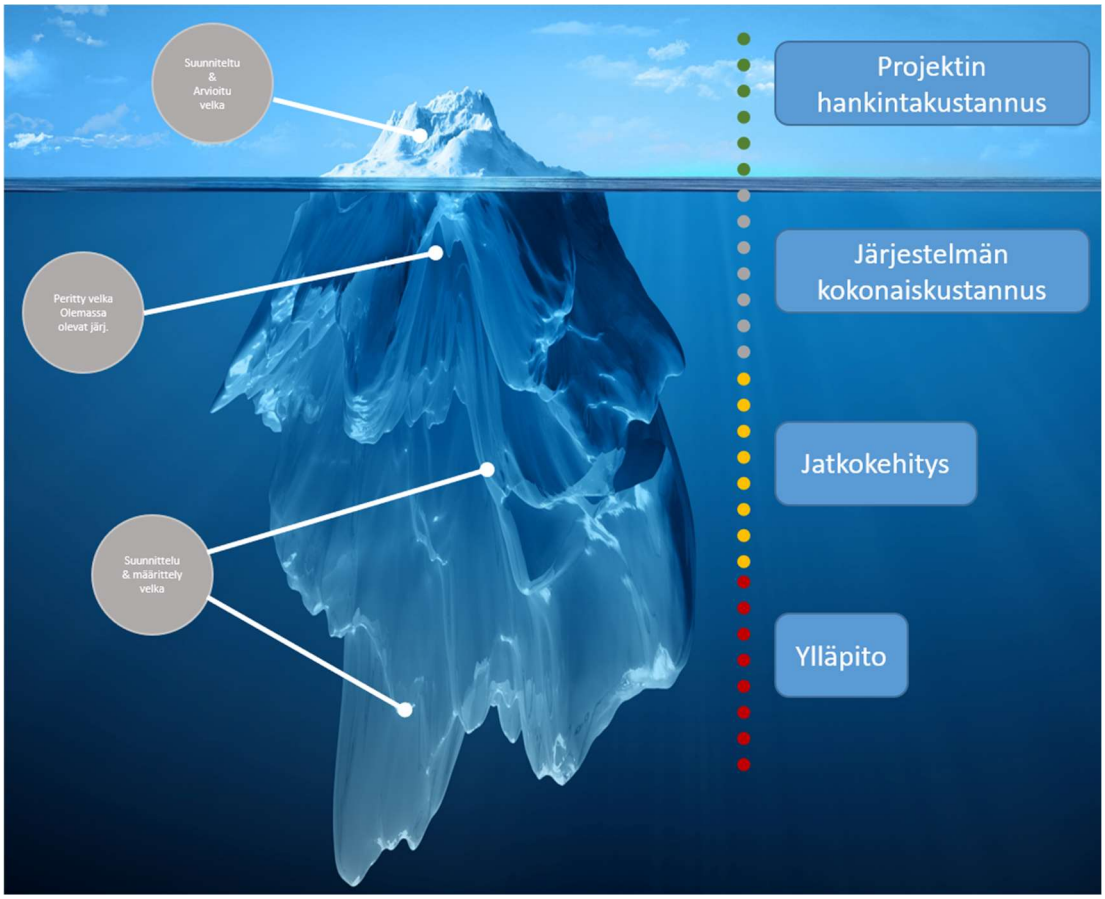
Teknistä velkaa voi syntyä myös kehitysprojektien ulkopuolella, joka ei suoranaisesti liity käynnissä olevaan tai osaksi valmiiseen ohjelmisto- ja sovelluskehityshankkeeseen. Järjestelmien siiloutuminen yksi tämänkaltainen esimerkki, jossa asiakasyrityksen tuotannollista tietojärjestelmää ei kyetä uusimaan tarpeeksi nopeasti tai linjaamaan oikein. Kyseinen tietojärjestelmä ei siis vastaisi uusien sovelluksien tarpeisiin, jolloin voidaan osoittaa sen syyperäinen velkatyyppi.

Tekninen velka on kuitenkin jossain määrin aina maksettava takaisin ohjelmiston tai sovelluksen elinkaaressa. Sen arvioiminen ja kontekstiin asettaminen onkin tärkeää projektin alussa ja sen aikana, koska muutoin se voi käydä liian kalliiksi tulevaisuudessa. Kuten rahalainassa se ei ole ongelma, ennekuin se on.

Tekninen velka päätöksenteossa

Asiakasyrityksen tyyppisessä liiketoimintayksikössä, jossa ohjelmisto- ja sovelluskehityksellä haetaan jotain toiminnan muutosta tai ratkaisua, pitäisi hankkeiden päätöksisissä pyrkiä tunnistamaan ja ottamaan huomioon teknisen velan ilmiö. Projektille osoitetut hankintakustannukset jopa yksinkertaisille sovelluksille, luovat aina jonkin määrän teknistä velkaa.

Kuvassa 5 on esitetty yksinkertainen jäävuorimalli. Näkyvä hankintakustannusosa ei tule kattamaan projektin elinkaarikustannuksia tulevaisuudessa.



Kuva 6, *Teknisen velan jäävuorimalli.*

3.6.1 Esimerkkiohjelmisto *"Site Audit Tool"*- Tekninen velka

"*Site Audit Toolin*" suurin heikkous oli lähtötiedon saaminen olemassa olevista tietojärjestelmistä. Tämä aiheuttaa teknisen velan näkökannalta tilanteen, jossa jatkokehitys on erittäin haastavaa sen korkean kustannuksen vuoksi. Teknisen velan poistamiseksi tai oleellisesti vähentämiseksi, pitäisikin olemassa olevan tietojärjestelmän uusiutua niin että se palvelisi moderneja sovellusratkaisuja. Silloin järjestelmien yhteen saattaminen loisi uudenlaista arvoa organisaatiolle ja jatkokehittämisen hintapaineet sovellukselle alenisivat. Toinen merkittävä teknisen velan ilmentymä projektissa oli budjetti ja kiinteähintainen kokonaistoimitus toteutus. Vaikutukset heijastuivat liikaa ominaisuuksien ja toimintojen karsimiseen, jotta projekti saatiin vietyä auttavasti loppuun asetetuissa reunaehdoissa. Lopputulos ei siten vastannut kaikkia päätöksentekijöiden asettamia tavoitteita.

3.7 Ohjelmistoprojektin epäonnistuminen - onnistuminen

Vuonna 1986 Alfred Spector kirjoitti artikkelin, jossa verrataan sillan rakentamista ohjelmisto- ja sovelluskehittämiseen. Perustana artikkelissa esitettiin, että yleensä sillat rakennetaan ajallaan, niissä pysytään budjetissa ja harvemmin ne putoavat tai romahtavat. Ohjelmisto- ja sovelluskehityshankkeet ovat aina myöhässä, yli budjetin ja yleensä rikkoutuvat tai ovat laadullisesti huonoja (huom. Siltaprojektit olivat kyllä monesti myöhässä, yli budjetin ja niitä on pudonnut, romahtanut ajansaatossa)

Perustana olleeseen väitteeseen siltojen ylivoimaisesta onnistumisesta onkin rationaali, jossa siltojen suunnittelussa käytetään ennennäkemätöntä tarkkuuttaa. Suunnitelmat jäädytetään ja toimittajalla tai urakoitsijalla on erittäin vähän liikkumavaraa suunnitteluyksityiskohtien tai laskemien muutoksissa. Nopeasti muuttuvassa liiketoimintaympäristöissä ja kilpailusta markkinoilla, joustavuus on tuonut uudenlaisen haasteen ohjelmistokehitys- ja suunnittelun toimintamalleihin.

Erona siltojen rakentamiseen ja ohjelmisto- ja sovelluskehittämiseen onkin, että siltoja on rakennettu viimeiset 3000 vuotta. Sillan putoaminen tai romahtaminen tutkitaan usein hyvin huolellisesti ja siitä muodostetaan yksityiskohtainen raportti. Raportissa pyritään selvittämään kaikki mahdolliset tapahtumat ja asiat, jotka johtivat sillan

putoamiseen tai romahtamiseen. Ohjelmisto- ja sovelluskehityshankkeissa ei ole tämänkaltaista lähestymistä ole ollut, ja sen vuoksi useasti voidaankin toistaa samankaltaisia virheitä ohjelmisto- ja sovelluskehitysprojekteissa (The Standish Group, 2014)

Määrittely mittarit

Ohjelmisto- ja sovelluskehityshankkeiden epäonnistumisissa onkin haasteellista nähdä selkeitä mitattavia ilmiöitä projektien luonteen vuoksi. Projektien mitattavat ilmiöt vaihtelevat merkittävästi esimerkiksi vertailemalla julkis- ja yksityissektorin hankkeita. Lisäksi erityyppiset teollisuusalat tai kaupalliset toimialueet luovat hyvinkin pirstaloituneen projektikentän, puhuttamattakaan toimittajien ja asiakkaiden prosesseista ja työvälineistä.

Mittareina käytetäänkin monesti syyperäisiä selvityksiä, jotka perustuvat raportoituihin julkisiin projekteihin, ja haastatteluiden kautta saatuun tutkimustietoon.
Projektikolmion peruselementtejä voidaan joissain tapauksissa käyttää reunaehtoina, miten mitata ohjelmisto- tai sovelluskehityshankkeita. (McManus, 2004)

Tulos1:
Projektin luetaan onnistuneeksi, kun se on valmistunut ajallaan, budjetissa ja se sisältää sille määritellyt toiminnot ja ominaisuudet

Tulos2:
Projekti luetaan vaillinaiseksi, jos hankkeessa ylitetään budjetti merkittävästi tai aikataulullisesti. Vaillinaisuus voi esiintyä myös, jos tuotteen sisältämät ominaisuudet ja toiminnallisuudet eivät vastaa asetettuja tavoitteita.

Tulos3:
Projekti luetaan epäonnistuneeksi, kun se keskeytetään tai peruutetaan jossain vaiheessa sen elinkaarta.

Onnistumiseen johtaneita pääkohtia

Tutkimustiedon analysoinnista ja haastatteluiden tuloksena tunnistettiin kolme merkittävää ilmiötä tai vaikutusta, jotka esiintyvät onnistuneissa Tulos1-tyyppissä

projekteissa. Näiden syyperäisten ilmiöiden puuttuminen projekteissa merkitsi myös huomattavaa muutosta epäonnistumisen riskissä. (The Standish Group, 2014)

Kolme merkittävää ilmiötä ja vaikutusta, onnistuneisiin projekteihin:

1. Käyttäjän / käyttäjien osallistuminen
2. Johdon tuki
3. Selkeästi määritellyt tavoitteet ja ominaisuudet

Muita onnistumiseen kartoitettuja pääkohtia ja vaikutteita:

- Selkeä ja kunnollinen suunnittelu
- Realistiset odotukset
- Pienemmät projektivaiheet
- Osaava henkilöstö ja projektinjohto
- Tuotteen omistajuus
- Selkeä visio tuotteesta
- Sitoutunut ja keskittynyt henkilöstö

Epäonnistumiseen johtaneita pääkohtia

Vaillinaisissa projekteissa, jotka ovat Tulos2-tyyppisiä. Ilmenee seuraavanlaisia vaikutteita:

- Puutteellinen käyttäjän / käyttäjien palaute
- Puutteellinen tai vajavainen tavoitteiden määrittely
- Tavoitteiden ja toimintojen merkittävä muutos projektin aikana
- Ei johdon tukea, tai puutteellinen osallistuminen
- Tekninen epäpätevyys
- Riittämättömät resurssit
- Epätodelliset odotukset
- Epäselvät ominaisuudet
- Aikataululliset odotukset
- Uudet teknologiat

Keskeytetyissä ja peruutetuissa projekteissa, jotka ovat Tulos3-tyyppisiä, esiin nousivat seuraavat vaikutteet:

- Keskeneräiset ominaismäärittelyt tai puutteelliset ominaisuudet

- Puutteellinen käyttäjän / käyttäjien osallistuminen
- Riittämättömät resurssit
- Epärealistiset odotukset
- Ei johdon tukea, tai puutteellinen osallistuminen
- Suunnittelun epäonnistuminen
- Ohjelmistoa tai sovellusta ei tarvittu
- Puutteellinen IT:n hallinta, ylläpidolliset seikat
- Teknologian lukutaidottomuus, tai vääränlaiset teknologiset valinnat

3.7.1 Esimerkkiohjelmisto *"Site Audit Tool"*- Tulos tai ulos

"*Site Audit Tool*" oli mielenkiintoinen projekti asiakasyrityksessä, jossa kokeiltiin muutakin kuin Excelin kahluuallasta. Vaikkakin sovellus on melko yksinkertainen ja se on helppo visioida, sen ohjelmisto- ja sovelluskehitysprojektin luonne on hyvin erityyppinen kuin asiakasyrityksessä on totuttu tuottamaan. Projektin lonkerot ulottuivat sellaisiin teemoihin, joihin asiakasyrityksessä ei ollut totuttu törmäämään.

Syyperäisiä vaikutteita tarkastelemalla voidaankin tulkita, että ohjelmisto on Tulos2-tyyppisesti vaillinainen. Jotta sovelluksen maturiteettia saataisiin kasvatettua tarvittavalle tasolle, tulisi perinteisiä olemassa olevia tietojärjestelmiä uusia. Siten saataisiin myös uuden mobiilisovelluksen arvoa nostettuja ja sen käyttöä vietyä nopeammin eteenpäin organisaatiossa.

3.8 Yhteenveto

Mitään ihmelääkettä ei siis ohjelmisto- ja sovelluskehityshankkeiden onnistumiseen ole. Se ei ole suoranaisesti riippuvainen, siitä minkälaisella projektihallinta mallilla sitä tehdään, tai minkälaisesta määrittelystä tai tavoitteista asiakasyrityksessä se koostuu. Pitäisikin mahdollisesti pyrkiä tunnistamaan ja kehittämään niitä osa-alueita, joissa syyperäisiä vaikutteita esiintyy, kuten selkeästi määritellyt tavoitteet.

Kuten kappaleessa 3.1 todetaan, haasteita riittää niin organisaatioissa kuin teknologioissa. Ratkaisuna tähän voisi olla, tuoda osaamista lähemmäksi päätöksentekijöitä ja luomalla vaihtoehtoisia ketteriä menetelmiä perinteiselle projektisuorittamiselle. Pyrittäisiin siten ymmärtämään, miten saavuttaa tarvittava tasapaino perinteisen ja ketterän projektimallin välillä organisaatioissa (Cobb, 2014)

Jotta voisi tulla ketterämmäksi, pitäisi muuttaa myös perinteisiä ajatusmalleja:

- Korosta maksimaalista arvoa, ei pelkästään kontrollia
- Korosta itseohjautuvuutta
- Rajoita korostettua dokumentointia
- Hallitse projektivaiheiden virtaa, ei pelkästään rakennetta
- Toimintojen rajat ylittävä johtajuus

Noudattamalla hyväksi koettuja käytäntöjä ja oppimalla virheistä ei ole mahdotonta siirtyä uudenlaiseen toimintakulttuuriin asiakasyrityksessä.

4 NYKYISET KETTERÄT MENETELMÄT

Ketteriin menetelmiin ei ole tarkoitus vain *siirtyä* vain sen vuoksi että niitä on olemassa. Polarisaatio perinteisen vesiputousmallin ja ketterien menetelmien välillä onkin ajan saatossa johtanut myyntipuheisiin niiden paremmuudesta keskenään, ja on syntynyt eräänlainen nollasummapeli. Ketterien menetelmien äärikannattajat epäinhimillistävät ohjelmisto- ja sovelluskehityksessä perinteisen vesiputousmallin perustelemalla sitä prosessien palvonnaksi. Vastaavasti perinteisen menetelmän äärikannattajat pitävät ketteriä menetelmiä ”lasten leikkeinä” ja ”pitämällä hauskaa” vakavassa liiketoiminta alueessa.

Ketterien menetelmien aineiston laajuuden vuoksi, tässä osiossa lähetystään menetelmiä kahden Suomessa yleisesti käytetyn projektiviitekehyksen Scrum ja Kanbanin kautta. Lähestymisen tulisi antaa tarvittava tietosisältö, jotta asioiden tai menetelmien

esiin noustessa ohjelmisto- ja sovelluskehityshankkeissa olisi asiakasyrityksellä valmiuksia toimia niiden kanssa.

Osion lopuksi on esitelty MVP (Minimun Viable Product) ratkaisua osana kehityshankkeita.

4.1 Agile-menetelmä

Lyhykäisyydessään Agile-menetelmässä ei ole muusta kyse kuin iteraatiosta ja inkrementoinnista. Käytännössä termillä, tarkoitetaan miten projekti tai työvaihe jaetaan tietyn pituisiin kehitysjaksoihin. Kehitysjaksossa tai kehityssyklissä suoritetaan ja toteutetaan aina toimiva ohjelmainkrementti, joka on toimiva osa kokonaissovellusta (Cockburn, 2006)

Kun puhutaan Agile-menetelmistä, sitä ei voida erottaa Lean-menetelmästä. Alkuperäinen Lean-mallin lähde on Toyota Production System (TPS) 1900-luvun puolestavälistä. Ohjelmisto- ja sovelluskehitysalalla Lean-menettely (Eng. Lean Software Development) perustuu seitsemään periaatteeseen, näitä perusperiaatteita on sittemmin päivitetty vastaamaan ajankohtaisuutta. Taulukossa 1 esitetään edelleen laajalti hyväksytyt vanhemmat perusperiaatteet, ja vuonna 2016 saman tutkijan päivitetyt periaatteet. (Poppendieck, 2003)

Taulukko 1, Lean-menettelyn perusperiaatteet ohjelmisto- ja sovelluskehityksessä. (Poppendieck, 2003)

Alkuperäiset, 2003	Nykyiset, 2016
Hankkiutuminen jätteistä eroon	Vähennä kitkaa
Päätöksenteon venyttäminen	Rakenna laatu edellä
Nopea toimitus	Paranna oppimista
Palautteesta oppiminen	Paranna jatkuvasti
Riittävät valtuudet tiimeillä	Keskity asiakkaisiin
Järjestelmän yhtenäisyys	Optimoi kokonaisuus
Kokonaisuuden hahmottaminen ja Optimoinen	Lisää virtausta

Lean-menetelmän keskiössä on jätteiden eli tarpeettomana työn poistaminen, sillä tarkoitetaan käytännössä prosessien arvioimista. Jos esimerkiksi projektissa esiintyy merkittävästä dokumentaatiota, kuten perinteisessä vesiputousmallissa on tapana tehdä. Lean-mallissa niistä karsitaan kaikki sellaiset, jotka eivät suoranaisesti tuota arvoa prosessille. Jos sitä ei tarvita niin, miksi sellaista pitäisi tehdä.

Päätöksenteon venyttämisellä halutaan välttää virheellisiä tai hätiköityjä päätöksiä. Nopeasti tehty päätös monesti perustuu henkilökohtaiseen kokemukseen tai arvioon, pidempään harkittupäätös taas saa tuekseen faktoja ja tutkittua tietoa. Monesti hätiköity päätös johtaa projektissa jätteisiin ja ylimääräisiin työvaiheisiin. (Poppendieck, 2003)

4.2 Agile-manifesti

Vuonna 2001 Utahin Snowbirdissä Amerikan Yhdysvalloissa, kokoontui 17 ohjelmistokehitysmallien edustajaa keskustelemaan ja tutkimaan, löytyisikö silloisista kevyt ohjelmisto menetelmistä (Eng. Light-weight methods) yhteisiä piirteitä.
Edustajia olivat muun muassa Extreme Programming (XP), Adaptive Software Development (ASD), SCRUM, Crystal menetelmät, Feature-driven Development (FDD) ja Dynamic Systems Development Methods (DSDM). Edustajien toimesta päästiin yhteisymmärrykseen yleisen tason teemoissa ja näin ollen perustettiin Agile Alliance niminen ei kaupallista hyötyä tavoitteleva järjestö.

Agile-menetelmiksi kirjattiin 12 pääperiaatetta, jotka ovat perusta *Agile manifestille.* Lisäksi kirjattiin neljä toimivan *ohjelmistoprosessin* ydinarvoa. Näitä ruvettiin kutsumaan ketteriksi menetelmiksi. (Cockburn, 2006)

” Ketterän ohjelmistokehityksen julistus

Löydämme parempia tapoja tehdä ohjelmistokehitystä, kun teemme sitä itse ja autamme muita siinä. Kokemuksemme perusteella arvostamme:

- *Yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja*

- *Toimivaa ohjelmistoa enemmän kuin kattavaa dokumentaatiota*
- *Asiakasyhteistyötä enemmän kuin sopimusneuvotteluja*
- *Vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa*

Jälkimmäisilläkin asioilla on arvoa, mutta arvostamme ensiksi mainittuja enemmän." (Agile Alliance, n.d.)

Agile manifesti löytyy kokonaisuudessaan aineiston liitteestä (LIITE 1).

4.3 Scrum menetelmä

Scrum on yksi Suomen yleisimpiä ketterän menetelmän projektiviitekehyksiä Kanbanin ohella. Aineistossa pyritään esittelemään keskeiset käsitteet ja toiminnot Scrumista ja sen viitekehyksestä. Scrum termi tulee Englanninkielen sanasta *Scrumming* jota muun muassa käytetään kuvaamaan rugbypelin tilannetta, jossa joukkue tai ryhmä pyrkii etenemään yksikkönä ja tiiviissä yhteistyössä. Nimi onkin jäänyt käyttöön kuvaamaan sekä projektiviitekehyksen ja rugbypelin toimintaa, jossa molemmat sopeutuvat erilaisiin tilanteisiin, itseohjautuvat ja ovat nopeita.

Scrum tiivistettynä

Perinteinen Scrum-menetelmä seuraa tiiviisti Lean-menetelmän perusperiaatteita ja prosessia, projektikehyksessä koitetaan välttää ”jätettä” ja pyritään hylkäämään ylimääräisiä vaihetuotteita.

Scrum tarjoaakin viitekehyksen, jonka avulla projektia ohjataan. Siinä harvemmin otetaan kantaa matalantason insinöörityöhön, aluksi perustetaankin korkean tason tavoitteet asiakkaan kanssa ja priorisoidaan tuotteen työlista eli ”Product Backlog”.

Kuten kaikissa muissakin ketterissä menetelmissä Scrumissa nähdään projekti rakentuvan vaiheisiin, jotka voivat olla eri mittaisia tai pituisia kokonaisuuksia.
Sprintillä eli jaksolla tarkoitetaan ajanjaksoa, jonka aikana valmistellaan ja tuotetaan jokin ohjelmistontoiminto tai ominaisuus. Sprintti voi kestää viikosta kuukauteen

mutta ketteryydenvuoksi ne pyritään pitämään mahdollisimman maltillisena. Kvartaali on jo monesti liian pitkä Sprintti ajanjakso. (Schwaber, 2004)

Roolit

Scrumissa on vain kolme erityyppistä roolia. Toisin kuin perinteissä vesiputousmallissa, jossa voi olla useampi rooli jokaiselle prosessille. Scrumin roolit ovatkin: Tuotteen omistaja (Product Owner), Scrum-mestari (Scrum master) ja tiimi (Team)

Tuotteen Omistaja

(Product Owner) on henkilö, joka on vastuussa tuotteen ominaisuuksista ja siitä että ne ovat kartoitettu tuotteen työlistaan (Product Backlog). Tuotteen omistajalla on siis velvollisuus ylläpitää työlistan ominaisuuksia ja toimintoja oikeassa prioriteettijärjestyksessä. Henkilö on tyypillisesti toimittajan tuotepäällikkö tai asiakasyrityksen tapauksessa tekninen projektipäällikkö.

Toimittajan näkökannalta Scrum on vaativa tässä tilanteessa, koska se vaatii yhdet kasvot tai rajapinnan asiakkaan puolelta vastaamaan työlistan ajantasaisuudesta ja oikeellisuudesta. Käytännössä projektin omistaja ja projektipäällikkö tulisi olla sama henkilö, muutoin toimivaltuuksien ja päätöksen teko voi olla hidasta ja hankalaa. Tämä ei näin ollen tue projektin luonteenomaista ketteryyttä. (Schwaber, 2004)

Scrum-mestari

Scrum-mestarin tehtävänä on huolehtia, että työtä tehdään optimaalisella tavalla. Scrum-mestari ei ole projektipäällikkö eikä prosessipoliisi. Nyrkkisääntönä Scrum-mestarin päätehtävä on antaa tiimille tarvittava työrauha ja tehostaa tiimin toimintaa poistamalla esteitä tai hidasteita projektissa.

Projektitiimin jäsenet raportoivat ongelmat ja haasteet, jotka hidastavat töiden etenemistä Scrum-mestarille. Tämä onkin monesti vaikeata sisäistää perinteisessä projektimallissa, koska projektipäälliköt ovat tottuneet kantamaan kokonaisvastuun ja jakamaan tehtäviä. Siksi itseohjautuvan tiimin käsite onkin vaikea sisäistää osana projektihallintaa. Scrum-mestarilla ei siis ole mitään auktoriteettia projektitiimin kokonaisuuden suhteen, (Schwaber, 2004)

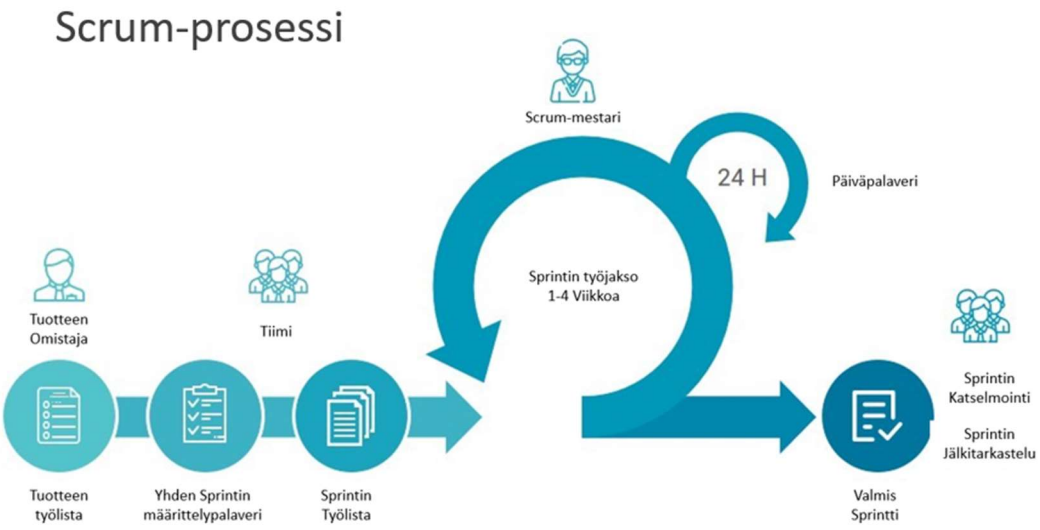
Tiimi

Vaikka Scrum-menetelmässä painotetaan yksilön vastuuta ja itseohjautuvuutta sen keskiössä on kuitenkin tiimi. Tiimiin ei erikseen nimetä eri rooleja kuten arkkitehti, ohjelmoija tai käyttöliittymäsuunnittelijoita. Tiimit kasataan henkilöistä, joilla on tarvittava kompetenssi ja osaaminen kyseessä olevan toiminnon tai ominaisuuden tuottamiseen. Tiimin jäsenet yhdessä rakentavat asiakasyrityksen määrittelemän tuotteen ja kaikilla on yhtä tärkeä osa kokonaisuutta.

Olennaista olisikin ymmärtää miten tehtäviä jaetaan yhteisöllisesti yksilön osaamiseen perustuen, tällöin vältytään perinteiseltä projektipompottelulta. Kukaan ei voi siis sanoa että ”tämä ei kuulu minulle, ja tuo on tämän tai tuon hommaa” kun kaikki tekevät kaikkensa projektin edistämiseksi. Karsitaan siis ylimääriset vaiheet ja roolitukset pois ja nähdään tarpeettomat työtehtävät jätteenä.

Scrum-prosessi

Prosessi alkaa tuotteen omistajan määrittämästä tuotteen työlistasta (Product Backlog). Tärkein ominaisuus tai toiminto, jonka on arvioitu tuovan eniten lisäarvoa ohjelmistolle tai sovellukselle pyritään laittamaan listan alkuun. Tämän jälkeen ominaisuudet tai toiminnot jaetaan jaksoihin tai julkaisuihin samaa periaatetta käyttäen, mutta niitä joudutaan useasti muuttamaan projektin aikana. (Schwaber, 2004)
Kuvassa 7 on esitetty perinteinen Scrumin prosessimalli.



Kuva 7, *Scrum-prosessi* (Schwaber, 2004)

Vaiheet

Visiointi - ennen projektia luodaan korkean tason visio, mitä projektilta halutaan ja miten siihen päästäisiin.

Työlistan muodostaminen – muodostetaan alustava tuotteen työlista tarvittavista ominaisuuksista ja tehdään sille priorisointikatselmus.

Sprintin suunnittelu – tehdään tarvittava työlista projektitiimille. Sprintin aikana vaatimusmäärittelyä ei voi tehdä, tai vaatimuksia muuttaa. Projektitiimi organisoi itsensä tarpeen mukaan vastaamaan sprintin työsuunnittelun odotuksia.

Sprintti – projektitiimi alkaa työstämään valittuja ominaisuuksia ja toiminnallisuuksia. Tiimillä on oikeus allokoida tarvittavat resurssit itsenäisesti, jotta tarvittava päämäärä saavutetaan sprintin aikana.

Päiväpalaveri – lyhyt tilannepalaveri, joka pidetään päivittäin. Palaveri on kestoltaan noin 15 minuuttia, ja jos mahdollista se pyritään pitämään seisten. Tarkoituksena on pitää yllä tilannekuvaa, josta saadaan kaikille tarvittava tieto missä projekti menee ja mitä akuutteja haasteita tai esteitä siinä on.
Palaveriin osallistuvat kaikki tiiminjäsenet ja Scrum-mestari. Jos jotain asiaa pitää erikseen käsitellä, niin siihen varataan toinen palaveriaika ja siihen osallistuvat vain asianosaiset ja Scrum-mestari.

Sprintin katselmointi ja jälkitarkastelu – Jokaisen sprintin jälkeen tiimi, Scrum-mestari ja tuotteenomistaja tarkastelevat saavutettua tulosta. Tiiminjäsenet kertovat tarvittaessa oman toimenkuvansa näkökantoja tai miten projekti on edennyt. Samalla arvioidaan seuraavan sprintin työlistaa ja tarvittaessa ominaisuuksien priorisointia muutetaan työlistassa. Lopuksi tuotteenomistaja päättää seuraavan sprintin sisällöstä ja aikataulusta. Prosessi käydään uudelleen läpi, kunnes lopuksi saavutetaan valmis käyttöönotettava tuote, jossa ominaisuudet ja toiminnot toteutuvat halutulla tavalla.

4.4 Scrum-yhteenveto

Scrumissa joukko käytäntöjä kohtaa valikoidun arvopohjan. Nämä arvot ovatkin kaiken Scrum-projektihallinnan ja aktiviteettien takana. Scrumissa ei takerruta perinteisiin työmäärittelyihin, jossa työtehtävät jaetaan roolienperusteella ja ainoastaan projektipäälliköllä on selkeä kokonaistilannekuva. Tiimi ja yksilöt sitoutuvat yhteiseen päämäärään, jossa Scrum-mestari on tukemassa ja antamassa tarvittavat keinot päästä tuohon päämäärään mahdollisimman vähäisillä häiriötekijöillä. Tuotteen työlista mahdollistaa ominaisuusarvojen ja tavoitteiden analysoinnin projektin aikana, jolloin ne ovat laadullisesti parempia ja kestävämpiä ratkaisuja kokonais- ohjelmistoa tai sovellusta tarkasteltaessa. (Schwaber, 2004)

4.5 Kanban-menetelmä

Scrumin lisäksi ketteränä menetelmä tarkastellaan Kanban-menetelmää, se onkin saanut huomattavaa kannatusta ohjelmisto ja sovelluskehityspiireissä.

Kanban on Japania ja tarkoittaa ”näkyvää korttia”. Menetelmä tai ajattelumalli on lähtöisin tuotantolinjalta sekä yhtenäisistä tuotantoprosesseista. Vahvana vaikuttajana Kanban menetelmässä on ollut Toyota Production System (TPS) kuten muissakin Lean-menetelmään pohjautuvissa ratkaisuissa. Perinteisessä tuotantoprosessissa voidaan mitata suoraan läpimenoaikaa esimerkiksi työvaiheelle tai menettelylle, jotka tapahtuvat konkreettisesti tuotantolinjalla. Ohjelmisto- ja sovelluskehitystyössä, joka määritellään luovaksi tietotyöksi, ei esiinny niin useasti työvaiheen tai menettelyn samankaltaisuutta. Pääsääntöisesti luovassa työssä arvopohja on erilainen, koska mitattava arvo ei ole tuotantolinjan vaihtelun ja hukan optimoimista. Arvopohja syntyykin kyvyllä ratkaista ongelmia tai haasteita, ratkaisuprosessin optimoimisella pyritäänkin lyhentämän tuotteen kehityksen läpivientiaikaa ja siten ”tehostamaan” tuotantoprosessia. Kanban-menettelyssä perusperiaatteena on jatkuva prosessin kehittäminen, oli kyse sitten luovasta tai tuotannollisesta työstä.

Kanban-periaatteet ohjelmisto- ja sovelluskehityksessä

Kanban-menetelmän ohjelmisto- ja sovelluskehitysalalla kuvasi David Anderson vuonna 2003. Menetelmää on sen jälkeen päivitetty ja siihen on lisätty ajantasaisia

elementtejä, nykyinen Kanban-menetelmä sisältää neljä perusperiaatetta ja kuusi käytäntöä. (Anderson, 2010)

Perusperiaatteet:

1. *Aloita siitä mitä tiedät.*
   Kanbanin joustavuus mahdollistaa sen päällekkäisyyden olemassa olevien prosessien ja työvaiheiden kanssa häiritsemättä niitä liikaa. Tarkoituksena olisikin arvioida asioita niin että ne korostaisivat mahdollisimman pientä muutosta ja poikkeamaa jo hyväksi koetussa käytännöissä. Kanbanin vahvuutena onkin sen monimuotoisuus. Se mahdollistaa muutoksien vaiheittaisen käyttöönoton hyvin erityyppisissä organisaatioissa ja työyhteisöissä, luomatta vahvaa muutoksen pelkoa.

2. *Tavoittele jatkuvaa ja vaiheittaista muutosta.*
   Kanban-menetelmän tarkoitus olisikin tuottaa mahdollisimman vähäistä vastustusta, ja siten edistää ja kannustaa pieniin vaiheittaisiin muutoksiin nykyisessä prosessissa.

3. *Arvosta nykyistä prosessia, rooleja ja vastuuta.*
   Kanban tunnistaa arvon olemassa olevissa prosesseista, rooleista, toimenkuvissa ja vastuissa ja siten haluaakin säilyttää niitä. Kanban kannustaakin loogisen, systemaattisen tai vaiheittaisen muutoksen tekemiseen, jos sille on tarvetta.

4. *Kannusta johtajuutta kaikilla tasoilla.*
   Kanban-menetelmässä kannustetaan yksilöitä tarttumaan muutosajatteluun joka päivä. Ei siis takerruta perinteiseen johtamishierarkiaan, jossa päätökset ja muutokset voidaan viedä läpi vain johtoryhmässä. Tärkeää olisikin, että kaikilla on asenne ja mielentila kunnossa lähestyä prosessia vaiheittaisenmuutoksen kautta.

Käytännöt:

1. *Visualisoi työnkulku ja työvaiheet.*

   Perusta Kanban-taulu, sen tehtävä olisi kuvata nykyinen työnkulku tai prosessinkierto. Helpoiten tämän voi mallintaa laittamalla tauluun sarakkeita ja rivejä, jotka kuvaavat jotain yksittäistä työnvaihetta. Kun visualisoidaan nykyinen prosessi, voidaan siihen liittää asioita tai elementtejä, joiden avulla säädetään muutosta.
   Kanban-taulusta lisää alempana aineistossa.

2. *Rajoita keskenolevan työn määrää.*

   Kanban-menetelmässä esiintyy termi Work In Progress (WIP). Se tarkoittaa suurinta määrää tehtäviä, jotka voivat olla yhdessä sarakkeessa kerrallaan Kanban-taulussa. Jos tiimin tehtäviä muutetaan kesken työjaksotuksen, suurella todennäköisyydellä aiheutetaan prosessille Lean-menetelmässä kuvattua jätettä. Työmäärä (WIP) ei voisi siis ylittää tiimin kokoa, koska se olettavasti hidastaisi koko prosessia. Luovassa tietotyössä työtehtävät ovat hyvinkin erimittaisia verrattuna tuotantotyöhön, joka pitäisi pyrkiä ottamaan huomioon työmäärien suunnittelussa.

3. *Hallitse työnkiertoa.*

   Kanban menettelyssä keskeinen asia onkin hallita kokonaistyönkiertoa. Sen saumaton ja esteetön toiminta lisää tuottavuutta ja arvoa. Ajatuksena olisikin hallita työvaiheita, niin että ne luovat pohjan itsenäiselle, itseohjautuvalle ja luovalle tietotyölle. Ei siis mikro-hallita ihmisten tekemistä, vaan sovitaan tehtävät työvaiheet kokonaisprosessissa oikein, jolloin ne tuottavat mahdollisimman nopeasti arvoa.

4. *Tee prosessikäytännöistä tarkkoja*.

   Et voi kehittää tai parantaa jotain mitä et ymmärrä. Pitäisikin pystyä kuvailemaan nykyiset prosessit olemassa olevassa organisaatiossa mahdollisimman selkeästi ja julkaisemaan ne siten kaikille. Ihmisillä ei ole tapana asioida tai osallistua käytäntöihin, joihin he eivät usko tai eivät näe ja koe niitä.

5. *Palautteen käsittely.*

   Jotta positiivisen muutoksen voisi saada aikaiseksi, tulisi siitä saada palautetta. Lean-menetelmä nojaakin vahvasti palautteeseen ja siihen että se tarvitaan muutoksen aikaansaamiseksi. Palaute voi olla lyhyt palaverikäytäntö tai jokin muu kommunikointi välineitä hyväksikäytettyjä menetelmiä, mutta pääsääntöisesti sitä pitäisi saada useasti.

6. *Lisää yhteistoimintaa*

   Kanban-menetelmässä tulisi jakaa yhteinen visio koko organisaatiossa, mitä tavoitellaan ja miten siihen päästäisiin. Se on yksi keino saada jatkuvaa vaiheittaista muutosta, niin että se olisi kestävällä pohjalla. Tiimeillä on useasti jaettu käsitys tai ymmärrys töistä, työnkierroista ja prosesseista. Näiden tulisikin jalostua käytännöksi, miten vähentää riskiä ja rakentua malliksi ratkaista ongelmia tulevaisuudessa. Kaikki tämä on askeleita kohti vaiheittaista muutosta ja kehitystä.

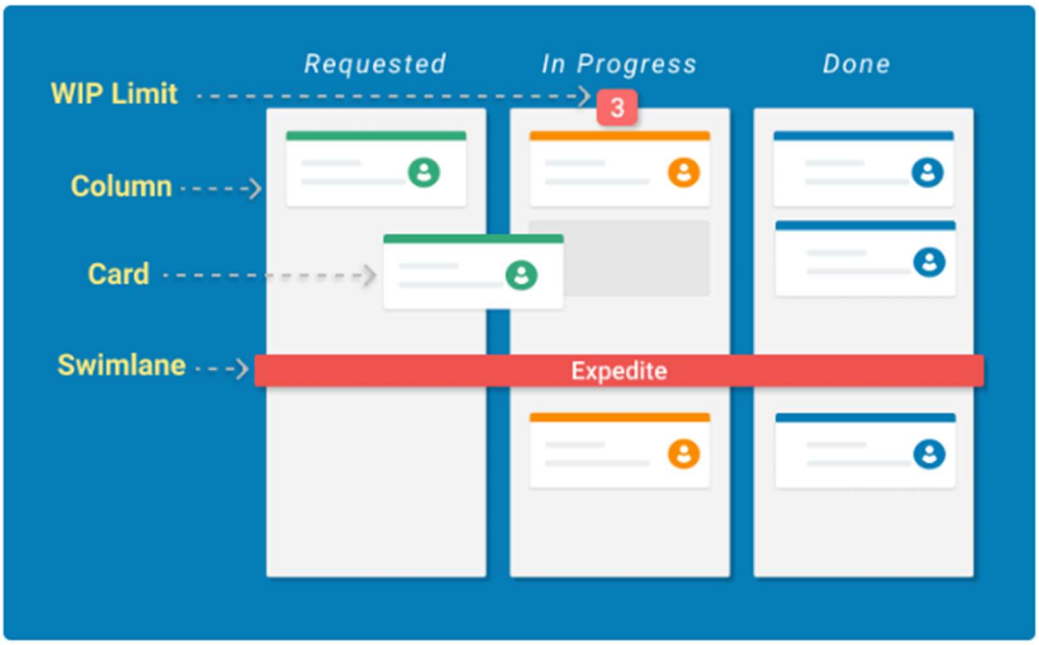
Kanban-taulu

Jonkinasteinen tehottomuus voi olla laaja-alainen ongelma projekti ja prosessityyppisessä työskentelyssä. Monesti ongelma voi johtua, että ei ole selkeää kokonaiskuvaa mitä pitäisi tehdä ja milloin. Tällöin seurauksena voi olla prosessin läpiviennin hidastuminen. Kanban-taulu mahdollistaa läpinäkyvyyden prosessiin ja työn alla oleviin tehtäviin. Taulusta paljastuu mahdollisia pullonkauloja, mihin tulisi panostaa ja tehdä tarvittavia muutoksia.

Kanban-taulu onkin visuaalinen tapa esittää työvaiheita. Käytännössä tämä voi olla perinteinen taulu, johon kiinnitetään paperilappuja. Kun perinteinen taulu käännettiin digitaaliseksi työkaluksi, siitä syntyi yksi tehokkaimmista ketterän menetelmän projektihallintatyökaluista.

Taulun komponentit.

Kuvassa 8 on esitetty perinteinen yksinkertaistettu malli kanban-komponenteista.



Kuva 8, *kanban-taulu ja komponentit* (Kanbanize, n.d.)

Kanban-kortti – visuaalinen esitystapa tehtävälle. Jokainen kortti sisältää tarvittavat tiedot kyseisestä tehtävästä, kuten aikataulun, kenelle se kuuluu, tarkemman kuvauksen.

Kanban sarkkeet – Jokainen sarake esittää erivaiheita työkulussa tai prosessissa. Kortit kulkevat sarakkeiden läpi, kunnes ne ovat kokonaan valmiita ja poistuvat taululta.

Kanban-työmäärä rajat (WIP) – Suurin mahdollinen työmäärä, joka rajoitetaan tiimin koon mukaan. Jokaisella vaiheella voi olla yksilöllinen määrä WIP-rajoitteita, jotka esitetään vaiheen yläpuolella. Rajoittamalla työmääriä pystytään periaatteessa nopeuttamaan muiden tehtävien läpimenoa aikoja.

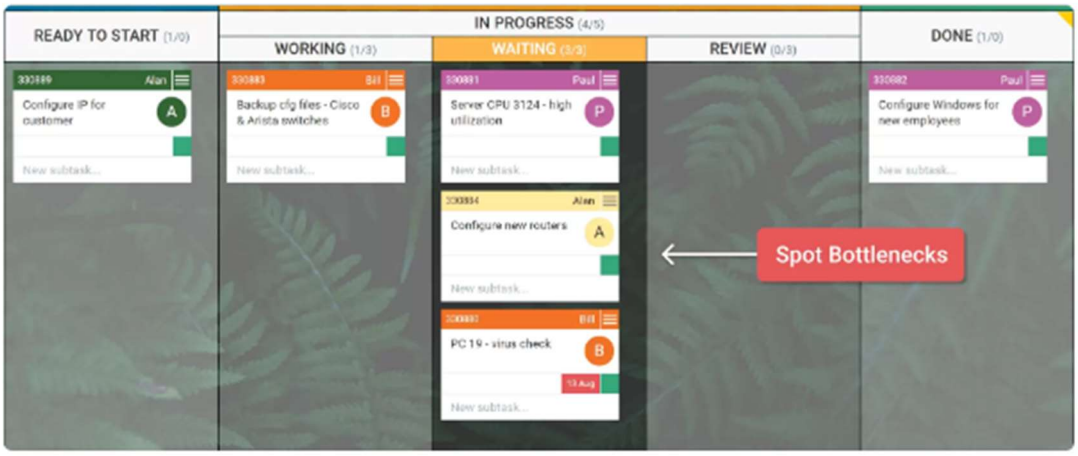
Kanban-uimarata – Horisontaalinen linja, jolla voidaan erotella erityyppisiä aktiviteetteja kuten erimittaisia työmääriä, irrallisia työtehtäviä, korjauksia yms. Uimarataa voidaan jossain tapaukissa käyttää myös ohituskaistana.

Kanban-taulun toiminta

Taulun toiminta on hyvinkin yksinkertainen ja siten melko tehokas. Koska työmäärät (WIP) on rajoitettu tiimin koon mukaan, ei pitäisi prosessissa syntyä ylituotantoa. Työmääräarvio (WIP) näkyy useasti sarakkeen tai taulun yläosassa.

Kun työvaihe tai tehtävä laitetaan taululle, se kulkee sille osoitetun prosessin läpi. Kun jokin kortti eli tehtävä siirtyy taululla seuraavaan sarakkeeseen. Sen imuohjauksessa voidaan edellisistä vaiheista tuoda uusikortti tai tehtävä. Tämän tyyppisessä imuohjauksessa seuraavavaihe on aina tavallaan asiakas, jota edellinen vaihe palvelee. Näin ollen kriittiset tehtävät ja niiden riippuvuussuhteet saadaan valmiiksi mahdollisimman pian.

Uimarata mahdollistaa käytännön, jolla esimerkiksi kriittinen korjaus saadaan nopeasti vietyä muiden ohitse. Tämä vaatisi käytännössä sopimuksen kuinka paljon tehtäviä voidaan uimaradalle laittaa, muutoin vaarana on rikkoa perusjärjestelmä.
Kuvassa 9 on malli, jossa työmäärä on kasvanut liian suureksi yhdessä työvaiheessa ja aiheuttaen pullonkaulan koko prosessille.



Kuva 9, *Kanban-taulu – paikanna pullokaulat (kanbanize.com)*

4.6 Kanban-yhteenveto

Nykypäivänä useat organisaatiot käyttävät kanban-menetelmää sen tuoman ketteryyden ja projekti kaaoksen hallittavuuden vuoksi. Käytännössä jos Kanban on tehty

oikein se mahdollistaa kaikille projektinjäsenille kokonaiskuvan tarkastelun ja tehtävät eivät jää suorittamatta. Oikein käytettynä ja henkilöiden sitouduttua menetelmään pitäisivät prosessin pullonkaulat ja haasteet nousta esille. Siten ongelmien ja muutoksien ratkaisu helpottuu, kun selkeässä konkreettisessa kokonaiskuvassa esitetään, miten asiat etenevät.

Menetelmä ei ota kantaa olemassa oleviin prosesseihin, vaan pyrkiikin vaiheittaiseen parantamiseen. Siten menetelmä myös pyrkii osaltaan vähentämään organisaation muutoksenpainetta. Palautteen ansiosta asioihin, jotka vaikuttavan arvoketjun edistämiseen tuleekin selkeämpi läpinäkyvyys kokonaiskuvaa tarkasteltaessa.

Poimintoja Kanban ”virheistä”

Arvosta arvoketjua ja prosessikarttaa (Value Stream Map). Analysoi ja kokeile miten tehtävät ja toiminnot kulkevat visuaalisen prosessin läpi, ennen kuin jalkautat sen tuotantoon. Pyritään tunnistamaan mahdollista Lean-menetelmän jätettä mahdollisimman aikaisessa vaiheessa. (Wingfield, 2010)

Älä yritä mallintaa nykyistä prosessia täydellisesti. Tekemällä 6 metriä pitkän taulun täynnä niche-tyyppisiä vaiheita ja opettamalla uudelle henkilölle tunnin ajan, miten taulua tulisi käytetään ja miten tehtäviä liikutellaan. Liian monimutkainen prosessinkuvaus, poistaa helppouden koko Kanban-menettelyssä. Yksityiskohtainen työvaiheiden esittäminen kanban-taulussa tuo esille myös *ei toivotun* työskentelytavan. Miksi liikuttaa tehtäviä kohta kerrallaan, kun voi tehdä kaikki vaiheet kerralla valmiiksi ja siirtää lapun alkupäästä suoraan loppuun. Mitään näkyvyyttä siis ei varsinaisesti ole prosessin kokonainaiskuvaan, mitä vaiheissa ”tapahtuu” ja ovatko ne mahdollisia pullonkauloja. (Wingfield, 2010)

Kunnioita työmääräarviota (WIP). Rajaa siis määrällisesti paljonko tehtäviä voidaan realistisesti suorittaa tai toimittaa prosessin vaiheessa. Älä sorru alikuormittamaan tai ylikuormittamaan resursseja, optimoi. (Wingfield, 2010)

4.7 Scrum ja Kanban vertailutaulukko

Lyhykäisyydessään tässä aineistossa ei vertailla ketteriä menetelmiä keskenään kovinkaan paljoa. Tarkoitus olikin antaa parista nykyisestä ja laajalti käytössä olleesta menetelmästä läpileikkaus.

Vertailuna käytetään Knibergin ja Skarin (2010) tekemää taulukkoa 2 menetelmien eroista. Scrumin haasteena on sen sopeutumisen riittämättömyys tarkoittaen itseohjautuvien tiimien kykyyn luoda ja kehittää prosesseja niiden ulkopuolella ja arvonhallintaa sprintissä. Tämä korostuu tilanteissa jossa tiimin pitäisi tehdä tutkimus ja kehitystyötä laadun tai arvon varmistamiseksi. Tämä ei tuota asiakkaalle lainkaan lisäarvoa. Kritiikkiä antanut Kniberg ja Skarin (2010) pitävät keskeisenä perusteena sprinttien rajoitettua aikamäärää, se ei mahdollista tarvittavaa mukautuvuutta projektissa. (Kniberg, 2010)

Taulukko 2, *Scrumin ja Kanbanin eroja (Kniberg & Skarin, 2010, s. 50)*

Scrum	Kanban
Aikarajoitetut iteraatiot pakollisia.	**Aikarajoitetut iteraatiot mahdollisia.** Suunnittelu, julkaiseminen ja prosessin kehittäminen voidaan suorittaa eri rytmissä. Aikarajoiteen sijaan voidaan käyttää tapahtumalähtöistä lähestymistapaa.
Tiimi sitoutuu tiettyyn työmäärään iteraatiokohtaisesti.	**Sitoutuminen valinnaista.**
Käyttää **nopeutta** (velocity) suunnittelun ja prosessikehityksen oletusmittarina.	Käyttää **läpimenoaikaa** (lead time) suunnittelun ja prosessikehityksen oletusmittarina.
Toiminnallisesti monipuolisten tiimien käyttö pakollista.	Toiminnallisesti monipuolisten tiimien käyttö valinnaisia. **Spesialistitiimit sallittu**.
Töiden koko täytyy suhteuttaa toteutettavaksi yhden sprintin sisällä	Ei määrittele mitään erityistä työn kokoa.
Sprintin edistymiskaavion (burndown chart) **käyttö määrätty.**	Ei määrittele minkään erityisen kaavion käyttöä.
Käynnissä olevien **töiden määrän rajoittaminen epäsuorasti** (sprinttiä kohden)	Käynnissä olevien **töiden määrän rajoittaminen suoraan** (työvaihetta kohden)
Arvioinnit (estimation) **pakollisia**.	**Arvioinnit** (estimation) **valinnaisia**.
Käynnissä olevaan iteraatioon ei voi lisätä töitä.	**Uusia töitä voi lisätä milloin tahansa, kun resursseja on vapaana.**
Sprintin työlistan (backlog) omistaa yksi tiimi.	**Useammat tiimit tai yksilöt voivat jakaa Kanban-taulun.**
Määrittelee kolme roolia.	**Ei määrittele mitään rooleja.**
Scrum-taulu tyhjennetään sprinttien välillä.	**Kanban-taulu on pysyvä.**
Määrittelee priorisoidun työlistan (backlog) tuottelle.	**Priorisointi on valinnaista.**

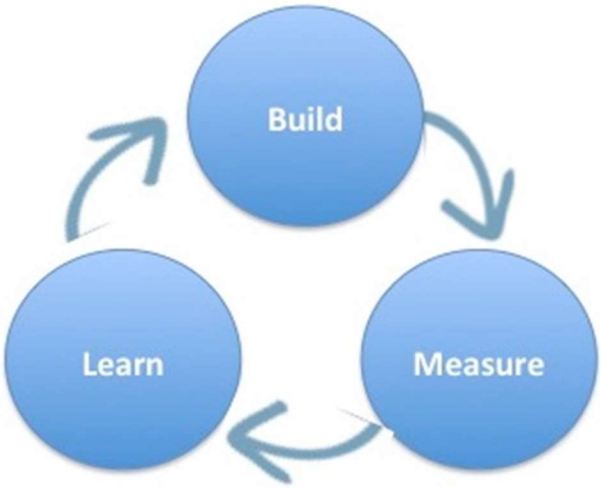
4.8 MVP-malli (Minimum Viable Product)

Termi MVP (Eng. Minimum Viable Product) voidaan vapaasti Suomentaa ”pienin julkaisukelpoisin tuote”. Käytännössä termiä sovelletaan nykyaikana kuvaamaan melkein mitä tahansa vaihetta ohjelmisto ja sovelluskehityksessä. Yhdelle se tarkoittaa konseptia, toiselle taas prototyyppiä ja kolmannelle ensimmäistä julkaistavaa versiota tuotteesta.

Termi tuli tunnetuksi Eric Riesin (2009) The Lean Startup-kirjan myötä, kirjassa kuvataan määritelmä näin: MVP on versio tuotteesta tai palvelusta, joka mahdollistaa enimmäismäärän validoidun oppimisen haalimista vähimmäisresursseilla. Kyseessä siis on oppimisprosessi. (Ries, Startup Lessons Learned: Minimum Viable Product: a guide, 2009)

Termi ei nimestään huolimatta *aina* kuvaa varsinaista tuotetta, kyseessä voi esimerkiksi olla strategian testaus tai idean esittäminen. Ilmiönä se perustuu olettamukseen, onko esimerkiksi asiakasyrityksellä löydetty tarpeeksi suuri ongelma ja riittävän hyvä ratkaisu siihen. Silloin voidaan nopeasti ja ketterästi mallintaa tuoteidean markkinakelpoisuus.

Ydinajatuksena MVP ratkaisuissa onkin soveltaa Startup mentaliteettia kestävän liiketoiminnan tueksi. Prosessina hyödynnetään rakenna – mittaa – opi palautesykliä, jossa nopeakierto maksimoi oppimisen ja toiminnan kasvuhypoteesit. (Blank, 2015)
Kuvassa 10 on esitetty rakenna – mittaa – opi palautesykli.



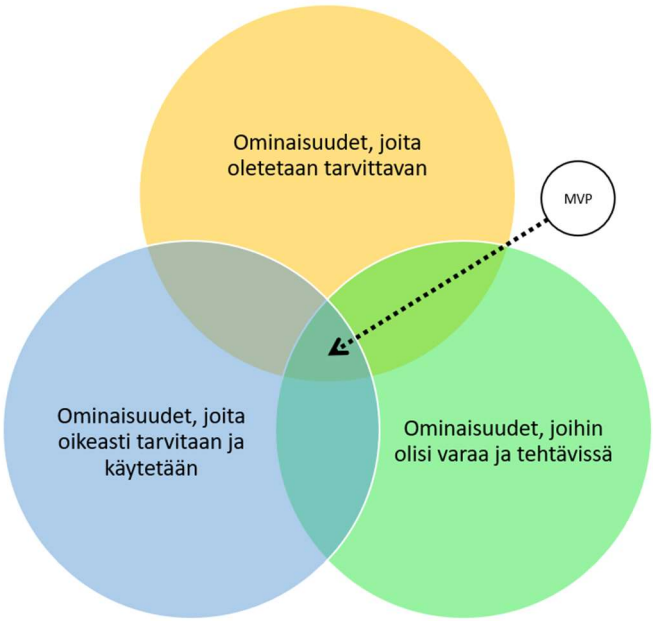
Kuva 10, *Rakenna-mittaa-opi-palautesykli* (Blank, Why Build, Measure, Learn, 2015)

Asiakasyritystyyppisissä ympäristöissä haasteeksi asettuukin, kuinka saadaan ideat testattua mahdollisimman halvalla ja nopeasti. Yhtenä vaihtoehtona on tehdä perustutkimus idealle ja työstää sitä perinteisen vesiputousmallin mukaisesti, toisena

vaihtoehtona on tehdä ominaisuus- ja tarvekartoitus. Kartoituksella pyritään löytämään jokin keskeinen asia, jota voidaan tuotteistaa ja kokeilla toimiiko kuten on ajateltu.

Ennen MVP ratkaisun luomista olisikin hyvä asiakasyrityksen perustella, miksi testataan, mitä testauksella haetaan ja miten testataan. Lopuksi pitäisi kyetä arvioimaan miten tuloksia tulkitaan, ja tukevatko ne alkuperäistä ideaa. (Blank, The Lean Approach: Minimum Viable Product, 2014)

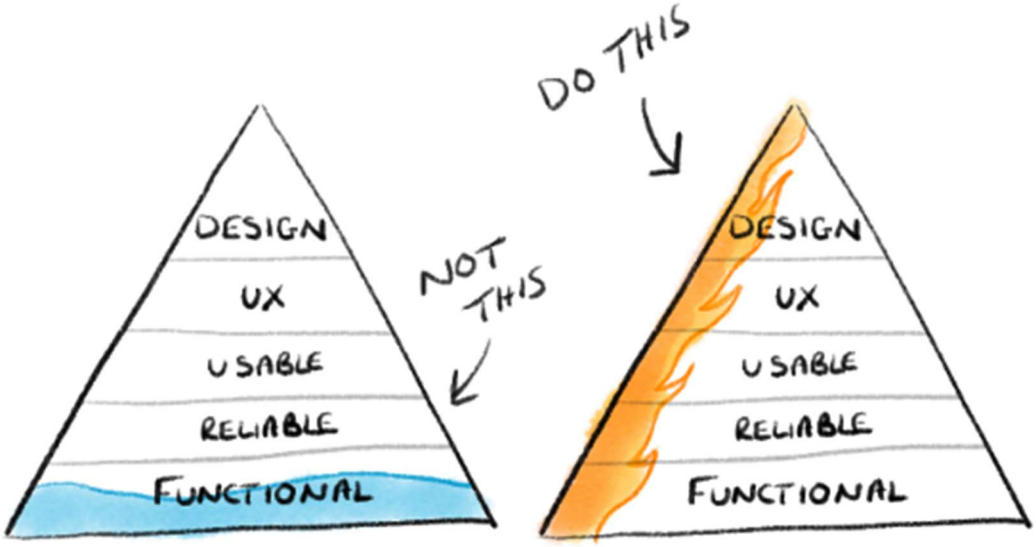
Kuva 11 on yksinkertaistettu diagrammi MVP ratkaisun ominaisuus ja tarvekartoituksesta.



Kuva 11, *MVP-ominaisuus ja tarvekartoitusdiagrammi.*

Kun asiakasyritys on päässyt mielestään tarpeeksi lähelle keskeistä tarvetta MVP-ratkaisulle, olisi hyvä tarkastella, miten kyseinen ratkaisu toteutetaan. Monesti lähdetään tekemään esimerkiksi pelkästään käyttöliittymäsuunnittelua, jolla kuvattaisiin idean markkinakelpoisuutta. Pelkkien toiminnallisuuksien esittäminen MVP-ratkaisussa ei monesti tuota tarvittavaa oppimäärää, jolla haluttu ratkaisu voitaisiin jalostaa tarvittaessa oikeaksi tuotteeksi.

Pitäisikin valita tarvittava määrä ydintoimintoja, jolla saadaan oppimismäärä ja tavoitteet kartoitettua ja todennettua tarvittavalla tasolla. Loppukäyttäjien palaute onkin tärkeässä roolissa, jotta saadaan tärkeimpien toiminnallisuuksien ominaisuudet määriteltyä lopulliseen tuotteeseen. Rajaamisen pohjana voidaan hyödyntää 20-80-ajattelumallia. Millaisia ovat ne 20% mahdolliset toteutuksen vaatimat ratkaisut, jotka täyttäisivät 80% käyttäjäntarpeesta. Kuvassa 12 esitetään ydintoimintojen tärkeys MVP-ratkaisuissa.



Kuva 12, *MVP ydintoiminnot ja toiminnallisuudet* (Tinytracker, n.d.)

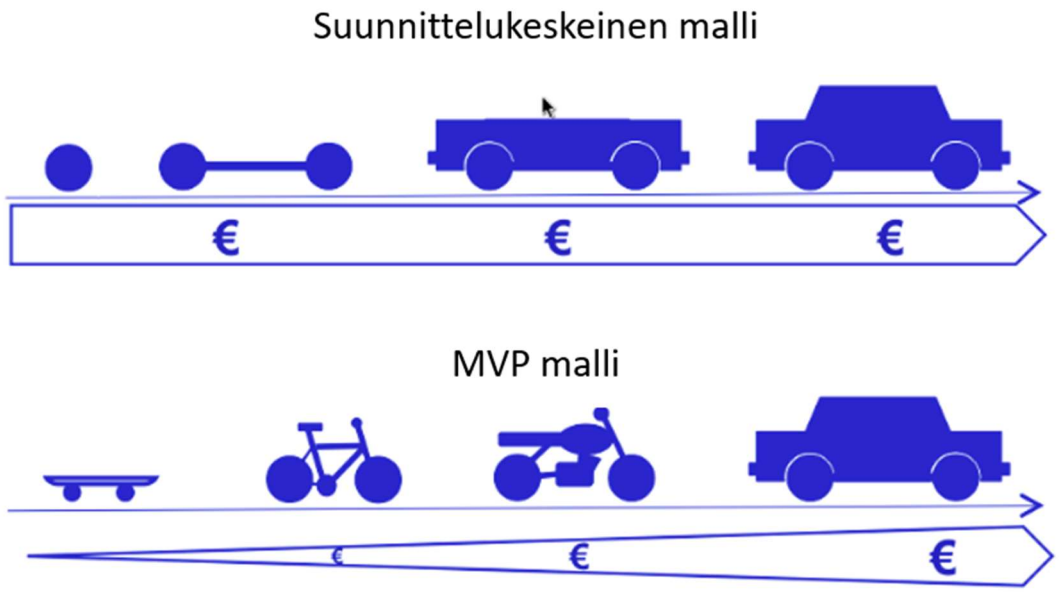
4.9 MVP yhteenveto

Riesin (2011) mukaan MVP:n onkin tarkoitus epäonnistua. Epäonnistumisen kautta saadaan enemmän arvokkaampaa palautetta kuin suoraan menestykseen johtaneella kokeilulla.

Steve Blank (2014) kirjoittaa blogissaan, ettei MVP ratkaisulla aina tarvitse olla suoraa yhteyttä tuotteen lopputulokseen. MVP:n tärkein arvo onkin sen ominaisuudessa tuottaa maksimaalinen oppimistulos, tarkoittaen empiiristä tiedonkeräämistä joko asiakkailta tai asiakasyrityksen sisäisiltä toiminnoilta. (Blank, The Lean Approach: Minimum Viable Product, 2014)

MVP:tä käytetäänkin yleisesti kuluttajille suunnatuissa ratkaisuissa hypoteesien todentamiseen jollain konkreettisella tavalla. Tämä antaa kehittäjille mahdollisuuden reagoida palautteeseen ennen sitoutumista kalliisiin sovellusprojekteihin. Ratkaisumallina toimiikin monesti nopein ja kevein ydintoiminnot sisältävä ohjelmisto- tai sovellusrajapinta, jolla saavutetaan tarvittava oppimistulos.

Asiakasyrityksen kannalta MVP ratkaisumalli on yksi keino välttyä luomasta ohjelmisto- ja sovelluskehitysprojektien jätettä. Se myös tuottaa oppimistuloksen, jota voidaan hyödyntää päätöksenteon välineenä.



Kuva 13, *MVP ratkaisu vastaan suunnittelukeskeinen malli* (Kehmet, 2019)

5 LOPPUPÄÄTELMÄ

Aineiston tavoitteena oli tutkia ilmiötä ohjelmisto- ja sovelluskehityshankkeiden ympärillä, ja pyrkiä ilmaisemaan *miten* päästään arvoa tuottavaan lopputulokseen. Teoriaosiossa käsiteltiin muutamia tunnettuja ja tutkittuja ilmiöitä, joilla voi olla suoranaisia vaikutuksia ohjelmisto- ja sovelluskehityshankkeissa.

Uusien digitaalisten kehityshankkeiden alkaessa ja niitä ideoitaessa tiedostettaisiin aineistossa esiintyneitä ominaispiirteitä ohjelmisto- ja sovelluskehitysprojekteille. Huomioimalla ohjelmisto- ja sovelluskehityshankkeiden uniikin luonteen ja välttämällä tiettyjen syyperäisten vaikutteiden esiintymistä projektia aloitettaessa ohjelmistojen onnistuminen on hyvinkin todennäköistä ja niiden kehittäminen tulevaisuudessa kannattavaa.

Oleellista olisi arvon tunnistaminen ohjelmistoprojektin tavoitteena. Arvo syntyy useasti jonkin tarpeeksi suuren ongelman löytymisestä ja sen toimivasta ratkaisusta. Ongelmanratkaisu arviossa pyrittäisiin tunnistamaan myös laadullisia vaikuttajia, joita useasti ovat käytettävyys, suunnittelun oikeellisuus ja tehokkuus. Projektien laadulliseen lopputulokseen vaikutti myös tutkimuksen perusteella merkittävästi käyttäjien palaute ja johdon tuki.

Tärkeänä vaikuttajana ohjelmisto- ja sovelluskehityshankkeissa ovatkin ihmiset. Tunnistetaankin, että kaikki projektiin liittyvät päätöksentekijät ovat laatuun vaikuttavia tekijöitä. Hyvilläkin työntekijöillä epäonnistumisen riski kasvaa ilman kunnollista prosessia.

Asiakasyrityksen kohdalla olisikin hyvä pohtia, miten saataisiin tulevaisuudessa tarvittava tasapaino ja osaaminen perinteisen ja ketterien menetelmien välillä. Uudenlaisilla käytännöillä ja menetelmillä voitaisiin pyrkiä tuottamaan moderneja digitaalisia tuotteita nopeammin perinteistentuotteiden rinnalle ja niiden tueksi.
Prosessi vaikuttaa kuitenkin lopputuloksen laatuun, tarkasteltaisiin siis avoimesti digitaalisten kehityshankkeiden luonnetta. Siten pyrittäisiin löytämään sopivin menetelmä asiakasyritykselle, jolla saavutettaisiin edullisin ratkaisu organisaatiossa sekä tuotteen kehityksessä.

Tulevaisuudessa isoin muutos perinteisessä konepajaliiketoiminnassa vaan voikin olla projektihallinnan ja projektipäälliköiden siirtyminen uusiin menetelmiin ja käytäntöihin puhuttaessa modernisaatiosta. Silloin vaadittaisiin aivan uudenlaista ajattelua ja arvojen määrittelyä organisaatioissa. Ohjelmistojen ja sovelluskehityshankkeisiin pitäisikin löytää sopiva projektinhallinnan lähestyminen, jossa ei keskitytä pelkästään projektin kontrolliin. Prosessissa ei myöskään takerruttaisi liiaksi terminologiaan ja

hienoihin menetelmäkuvauksiin, vaan keskityttäisiin arvoon ja lopputulokseen soveltaen parhaiksi havaittuja käytäntöjä.
Käytännössä asiakasyrityksen keinoksi tähän haasteeseen voisi tulla osaamiskartan arviointi organisaatiossa. Tulevaisuudessa pyrittäisiin siten mahdollisesti lisäämään osaamista ja koulutusta teknologia-alan menetelmien osalta. Siirtyminen olisi vaiheittaista, jossa poistuttaisiin matalan tason insinööritaidoista kohti uusia menetelmiä ohjelmisto- ja järjestelmäsuunnittelussa tai tietojenkäsittelyssä. Modernit ja uudet menetelmät eivät saisi myöskään rikkoa olemassa olevia käytäntöjä ja prosesseja. Organisaatiot ovat mahdollisesti käyttäneet satoja tai tuhansia miestyötunteja hioakseen ydinliiketoiminnan tarvitsemat menetelmät huippuunsa pärjätäkseen markkinoilla.

Asiakasyrityksen digitaalisten kehityshankkeiden päätöksenteon tueksi aineistossa pyrittiin tuomaan esille yleisiä ratkaisuja ja käytäntöjä. Ensimmäisenä esiteltiin teknisen velan teoreettinen pohja ja sen lisäksi toisena työvälineenä MVP ratkaisumalli.

MVP eli Minimum Viable Product tulisi olla osana uudenlaista prosessiajattelua, jolla pyrittäisiin kaventamaan perinteisen kontrollisuunnittelu ja ketterien menetelmien toteutuksien välistä kuilua digitaalisissa kehityshankkeissa. Ongelmanratkaisu ei välttämättä aina löydy tutkitusta tiedosta ja valmiista konsepteista. Tekemällä tietoisesti empiirisentutkimuksen MVP-ratkaisun avulla saadaan epävarmuustekijöitä karsittua tuotteen ominaisuusmäärittelystä ja lisäksi maksimaalinen oppimistulos. Tuloksella on jo siten itseisarvoa ja se palvelee pidemmällä aikajaksolla kokonaissuunnittelua ohjelmisto- ja sovelluskehityksessä.

Koska tekninen velka on melko abstrakti käsite, sitä ei esiinny samalla tavalla perinteisissä järjestelmätoteutuksissa kuin ohjelmisto- ja sovelluskehityksessä. Asiakasyrityksen päätöksenteko työvälineenä sitä voitaisiin hyödyntää ohjelmisto- ja järjestelmäprojekteja arvioitaessa. Pyrittäisiin tekemään velan havainnointi osaksi prosessin vaiheita kuten suunnittelu, olemassa olevat järjestelmät, esitutkimus ja toimintojen kuvaus. Silloin mahdolliset muutokset ja tarpeet eivät mahdollisesti toisi kustannuspaineita kehityshankkeissa niin voimakkaasti ja olisivat helpommin ennakoitavissa.

Ohjelmisto– ja sovelluskehityshankkeissa tulisikin ottaa kokonaisuuksia huomioon. Puhuttiin sitten itse tehdyistä asiakasyrityksen omista sovelluksista tai ulkoa

hankituista standardiratkaisuista, ilman asiakasyrityksen linjausta liiketoimintayksikön kaupallisten ja tietojärjestelmien suuntaamisesta oikein ei ole tulevaisuudessa digitalistenhankkeiden läpivieminen kovin helppoa.

Tulevaisuus vaikuttaisi olevan valoisa konepajatyyppisille yrityksille, jotka ponnistelevat informaatioteknologioiden ja modernisaation kehityskaaressa. Uusien menetelmien ja hyväksi koettujen käytäntöjen implementointi tuotekehityksessä, siivittääkin perinteistä teollisuusalaa eteenpäin. Vauhdittamassa tätä siirtymää on muun muassa siviili kehittäjät (Eng. Citizen developers) ja *Low-Code, No-Code* palvelualustat jotka tuovat aivan uudenlaisia työkaluja ja notkeita toteutustapoja liiketoimintaympäristön digitaaliseen murrokseen.

6 LÄHTEET

Agile Alliance. (ei pvm). *Manifesto for Agile Software Development*. Noudettu osoitteesta Agile Manifesto: https://agilemanifesto.org/

Alves, N. &. (2014). Towards an Ontology of Terms on Technical Debt. Noudettu osoitteesta https://www.researchgate.net/publication/286010286_Towards_an_Ontology_of_Terms_on_Technical_Debt

Anderson, D. J. (2010). *Kanban: Succesful Evolutionary Change for Your Technology.* Blue Hole Press.

Blank, S. (2014). *The Lean Approach: Minimum Viable Product*. Noudettu osoitteesta Https://www.youtube.com/watch?v=xxjbxk8dUqI.

Blank, S. (2015). Why Build, Measure, Learn. *isn't just throwing things against the wall to see if they work*. Noudettu osoitteesta Https://steveblank.com/2015/05/06/build-measure-learn-throw-things-against-the-walland-see-if-they-work/.

Cobb, C. G. (2014). *The Project Manager's Guide to Mastering Agile.* John Wiley & Sons. Noudettu osoitteesta http://ebookcentral.proquest.com/lib/samk/detail.action?docID=1895876

Cockburn, A. (2006). *Agile software development : the cooperative game.*

Damasiotis, V. &. (2017). Analysis of software project complexity factors. Researchgate. doi:10.1145/3034950.3034989

Haikala&Merijärvi. (2004). *Ohjelmistotuotanto* (Osa/vuosik. 10). Talentum.

Kanbanize. (ei pvm). *Kanbanize*. Noudettu osoitteesta https://kanbanize.com/

Kehmet. (2019). *MVP - pienin julkaistava tuote*. Noudettu osoitteesta https://kehmet.hel.fi/menetelmalaari/mvp/

Kerzner, H. e. (2010). *Managing Complex Projects.* ohn Wiley & Sons, Incorporated. Noudettu osoitteesta http://ebookcentral.proquest.com/lib/samk/detail.action?docID=573717

Kniberg, H. &. (2010). *Kanban and Scrum - making the most of both.* Publisher of InfoQ.com.

Massotte, P. a. (2017). *mart Decisions in Complex Systems.* ohn Wiley & Sons, Incorporated. Noudettu osoitteesta http://ebookcentral.proquest.com/lib/samk/detail.action?docID=4908155

Maylor, H. V. (2008). *Managerial complexity in project based operations: a grounded model and its implications for practice.*

McConnell, S. (2007). Technical Debt. 10x Software Development. Noudettu osoitteesta http://blogs.construx.com/blogs/stevemcc/archive/2007/11/01/technical-debt-2.aspx

McCormick, S. (2017). Why the way we look at technical debt is wrong. Noudettu osoitteesta https://www.bigeng.io/why-the-way-we-look-at-technical-debt-is-wrong/

McManus, J. W.-H. (2004). Understanding the Sources of Information Systems Project Failure. Noudettu osoitteesta https://www.researchgate.net/profile/John_Mcmanus21/publication/329539985_A_study_in_project_failure/links/5c1138aea6fdcc494fef184b/A-study-in-project-failure.pdf

Measey, P. (2015). *Agile Foundations : Principles, practices and frameworks.* BCS Learning & Development Limited. Noudettu osoitteesta http://ebookcentral.proquest.com/lib/samk/detail.action?docID=1759633.

Poppendieck, M. (2003). *Lean Software Development.*

Ries, E. (2009). *Startup Lessons Learned: Minimum Viable Product: a guide.* Noudettu osoitteesta Http://www.startuplessonslearned.com/2009/08/minimum-viableproduct-guide.html.

Ries, E. (2011). The Lean Startup. How Today's Entrepreneurs Use Continous innovation to create redically succeful businesses.

Schwaber. (2004). *Agile project management with Scrum.* Noudettu osoitteesta https://www.dawsonera.com/readonline/9780735636002

The Standish Group. (2014). CHAOS Report. Noudettu osoitteesta https://www.projectsmart.co.uk/white-papers/chaos-report.pdf

Tinytracker. (ei pvm). *An alternative to a Minimum Viable Product*. Noudettu osoitteesta Tinytracker Corporation: https://tinytracker.co/blog/minimum-viable-product-vs-minimum-lovable-product

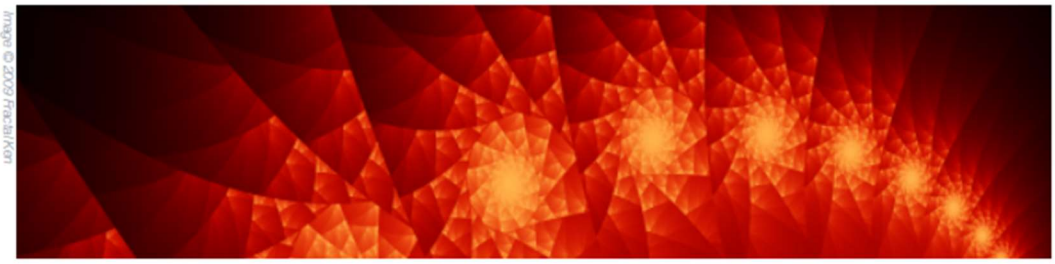
Wingfield, T. (2010). *Doing Kanban Wrong*. Noudettu osoitteesta www.infoq.com: https://www.infoq.com/articles/doing-kanban-wrong/

Wysocki, R. K. (2013). *Effective Project Management : Traditional, Agile, Extreme.* John Wiley & Sons, Incorporated. Noudettu osoitteesta http://ebookcentral.proquest.com/lib/samk/detail.action?docID=1575628

Xia, W. L. (2004). *rasping the complexity of IS development projects.* doi:10.1145/986213.986215

7 LIITTEET

LIITE 1. Agile-manifesti



AGILE MANIFESTO

We are uncovering better ways of developing software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Twelve Principles of Agile Software

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity – the art of maximizing the amount of work not done – is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

http://agilemanifesto.org