Tampereen yliopisto

Joel Rinkinen

SUURIEN OHJELMISTOPROJEKTIEN MUUTOSTEN HALLINTA

Kandidaatintyö
Johtamisen ja talouden tiedekunta
Tarkastaja: Jaakko Siltaloppi
Toukokuu 2023

TIIVISTELMÄ

Joel Rinkinen: Suurien ohjelmistoprojektien muutosten hallinta
Change management in large scale software projects
Kandidaatintutkielma
Tampereen yliopisto
Tuotantotalouden tutkinto-ohjelma
Toukokuu 2023

Ohjelmistoalan kasvaessa ohjelmistoprojektit laajenevat ja monimutkaistuvat. Pitkät ja suuret ohjelmistoprojektit ovat vaikeammin hallittavissa ja altistuvat monenlaisille muutoksille. Projektien aikana tapahtuvia muutoksia pitää pystyä hallitsemaan oikein, sillä ilman oikeaoppisia muutostenhallintatapoja saattavat projektit pahimmillaan jopa kaatua. Tämän työn tarkoituksena on tutkia minkälaisia muutoksia voi ohjelmistoprojekteissa tapahtua ja miten suurissa ohjelmistoprojekteissa pitäisi toteuttaa muutostenhallintaa.

Työ on toteutettu kirjallisuuskatsauksena ja jakautuu kahteen osaan. Teoriaosuudessa käsitellään ohjelmistoprojektien muutoslähteitä ja muutokset kategorisoidaan helpomman analyysin puolesta. Kirjallisuudessa on ohjelmakoodimuutoksia käsitelty runsaasti, mutta muita yksittäisiä muutoksia on tuotu esille vähänlaisesti. Tulososuudessa tutkitaan tunnistettuja muutostenhallintaprosesseja, itse ketterää kehitystä ja sen merkitystä ohjelmistoprojekteille sekä erilaisia ketterän kehityksen viitekehyksiä ja lopulta näiden yhdistelmiä ja sopivuutta suurille ohjelmistoprojekteille.

Ketterä ohjelmistokehitys (eng. Agile software development) on tunnistettu toimivaksi tavaksi toteuttaa ohjelmistoprojektien muutostenhallintaa. Muutostenhallintaprosessien hyvät puolet yhdistettiin esimerkilliseksi prosessiksi, jota tarjottiin vaihtoehdoksi muutostenhallinnalle. Ketterä kehitys tunnistettiin sopivan hyvin esiteltyyn muutostenhallintaprosessiin, jonka tarkoituksena on varmistaa toteutettavien muutosten onnistuminen ja tärkeys ohjelmistoprojekteille. Ketterä kehitys ei kuitenkaan toimi suurissa projekteissa täysin ongelmitta ja niiden toteutuksessa tunnistettiin lukuisia haasteita. Kolmea eri ketterää viitekehystä käsiteltiin tarkemmin ja niiden sopivuutta muutostenhallintaan ja suuriin ohjelmistoprojekteihin tutkittiin. Ketterien viitekehysten suurimmat haasteet olivat projektien hallinnollisella puolella, eikä suoraan ohjelmiston tuottamisessa tunnistettu ongelmia, vaikka toimintaa kasvatettaisiin.

Käsitellyissä viitekehyksissä ei ollut selkeästi parasta vaihtoehtoa suurille ohjelmistoprojekteille, vaan parhaaksi vaihtoehdoksi tunnistettiin yhdistelmä erilaisia ketteriä ja perinteisiä -toimintoja. Esimerkkinä tästä käsiteltiin LeSS-viitekehystä, jossa sopeutetaan Scrum toimintaa suurille projekteille. Ketterien viitekehysten hyödyntäminen varmistaisi muutostenhallinnan ja ohjelmiston tuottamisen toimivuuden ja perinteiset toiminnot mahdollistaisivat projektien koon kasvattamisen ilman että projektin hallitseminen muuttuisi liian raskaaksi ja vaikeaksi.

Avainsanat: ohjelmiston muutos, muutostenhallinta, ohjelmistoprojekti, ketterä ohjelmistoprojekti

ALKUSANAT

Oma mielenkiintoni ohjelmistotekniikkaa kohtaan on herännyt tekemieni töiden ja kouluni myötä. Yliopistossa suorittamani projektinhallintakurssin jälkeen tajusin haluavani yhdistää ohjelmistotekniikkaan projektinhallintaa, minkä pohjilta päädyin tämän työn aiheeseen. Työn tarkoituksena oli tutkia ohjelmistoprojektien muutoksia ja hallintatapoja näihin muutoksiin. Vaikka ohjelmistoprojektit ovat kokonaisuudessaan mielenkiintoisia, halusin tutkia aihetta enemmän projektien laajuuden kannalta, minkä takia päätin rajata projektin budjetin ja ajan käsittelyn työn ulkopuolelle.

Haluan kiittää ohjaajaani Jaakko Siltaloppia ja professori Tuomas Aholaa heidän ohjeistuksesta ja erinomaisista neuvoista työn ohella. Kiitokset kuuluvat myös ystävilleni ja perheelleni heidän antamasta ajasta ja hyvistä neuvoista.

Tampereella 7.5.2023

Joel Rinkinen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

XP	Extreme Programming
FDD	Feature Driven Development
LeSS	Large Scale Scrum

1. JOHDANTO

1.1 Työn tausta

Ohjelmistoalan kasvaessa ja yritysten toteuttamien ohjelmistoprojektien monimutkaistuessa niiden hallinta vaikeutuu ja riippuvuus ympäristöstä kasvaa. Ohjelmointiympäristö ei pysy vakiona ohjelmistoprojektin tuotannon aikana, vaan projektin on pystyttävä omaksumaan tapahtuvat muutokset osaksi toimintaansa. (Jayatilleke & Lai, 2013) Ohjelmistoprojektit eivät kuitenkaan yleensä halua toteuttaa tapahtuvia muutoksia, mutta koska kaikki muutokset eivät aina ole projektin johdon vaikutettavissa, on muutostenhallinta pakollista (Highsmith & Cockburn, 2001). Tapahtuvat muutokset voivat olla suuriakin, mikä aiheuttaa projektin kaatumisvaaran ilman oikeaoppisia muutoksien hallintatapoja. Haasteet ovat erityisesti näkyvillä suurissa projekteissa, joissa projektin osa-alueiden ja työntekijöiden suurempi määrä vaatii lisää projektin hallinnalta. Tässä työssä suurella ohjelmistoprojektilla tarkoitetaan projekteja, jotka sisältävät vähintään useita kymmeniä henkilöitä.

Suurin osa ohjelmistoprojektin aikana aiheutuvista riskeistä johtuu projektin ympäristön muutoksista. Tapahtuvat muutokset ympäristössä vaativat nopeaa toimintaa projektin johtajalta, koska projekti voi muuten kärsiä tappioita tai pahimmillaan kaatua kokonaan. (Debnath et al., 2006) Iso osa näistä tapahtuvista ulkopuolisista muutoksista liittyy asiakastarpeiden muuttumiseen ja päivittymiseen. Asiakastarpeiden muutokset voivat liittyä esimerkiksi väärin asetettuihin alkuvaatimuksiin tai ohjelmistolta vaadittavien ominaisuuksien muuttumiseen. (Anwer et al., 2019)

Vielä yleisesti ohjelmistotuotannossa käytössä oleva toimintamalli, vesiputousmalli (eng. waterfall model), on erityisesti muutosten hallinnan kannalta epäkäytännöllinen. Vesiputousmallia pidetään vanhanaikaisena ja kömpelönä, sillä mallia hyödyntävässä ohjelmistoprojektissa ei voida reaktiivisesti muokata toimintasuunnitelmaa kesken tuotannon. Kankeus vaikeuttaa erityisesti muutostenhallintaa. (Petersen et al., 2009)

Alawairdhi (2016) esittää ketterää kehitystä toimintatavaksi muutosten hallitsemiseksi ohjelmistotuotantoon. Ketterä kehitys (eng. agile software development) pyrkii korjaamaan ohjelmistojen tuotannon haasteita lyhytkestoisilla pyrähdyksillä (eng. sprint), joissa jokaista projektin aspektia käsitellään tasaisin ja mieluiten lyhyin väliajoin (Highsmith & Cockburn, 2001; “Principles behind the Agile Manifesto,” 2001). Ketterä

kehitys parantaisi erityisesti vesiputousmallia hyödyntävän organisaation toimintaa nopeuttamalla tuotannon aikana tapahtuvien muutosten käsittelemistä ja hallintaa.

Ketterä kehitys ei kuitenkaan ole kaiken korjaava yliluonnollinen ratkaisu ohjelmistoprojektien toteutukseen, sillä sen toteutuksesta on tunnistettu myös paljon haasteita (Aziz Butt et al., 2022). Vaikka Petersen et al. (2009) tunnistivat myös vesiputousmallin haasteelliseksi suurille organisaatioille, ei ketterä kehitys yksinään ole suurille organisaatioille helpompi vaihtoehto. Ketterän kehityksen omaksuminen onkin tunnistettu haasteelliseksi suurille organisaatioille (Bass, 2012; Aziz Butt et al., 2022). Ketterän kehityksen painotus henkilökohtaisiin ja usein järjestettäviin keskustelutilaisuuksiin vaatii projektin hallinnolta ylimääräisiä resursseja, mikä kertautuu suurissa ohjelmistoprojekteissa.

1.2 Työn tavoite

Tämän työn tarkoituksena on tutkia muutoksia ohjelmistoprojekteissa ja menetelmiä hallita ja johtaa näitä tapahtuvia muutoksia. Hankitun tiedon avulla pyritään parantamaan ymmärrystä erilaisista tapahtuvista muutoksista ja miten niitä pitäisi hallita. Tiedolla pyritään vastaamaan seuraaviin tutkimuskysymyksiin:

1. Minkälaisia muutoksia voi ohjelmistoprojektissa tapahtua ja mistä ne johtuvat?
2. Minkälaisia menetelmiä on suurien ohjelmistoprojektien muutosten hallintaan ja minkälaisia riskejä nämä aiheuttavat?

Tutkimuskysymysten vastausten avulla pystyttäisiin johtamaan ja hallitsemaan ohjelmistoprojektia entistä tehokkaammin, poistaen tai vähintään minimoiden ohjelmistotuotannon odottamattomia haasteita ja parantaen projektien lopullista laatua. Saaduista vastauksista pyritään myös tunnistamaan puutteita, jotta mahdollisia jatkokysymyksiä ja tutkimusaiheita voitaisiin tuoda esille.

1.3 Tutkimusmenetelmät ja työn kulku

Tämä työ toteutetaan kirjallisuuskatsauksena. Lähteitä etsittiin Tampereen yliopiston kirjaston hakupalvelusta Andorista, Web of Science -hakupalvelusta, sekä Scopus-tietokannasta. Lähteiden etsinnässä käytettyjä hakusanoja ja löydettyjen artikkelien määrät ovat esiteltynä taulukossa 1.

Taulukko 1: Artikkeleiden hakutulokset

HAKUKOHDE	KÄYTETYT HAKUSANAT	LÖYDETYT ARTIKKELIT	VALITUT ARTIKKELIT
SCOPUS	“software engineering” AND "software project" AND ("change management" OR "change leadership")	33	3
SCOPUS	“Agile development” AND ("change management" OR "change leadership")	38	1
ANDOR	“software engineering” AND "software project" AND ("change management" OR "change leadership")	17	1
ANDOR	“Agile development” AND ("change management" OR "change leadership")	15	0
WEB OF SCIENCE	“Agile development” AND ("change management" OR "change leadership")	21	1
WEB OF SCIENCE	“software engineering” AND "software project" AND ("change management" OR "change leadership")	8	0

Jatkokäsittelyyn valittiin artikkelit jotka käsittelivät muutostenhallintaprosesseja, muutoksia itsessään tai ketterän kehityksen toteuttamista muutostenhallinnan näkökulmasta. Työn aiheen kannalta sopivia artikkeleita oli kuitenkin vähän ja suurin osa hylätyistä artikkeleista käsittelivät liian tarkasti ohjelmakoodissa tapahtuvia muutoksia, jotka eivät esitellyssä tarkkuudessa kuulu aihealueelle. Valituissa artikkeleissa aiheet oli esiteltynä joko laajasti ohjelmistoalalle tai suoraan suurille projekteille, sillä myös pelkästään pienien projektien kannalta aihetta käsittelevät artikkelit hylättiin valikoimasta. Päällekkäisyyksiä hakukohteiden kesken valituissa artikkeleissa oli vain yksi ja loput työhön valituista artikkeleista saatiin valituista artikkeleista helmenkalastusmenetelmän avulla.

Työn rakenne on seuraavanlainen. Toisessa luvussa esitellään ohjelmistoprojektien muutosten lähteet ja kategoriat. Kolmannessa luvussa käsitellään muutostenhallintaprosesseja, ketterää kehitystä sekä muutamaa ketterää viitekehystä ja niiden haasteita suurissa ohjelmistoprojekteissa. Työn lopuksi neljäs luku koostaa työn yhteen ja vastaa aikaisemmin esitettyihin tutkimuskysymyksiin.

2. OHJELMISTOPROJEKTIN HALLINTA

2.1 Suuret Ohjelmistoprojektit

Ohjelmistoprojektit eroavat huomattavasti ominaisuuksiltaan, hallittavuudeltaan ja toimintatavoiltaan muista teollisista projekteista, kuten rakennusprojekteista. Ohjelmistoprojektissa projektin asiakkaalle tuotetaan ohjelmistoa annettujen vaatimusten mukaisesti. Asiakkaan vaatimusten ja alkukartoituksen pohjilta laaditaan suunnitelmat ohjelmistolle, jota aletaan tuottamaan. Ohjelmistoprojektin suurimpina saavutuksina voidaan pitää suunnitelman valmistumista, ohjelman valmistumista ja luovuttamista asiakkaalle, tosin projektit eivät välttämättä lopu vielä luovuttamiseen. Perinteiseen teollisuuteen verrattuna ohjelmistoprojektin konkreettinen esiintymä on vaikeammin hahmotettavissa, eikä se kuluta samalla tavalla fyysisiä resursseja. Ohjelmistoprojekteissa on projektin suuruudella myös paljon merkitystä. Suuria ohjelmistoprojekteja pitää käsitellä pienempiin verrattuna eritavalla, projektin suuruuden aiheuttaman monimutkaisuuden takia (Alqudah & Razali, 2016).

Ohjelmistoprojektin hallinta on tärkeää projektin onnistumisen vuoksi. Debnath et al. (2006) ovat tunnistaneet kaatumisriskin ohjelmistoprojektille, jonka muutoksia ei hallita ja ohjata oikeaoppisesti. Garousi et al. (2019) tunnistivat projektin hallinnan ja seuraamisen yhdeksi tärkeimmistä kriittisistä menestystekijöistä (eng. critical success factor). Kriittisellä menestystekijällä tarkoitetaan jollekin toiminnolle tai kokonaisuudelle, tässä työssä koko ohjelmistoprojektille, tärkeätä tekijää, jonka omistaminen tai osaaminen on pakollista toiminnon onnistumiselle. Ahimbisibwen et al. (2015) tunnistivat tutkimuksessaan tärkeimmäksi kriittiseksi menestystekijäksi projektin johdon antaman tuen, jota seurasi ohjelmoijien ja asiakkaiden osallistuminen projektiin. Garousin et al. (2019) eivät tunnistaneet kriittisiä menestystekijöitä samassa järjestyksessä kuin Ahimbisibwen et al. (2015). Garousin et al. (2019) tutkimuksessa tärkeimmäksi menestystekijäksi tuli monitorointi ja ohjaaminen, jota seurasi suunnittelu, projektitiimin taidot ja muutostenhallinta. Molemmissa tutkimuksissa oli melkein kaikki samat menestystekijät, mutta niiden tärkeysjärjestystä ei nähty samana. Ahimbisibwen et al. (2015) tunnistivat muutostenhallinnan vasta yhdeksänneksi tärkeimmäksi menestystekijäksi. Erot näiden tutkimusten järjestyksessä voi johtua niiden toteutusvuosista. Muutostenhallinta on muuttunut tärkeämmäksi lähivuosina, mikä näkyy Garousin et al. (2019) tutkimuksessa. Kokonaisuudessa tutkimukset eivät kuitenkaan eroa kovin paljoa. Molemmissa tunnistetaan tärkeiksi aiheiksi projektin visio, suunnittelu, projektinhallinta, muutostenhallinnan toimintatavat ja kommunikaatio. Kaikki nämä

aiheet ovat tärkeä osa projektinhallintaa ja muutostenhallintaa. (Ahimbisibwen et al., 2015; Garousi et al., 2019) Tämän pohjilta voidaan muutostenhallintaa pitää kriittisenä menestystekijänä nykypäivän ohjelmistoprojekteille.

2.2 Ohjelmistoprojektin muutokset

Tässä työssä ohjelmistoprojektin muutoksella tarkoitetaan projektin aikana tapahtuvaa muutosta, joka vaikuttaa projektin kulkuun, sen päämääriin, tavoitteisiin tai sisältöön. Muutoksia käsitellessä oletetaan myös, että ohjelmistoprojekti ei voi kasvattaa budjettiaan tai resurssejaan muutoksen käsittelemiseksi. Muutoksen voi aiheuttaa moni erilainen tekijä projektin aikana, eikä muutosten taustalla olevat tekijät ole aina projektin johdon vaikutettavissa (Highsmith & Cockburn, 2001). Ohjelmistoprojektissa tapahtuvat muutokset voivat olla joko sisäisiä, jotka voidaan jakaa vielä prosessin tai projektin perusteella, tai ulkoisia. Projektin sisäiset muutokset johtuvat projektin sisäisistä lähteistä ja projektin on mahdollista vaikuttaa näihin muutoksiin omalla toiminnallaan. (Highsmith & Cockburn, 2001; Debnath et al., 2006) Prosessikohtaiset muutokset liittyvät projektissa toteutettavaan prosessiin ja projektikohtaiset liittyvät projekin kokonaisuuteen (Debnath et al., 2006). Ulkoiset muutokset ovat projektin kannalta kriittisempiä. Ulkoiset muutokset vaativat tehokkaita muutostenhallintatapoja, sillä ne ovat projektin vaikutusvallan ulkopuolella eikä niihin voida vaikuttaa etukäteen. (Highsmith & Cockburn, 2001)

Muutoksia on kustannusten kannalta tärkeä hallita mahdollisimman nopeasti. Myöhäisemmässä vaiheessa toteutettavat ohjelmistomuutokset vaativat enemmän aikaa ja kustannuksia toteuttamiseksi. (Jones, 1996) Hyvät muutostenhallintatavat ovatkin projektille tärkeitä, sillä ne voivat helpottaa muutosten tunnistamista etukäteen. Aikaisemmalla muutosten tunnistamisella projekti saa lisäaikaa niihin reagoimiseen mahdollistaen paremman analyysin muutoksesta ja näin parantaen tehtävän päätöksen laatua ja laskien resurssikuluja (Rudic & Sobajic, 2012).

Ohjelmistoprojektin muutoksia voidaan käsitellä riskeinä tai mahdollisuuksina projektin toteutuksen ja onnistumisen puolesta. Näitä muutosriskejä ja -mahdollisuuksia voidaan jaotella vielä tarkemmin esimerkiksi henkilöstön, työkalujen, ympäristön ja teknologian perusteella. (Debnath et al., 2006) Tämä lisäkategorisointi muutoksille helpottaa niiden vaikutusalueen hahmottamista ja tarvittavien toimintojen tunnistamista. Kaikkien prosessien ja projektien läpikäynti voi olla suurissa projekteissa erittäin työlästä, joten yksittäisiin tärkeisiin muutoskategorioihin keskittymällä voidaan tehostaa riskienhallintaa. Koska muutoksia voi tulla monestakin eri suunnasta ohjelmistoprojektissa, aiheuttaa se lisäpaineita ja -töitä muutostenhallinnalle ja projektin kannalta valittaville toimintatavoille.

2.3 Muutoskategoriat

Ensimmäisenä muutoskategoriana on henkilöstö. Henkilöstöön liittyvät muutosriskit ovat esimerkiksi henkilöstövaje ja tiedon poistuminen organisaatiosta ohjelmoijien tai muiden työntekijöiden lähtiessä projektista. Yksittäisen työntekijän tehdessä muutoksia ohjelmistoon ilman riittävää dokumentaatiota jää tieto tästä muutoksesta vain kyseiselle henkilölle. Kadotetun tiedon aiheuttamia ongelmia on kuitenkin vaikea mitata, mikä vaikeuttaa henkilöstömuutosten hallintaa ja sen tärkeyden tunnistamista. (Nassif & Robillard, 2017) Työntekijän vaihtaessa työtehtävää organisaation sisällä ei häneen sidottua tietoa välttämättä menetetä, mutta tiedon uudelleenhankkiminen projektille vaatii kuitenkin töitä. Ohjelmoijan irtisanoutuessa kokonaan organisaatiosta voidaan kyseinen tieto olettaa täysin menetetyksi.

Henkilöstömuutokset voivat olla niin sisäisiä kuin ulkoisia haasteita. Highsmithin ja Cockburnin (2001) löydösten mukaan voidaan henkilöstövajetta pitää ulkoisena haasteena, sillä projektilla ei ole mahdollisuutta vaikuttaa saatavilla olevaan työvoimaan. Työntekijöiden irtisanoutuminen voidaan kuitenkin nähdä sisäisenä tai ulkoisena haasteena, sillä irtisanoutuminen voi riippua niin projektin toimintatavoista kuin ulkoisista muutoksista.

Toisena muutoskategoriana on työkalut. Vaikka ohjelmistoprojekteissa ei samalla tavalla käytetä fyysisiä työkaluja kuten esimerkiksi rakennusprojekteissa, ovat ne silti tärkeä osa ohjelmistoprojektia. Itse ohjelmointia varten tarvitsevat työntekijät tietokoneen tai jonkin muun vastaavan laitteen. Työkalumuutokset ovat yleensä ulkoisia projektin kannalta. Ellei projekti itse tuota omia työkalujaan, ei niihin silloin voida vaikuttaa. Esimerkkinä suuresta työkalumuutoksesta ohjelmistoalalla on viime vuosina tapahtunut tietokonesirupula (Casper et al., 2021). Sen takia on tietokoneiden ja niiden osien hinnat kasvaneet ja saatavuus huonontunut, minkä takia on tarvittavien tietokoneiden saaminen vaikeutunut (Casper et al., 2021). Ongelma on ollut kuluttajamarkkinoilla hyvin esillä, mutta käsitellyssä kirjallisuudessa ei tätä ole muutosriskinä juurikaan huomioitu, vaikka työkalut on tunnistettu aikaisemmin muutoslähteeksi (Debnath et al., 2006). Sirupula saattaakin olla ohjelmistoalalle uusi muutoshaste, koska sitä ei kirjallisuudessa olla käsitelty.

Ympäristö on kolmas muutoskategoria ja se on muita kategorioita huonommin rajattu. Ympäristömuutoksilla tarkoitetaan yksinomaan projektin ulkopuolisia muutoksia, joihin projekti ei pysty toiminnallaan vaikuttamaan (Debnath et al., 2006). Nämä muutokset voivat mennä sisällöllisesti hieman muiden muutoskategorioiden päälle, mutta ovat huomattavasti vaikeammin hallittavissa. Esimerkiksi sirupulaa voitaisiin pitää myös

ympäristömuutoksena ja sen jakaminen on työkalujen ja ympäristön välillä haasteellista. Toisaalta esimerkiksi erilaiset lakimuutokset on helpointa jaotella ympäristömuutoksiksi. Ne eivät sovi muiden kategorioiden kuvauksiin, mutta voivat silti olla hyvinkin kriittisiä projekin onnistumisen puolesta.

Neljäntenä muutoskategoriana ohjelmistoprojekteille on teknologia ja sen aiheuttamat muutokset. Uusien teknologioiden syntyessä ja vanhojen muuttuessa, saattavat työnalla olevat projektit joutua muuttamaan toimintaansa hyödyntääkseen uusia teknologioita tai muokkautuakseen vanhojen teknologioiden muutoksiin. Teknologiamuutoksia voidaan työkalujen mukaan pitää ulkoisina muutoksina, jos organisaatio ei tuota itse käyttämiään teknologioita. Ohjelmistotekniikan opettamisessa onkin tunnistettu jatkuvaa tarvetta koulutuksen muokkaamiselle ja uudelleenopettamiselle, aiheen jatkuvien ja nopeiden muutosten takia (Rosca et al., 2003). Vaikka tutkimus onkin tehty koulutuslaitokselle, tarvitsevat myös työelämässä olevan henkilöt koulutusta, jotta pysyvät muutoksissa mukana. Työntekijöiden syvällisempi uudelleenkoulutus kesken projektia on kuitenkin harvinaisempi ongelma. Ohjelmistoprojektin pitäisi kestää useita vuosia, jotta työntekijöiden uudelleenkouluttaminen muuttuisi varteenotettavaksi haasteeksi yksittäisen projektin kannalta. Uudelleenkoulutus on koko organisaation kannalta suuri haaste, mutta ei juurikaan yksittäisten projektien kodalla.

Suurimmaksi yksittäiseksi muutoslähteeksi ohjelmistoprojekteissa on tunnistettu virheellisesti asetetut tai ymmärretyt alkuvaatimukset ja muuttuvat asiakastarpeet projektin tuotannon aikana (Rudic & Sobajic, 2012; Anwer et al., 2019). Mitä myöhemmässä vaiheessa vaatimukset muuttuvat, sitä useampaan alueeseen ohjelmistossa muutokset vaikuttavat, mikä lisää muutosten kompleksisuutta ja vaatimaa työtä. Yksinkertaisten ohjelmointimuutosten tekeminen siis hidastuu ja samalla kallistuu eksponentiaalisesti ohjelmiston koon kasvaessa (Jones, 1996). Asiakastarpeet voivat muuttua esimerkiksi teknologiamuutoksien takia, jolloin niitä voidaan pitää ulkoisina muutoksina. Toisaalta asiakastarpeet tai alkuvaatimukset voivat muuttua puutteellisten prosessien takia, minkä pohjilta niitä voitaisiin pitää sisäisinä haasteina. Alkuvaatimukset ja asiakastarpeet ovatkin yksi haasteellisimmista muutoksista, sillä ne voivat johtua hyvinkin monesta eri lähteestä tehden niiden hallinnan erittäin haasteelliseksi (Rudic & Sobajic, 2012; Anwer et al., 2019).

3. MUUTOSTEN HALLINNOINTITAVAT

3.1 Muutostenhallintaprosessit

Jos budjettia tai resursseja ei muutosten käsittelemiseksi voida nostaa, on muutostenhallintaprosessi ainoa tapa käsitellä muutokset. Oikeaoppisella ohjelmistoprojektin muutostenhallinnalla varmistetaan, että ohjelmistoprojektin aikana tapahtuvat muutokset eivät kaada projektia ja että niiden aiheuttamat vaikutukset olisivat mahdollisimman pienet tai jopa positiiviset projektille (Debnath et al., 2006). Muutostenhallinnalle on kuitenkin monia erilaisia tapoja, eikä kaikki välttämättä sovi kaikkiin projekteihin yhtä hyvin. Suurissa ohjelmistoprojekteissa on erityisen tärkeää, että käytettävä muutostenhallinta on mahdollisimman sopiva, sillä suurissa projekteissa pienetkin hukat kertautuvat helposti. (Aitken & Ilango, 2013)

Muutostenhallintaprosessille on esitelty useita vaihtoehtoja. Debnathin et al. (2006) esittämä muunnelma alkaa muutoksen tärkeyden esittelemisellä. Tärkeyden perusteella pyritään muokkaamaan toimintaa prosessin ohjaamalle mallille, minkä jälkeen toiminta jäädytetään tälle uudelle tasolle. Heidän esittelemä prosessi perustuu halutun toimintamallin esittelemiseen ja kiireen tunteen aiheuttamiseen, jolloin projektissa tai organisaatioissa siirryttäisiin uusiin tapoihin mahdollisimman nopeasti, eikä jäätäisi epäröimään liikaa. Viimeinen askel toiminnan uudelleenjäädyttämiseksi on myös tärkeä, jotta toiminta ei lähde muuttumaan kuitenkaan sitä haluamatta.

Ali ja Lai (2016) esittelevät hieman tarkemman prosessin muutostenhallinnalle. Heidän prosessissa muutostenhallinta lähtee muutoksen ymmärtämisellä. Heidän mukaan ennen muutosten syvällisempää käsittelemistä pitää tietää mitä muutos todella tarkoittaa ja mitä se sisältää. Vasta ymmärtämisen jälkeen lähdetään analysoimaan muutoksen vaikutuksia ja sen vaatimia resursseja. Analysoinnin jälkeen tehdään päätös muutoksen toteuttamisesta tai hylkäämisestä tunnistettujen vaikutusten perusteella. Jayatilleken ja Lain (2013) esittelemä muutostenhallintaprosessi on samantyylinen kuin Alin ja Lain (2016). Heidän mukaansa muutoksen selittäminen ja sen ymmärtäminen ovat tärkeitä, mutta he myös lisäävät tarpeen käsitellä muutospyynnön kieltä. Koska muutokset tulevat yleensä organisaation ympäristöstä tai sen rajapinnasta, on muutos yleensä ensimmäisenä liiketoimintapuolen käsittelyssä. Liiketoimintapuoli ei välttämättä ymmärrä teknisiä asioita samalla tavalla kuin ohjelmoijat. Tämän takia on muutoksen kielellinen esittely tärkeää, jotta sen sisältö ja ymmärrettävyys eivät muutu eri osapuolien välillä. (Jayatilleke & Lai, 2013)

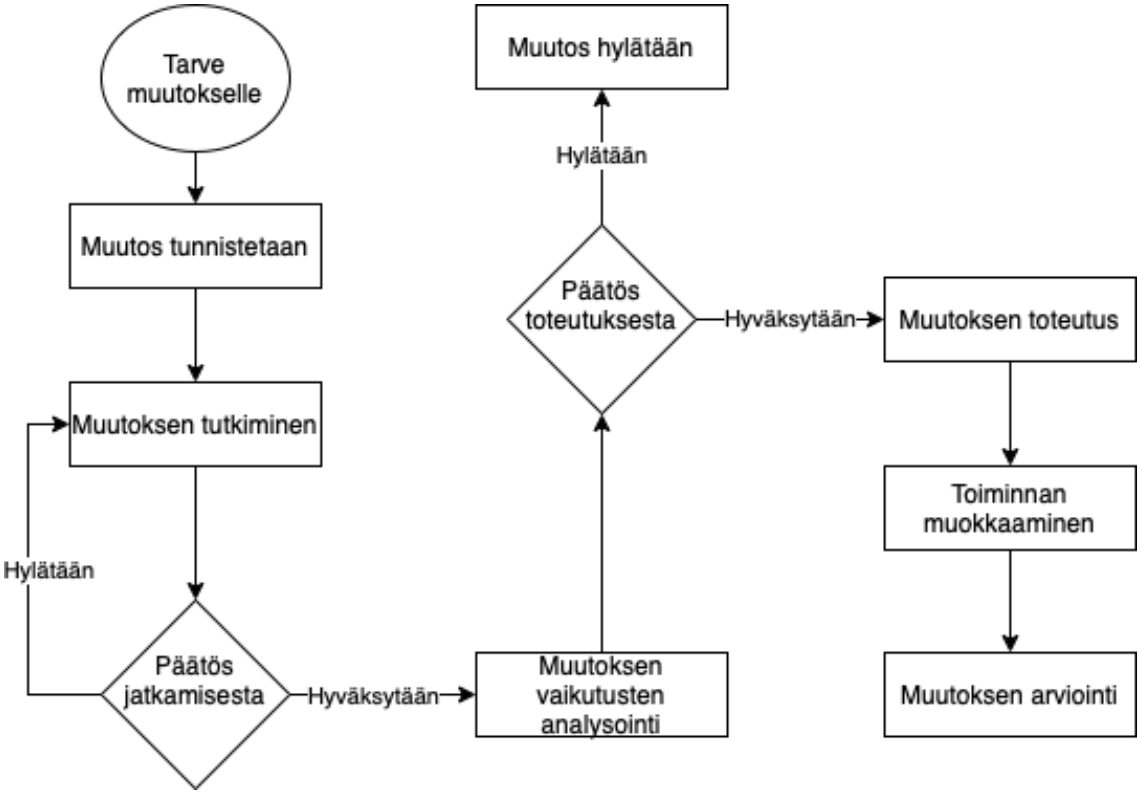
Debnathin et al. (2006), Jayatilleken ja Lain (2013) ja Alin ja Lain (2016) esittelemät muutostenhallintaprosessit ovat paikoittain hyvinkin samanlaisia ja päällekkäisiä, vaikka koskevat hieman eri vaiheita kyseisestä prosessista. Alin ja Lain (2016) ja Jayatilleken ja Lain (2013) esittelemä prosessi koskee eniten itse muutostenhallintaprosessia, kun Debnath et al. (2006) käsittelevät enemmän muutoksen käyttöönottoa. Drvodelić Cvitak ja Car (2010) esittävät oman prosessinsa ohjelmistoprojektin muutosten käsittelemiselle. Heidän esittelemänsä prosessi on hieman yksityiskohtaisempi muihin käsitellyihin prosesseihin verrattuna ja sisältää osia kaikista käsitellyistä prosesseista. Prosessi lähtee muutostarpeen tunnistamisesta, josta lähdetään tutkimaan muutoksen tarkoitusta. Alkutietojen kasaamisen jälkeen aloitetaan analysoimaan muutoksen vaikutusta ja lopullista tarpeellisuutta. Päätöksen perusteella muutos joko hylätään ja ilmoitetaan muutoksen pyytäjälle tai muutos hyväksytään ja siirretään ohjelmoijalle työstettäväksi. (Drvodelić Cvitak & Car, 2010) Drvodelić Cvitakin ja Carin (2010) esittelemä prosessi on lähimpänä Jayatilleken ja Lain (2013) esittelemää prosessia, mutta sisältää erityisesti muutoksen alkuperäisen syyn tutkimisen ja lopulta muutoksen siirtämisen ohjelmoijalle ja toteutuksen onnistumisen arvioinnin.

Drvodelić Cvitakin ja Carin (2010) esittelemästä prosessista puuttuu hieman painoarvoa muutoksen kiireellisyyden tuottamiselle. Vaikka heidän prosessissaan muutoksen alkuperäisen syyn tutkimisen tarkoitus on varmaan tarkoitettu kiireellisyyden tunnistamiseen ja luomiseen, voisi se kaivata kiireellisyyden tuottamista hieman enemmän. Muiden esittelemistä prosesseista puuttuu ohjeistusta muutoksen työstämisen aloittamisesta ja erityisesti lopullisen vaikutuksen arvioinnista. Muutostenhallintaprosessin onnistumisen puolesta tarvitaankin osia kaikista aikaisemmin käsitellyistä prosesseista. Hyvän muutosten ymmärtämisen ja analysoinnin avulla varmistetaan, ettei toteuteta huonoja ja mahdollisesti haitallisia muutoksia. Analyysin ohella tarvitaan myös tehokas tapa toteuttaa muutokset, jotta uusien muutosten kanssa epäröiminen ei kuluttaisi turhia resursseja.

Muutostenhallinta ei kuitenkaan onnistu vain näitä prosesseja suoraan seuraamalla. Anwer et al. (2019) toteuttamassa tutkimuksessa tunnistettiin muutostenhallinan prosessista vaikutusten analysointi yleisimmäksi haasteeksi kustannusten arvioinnin edelle. Muutokset voivat koskea useitakin alueita ja liittyä moniin eri kohteisiin, minkä takia sen haasteellisuus ei tule yllätyksenä. Kustannusten arvioinnin haasteellisuus liittyy varmaan samoihin syihin kuin vaikutusten arvioinnin. Kustannuksia voi tulla monista eri suunnista, joka vaikeuttaa niiden arviointia ennen muutoksen toteuttamista.

Anwer et al. (2019) tunnistivat myös kommunikaation haasteeksi erityisesti globaalille ohjelmistontuotannolle. Kommunikaatio muuttuu jo pelkästään suuren työntekijämäärän

takia suurissa projekteissa vaikeammaksi. Globaalissa toiminnassa haasteet kertautuvat maiden eri aikavyöhykkeiden, kulttuurillisten ja kielellisten haasteiden esiintyessä. (Anwer et al., 2019)



Kuva 1: Muutostenhallintaprosessikaavio (mukaillen Drvodelić Cvitak & Car, 2010; Ali & Lai, 2016)

Yhdistämällä eri muutostenhallintaprosessien vahvuudet saadaan tehtyä kuvassa 1 esitelty muutostenhallintaprosessikaavio. Ensimmäinen prosessin tärkeä ominaisuus on nopea ja tehokas muutosten tunnistaminen ja käyttöönotto (Debnath et al., 2006), mikä näkyy kaavassa muutosten tunnistamisessa ja toiminnan muokkaamisella. Toteuttamalla nämä vaiheet tehokkaasti voi projekti varmistaa muutoksen käsittelemisen tapahtuvan nopeasti. Toiseksi tärkeäksi ominaisuudeksi tunnistettiin muutoksen tutkiminen ja sen vaikutusten analysointi (Drvodelić Cvitak & Car, 2010; Jayatilleke & Lai, 2013; Ali & Lai 2016). Muutoksen tunnistamisen jälkeen muutosta tutkitaan, jotta saadaan yksinkertainen kuva toteutettavasta muutoksesta. Muutos voidaan tutkia uudelleen tässä kohdassa tai jopa hylätä, jos se ei vaikuta projektille tärkeältä. Jos muutos tunnistetaan tärkeäksi, analysoidaan sen vaikutuksia tarkemmin. Toisen analysoinnin jälkeen tehdään lopullinen päätös muutoksen toteuttamisesta, minkä jälkeen muutos joko hylätään lopullisesti tai siirretään toteutukseen. Viimeinen tärkeä vaihe on vaikutusten analysointi (Drvodelić Cvitak & Car, 2010). Vaikutusten analysoinnin avulla voidaan mahdolliset prosessin aikana tapahtuneet virheelliset

päätöksen vielä korjata ilman liian suuria jälkivaikutuksia. Muutoksen jatkuva analysointi voidaankin nähdä tämän muutostenhallintaprosessin suurimpana vahvuutena. Tehtävien päätösten ei pitäisi mennä väärin ajankohtaisen tiedon avulla ja muutoksen priorisointia on helppo muuttaa analyysien pohjilta jopa kesken prosessin.

3.2 Ketterä kehitys ohjelmistoprojekteissa

Vesiputousmalli on vielä yleisesti käytetty toimintamalli ohjelmistotuotannossa, mutta sen haasteet alkavat vaikeuttamaan sen tehokasta käyttöä jatkuvasti monimutkaistuvissa ohjelmistoprojekteissa. Vesiputousmallissa ohjelmistoa tuotetaan toisistaan irrallisissa vaiheissa ja jokaisessa vaiheessa keskitytään eri kohtiin tuotannossa. Vaiheet käsittelevät pieniä kokonaisuuksia ohjelmistosta, eikä eri vaiheiden välillä ole yleensä päällekkäisyyksiä. Mallissa nimensä mukaan ”pudotaan” ohjelmistoprojektin eri toimintojen välillä eikä toiminnoissa voida palata takaisin aikaisemmin toteutettuihin. (Petersen et al., 2009) Tämä aiheuttaa vaikeuksia tuotannossa, sillä jälkikäteen tunnistettuihin puutteisiin tai vaatimuksiin ei voida vaikuttaa kesken tuotannon.

Vesiputousmallissa muutosten ja mahdollisten virheiden tunnistaminen on toimintamallin takia vaikeampaa (Petersen et al., 2009). Mallin suurimpina haasteina on siis sen kankeus ja kykenemättömyys vaikuttaa ja tunnistaa tapahtuvia muutoksia ja virheitä reaaliajassa. Vesiputousmallissa ohjelma joudutaan tuottamaan kerralla kokonaan oikein, koska mahdollisiin kesken tuotannon esiintyviin muutoksiin ei voida vaikuttaa. Muutokselle ei voida tehdä mitään, jos silloin keskeneräinen toiminto ei mahdollista kyseisen muutoksen kohteen muokkaamista. Esimerkiksi vaatimusmuutokset ovat periaatteessa erittäin vaikeita vesiputousmallille. Jos vaatimus muuttuu kyseisen vaatimuksen ohjelmointivaiheen valmistuttua, ei esiintyvää muutosta ole mahdollista toteuttaa ennen vesiputouksen loppuun viemistä.

Ketterää kehitystä on esitelty ratkaisuksi kankeille ohjelmistotuotantotavoille, kuten vesiputousmallille (Alawairdhi, 2016). Ketterä kehitys keskittyy notkeuttamaan kankeaa ohjelmistontuotantoa mahdollistamalla muutosten käsittelemistä ja toteuttamista jokaisessa projektin vaiheessa ja parantamalla projektin osapuolien kommunikaatiota kasvokkain usein pidettävillä tapaamisilla (“Principles behind the Agile Manifesto,” 2001). Ketterän kehityksen toimintoja hyödyntävä projekti voi muokkautua paremmin tulevien muutosten ympärille, helpottaen niiden käsittelyä ja samalla laskien näiden muutosten käsittelemiseen tarvittavien resurssien määrää (Highsmith & Cockburn, 2001).

Ketterä kehitys korjaa tuotantotapoja muuttamalla toimintaa lyhyiksi toistuviksi kokonaisuuksiksi. Ketterässä kehityksessä jokainen lyhyt pyrähdys sisältää osia jokaisesta ohjelmistotuotannon osasta, jolloin mahdollisiin muutoksiin jokaisessa projektin osuudessa voidaan vaikuttaa. Ketterässä kehityksessä ohjelmistoa ei tarvitse tuottaa täysin oikein heti kättelystä, vaan pyrähdysten aikana voidaan vaikuttaa eri vaiheiden toteutukseen ja jopa muokata projektin päämääriä. Yhtenä tärkeänä vaiheena ketterässä kehityksessä pidetään myös jatkuvaa toiminnan arviointia. Ohjelmistoprojektin eri osakkaat arvioivat jatkuvasti projektin etenemistä, tehtyjä päätöksiä ja lopputuloksia. Arvioinnin avulla voidaan tunnistaa mahdollisia ongelmia toiminnassa ja korjata niitä kesken projektin. (Highsmith & Cockburn, 2001; “Principles behind the Agile Manifesto,” 2001)

Yksittäinen ketterän kehityksen pyrähdys sisältäisi siis kaikkea ohjelman suunnittelemisesta ja kirjoittamisesta aina ohjelman testaukseen ja muuhun ympäristöön integroimiseen sekä toiminnan ja tulosten arvioimiseen. Pyrähdykset ovat lyhyitä, jotta jokaisen vaiheen ja toiminnon välillä ei kestäisi liian pitkään. Agile manifestin (“Principles behind the Agile Manifesto,” 2001) mukaan pyrähdysten pituus olisi hyvä pitää lyhyenä mieluiten muutamassa viikossa. Lyhyet pyrähdykset ovat helpompi sisällyttää kaikkiin projekteihin, projektin pituudesta riippumatta, ja lyhyemmät pyrähdykset mahdollistavat nopeampaa reagointia muutoksiin ja muuhun toimintaan.

Aitken ja Ilango (2013) ovat myös tunnistaneet ketterälle kehitykselle halun pitää itse ohjelman kirjoittaminen mahdollisimman yksinkertaisena. Tämä vähentää tarvittavaa ohjelman uudelleenkirjoittamista ja samalla poistaa turhaa työtä ohjelmoinnilta. Ketterän kehityksen lyhyempien toimintojen avulla on helpompi tehdä pieniä muutoksia useasti, minkä avulla tapahtuvat muutokset eivät voi muuttaa suurta osaa ohjelmistosta vialliseksi kerralla. Ketterässä kehityksessä keskitytään myös vähemmän perinteisten sopimusten tekemiseen ja niiden täydelliseen noudattamiseen ja aika käytetään mieluummin projektin osien yhteistyöhön ja jatkuvaan toiminnan parantamiseen. (Aitken & Ilango, 2013; Aziz Butt et al., 2022)

3.3 Ketterän kehityksen viitekehykset ohjelmistoprojekteissa

Ohjelmistotuotannolle on olemassa useita erilaisia viitekehyksiä ja lähestymistapoja. Muutostenhallinnan näkökulmasta on jokaisessa viitekehyksessä omat puolensa, eikä kaikki sopeudu samoihin ympäristöihin. (Aitken & Ilango, 2013) Viitekehyksen valinta onkin suurille projekteille haasteellista (Alqudah & Razali, 2016). Toteutettavan viitekehyksen on mahdollistettava tehokas ohjelmistontuotanto sekä muutosten- ja

projektinhallinta. Projektinhallinta ja itse ohjelmistotuotanto riippuvat kuitenkin eri asioista, minkä takia niiden kohdalla saatetaan joutua tekemään kompromisseja.

Muutostenhallintaprosesseja on vaikeampi yhdistää perinteisiin viitekehyksiin niiden kankeuden ja toimintamallien takia, mutta ne sopivat hyvin ketteriin viitekehyksiin. Ketterien viitekehysten viikottaiset palaverit helpottavat Anwer et al. (2019) tunnistamia kommunikaatihaasteita, sillä usein järjestettävät palaverit helpottavat jatkuvaa kommunikaatiota ja mahdollistavat ongelmakohtien nopeampaa käsittelemistä. Prosesseissa toteutettavien muutosten kriittisyyttä on myös helpompi käsitellä ja muokata, kun muutoksen kohteena olevat henkilöt ovat mukana keskusteluissa. Drvodelić Cvitakin ja Carin (2010) muutostenhallintaprosessi sopii erityisesti Scrum-viitekehyksen mukaan toimivalle tiimille. Moninainen tiimi Scrumissa helpottaa muutosten vaikutusten tutkimista ja käsittelemistä useista eri näkökulmista.

Ketterä kehitys on toimintatapa ja monet erilaiset viitekehykset hyödyntävän tällaista toimintaa. Kuuluisimpia käytössä olevia ketteriä viitekehyksiä ovat Scrum ja XP (eng. extreme programming), mutta ei niin kuuluisia viitekehyksiä on lukuisia, mistä hyvänä esimerkkinä on FDD (eng. feature driven development). (Aitken & Ilango, 2013; Aziz Butt et al., 2022) FDD valittiin sen eroista Scrumiin ja XP:hen, sekä sen kuitenkin suhteellisen yleisyyden puolesta. Ketterät toimintatavat ovat yleensä hyvin toimivia ohjelmistontuotannon ja yleensä muutostenhallinnan näkökulmasta, mutta saattavat jäädä jälkeen projektinhallinnassa (Alqudah & Razali, 2016).

3.3.1 Scrum

Scrumin toiminta perustuu ketterän kehityksen mukaisiin lyhyihin iteratiivisiin pyrähdyksiin, joiden aikana ohjelmisto tuotetaan. Scrumin erikoisuus tulee siinä käytettävän tiimin kokoonpanosta ja toiminnasta. Tiimi Scrumissa sisältää henkilöitä jokaisesta projektin osa-alueesta, jopa projektin asiakkaalta. (Aitken & Ilango, 2013) Tämän avulla yksittäinen Scrumin tiimi voi käsitellä kaikkia tapahtuvia muutoksia, riippumatta muutoksen kohteesta tai lähteestä.

Scrum on iteratiivinen toiminnaltaan. Vaikka yksittäisen pyrähdyksen pituus ja sisältö on kiinteä, valitaan pyrähdyksessä tehtävät työt muuttuvasta työpinosta. (Ahimbisibwen et al., 2015) Scrum sopeutuu hyvin muutostenhallintaan iteratiivisuuden ja moninaisuuden avulla. Moninaisen tiimin avulla Scrumissa pystytään reagoimaan helpommin monipuolisiin muutoksiin ja iteratiivisuus varmistaa että viikottaisesti pidettävissä tapaamisissa tärkeiksi tunnistetut tehtävät saadaan aina mahdollisimman nopeasti työn alle. Asiakastarpeiden ja -vaatimusten muutokset ovat myös helpommin käsiteltävissä, sillä Scrumissa yleensä tehdään läheisempää yhteistyötä asiakkaan kanssa.

Scrumin painotus tiimityöskentelylle ja jatkuvalle kommunikaatiolle aiheuttaa kuitenkin vaikeuksia toiminnalle mahdollisten henkilöstömuutosten tapahtuessa (Highsmith & Cockburn, 2001). Koska tiimi sisältää paljon eri työntekijöitä eri lähtöpaikoista, voi yksittäisen ihmisen irtautuminen tiimistä aiheuttaa suuriakin ongelmia kyseisen osa-alueen käsittelemisessä ja hoitamisessa. Työntekijän irtautuessa Scrumin tiimistä on myös tiedon ylläpito vaarassa. Koska Scrumissa seurataan ketterän kehityksen ohjeita yleensä tarkasti, on painoarvo viikoittaisilla henkilökohtaisilla tapaamisilla ja toimivalla ohjelmalla, laajan dokumentoinnin sijasta (“Principles behind the Agile Manifesto,” 2001; Aitken & Ilango, 2013). Dokumentoinnin vähentyminen Scrumissa mahdollistaa kyseisen ajan käyttämistä muihin tarkoituksiin, mutta se samalla vaarantaa suuremman määrän tietoa, koska sitä ei kirjoiteta tarkasti työntekijöiden saataville.

3.3.2 XP

XP-viitekehys ohjaa projektitiimiä toimimaan ohjelmistolähtöisesti. XP-tiimissä ohjelmiston omistajuus ei osu vain projektia johtavalla henkilölle, vaan koko tiimi ainakin jossain määrin ”omistaa” ohjelman, eli ohjelman toimivuus ja oikeellisuus on jokaisen vastuulla. XP sisältää paljon erilaisia pieniä tehtäviä ohjelman tuottamiselle, jotta korkeille asetettuihin laatuvaatimuksiin yllettäisiin. XP tuotannossa painotetaan ohjelmistoa kaiken muun yli ja ohjelmakoodia käytetään suurena osana dokumentointia. (Bass, 2012; Aitken & Ilango, 2013; Aziz Butt et al., 2022) Projektinhallinnollisesta näkökulmasta XP ei anna mitään erillisiä toimintaohjeita ja sen hallitseminen muuttuu toiminnan kasvaessa haasteellisemmaksi (Aziz Butt et al., 2022). Yleisenä hallinnollisena tapana käytetään kuitenkin Scrum-viitekehyksen rooleja (Bass, 2012).

XP on ohjelmointipainotteisuuden avulla hyvä toimintamalli ohjelmaan kohdistuvien muutosten puolesta (Bass, 2012; Aitken & Ilango, 2013; Aziz Butt et al., 2022). Kun tehokasta ohjelmaa pystytään tuottamaan nopeasti, on myös siihen silloin helpompi tehdä muutoksia. XP:n ohjelmointipainotteisuus tuottaa kuitenkin haasteita työntekijämuutosten kanssa. Koska toiminta vaatii korkean tason ohjelmoijia, tulee työntekijöiden saaminen haasteelliseksi sillä osaavia työntekijöitä ei aina ole tarpeeksi saatavilla.

3.3.3 FDD

FDD viitekehyksen toiminnassa pyritään alkuun muodostamaan korkeatasoinen kuvaus haluttavasta ohjelmistosta, minkä jälkeen ohjelma jaetaan yksittäisiksi ominaisuuksiksi. Ohjelman ominaisuudet voidaan siten jakaa erillisille tiimeille suunniteltaviksi ja työstettäviksi. FDD:ssä painotetaan ohjelmiston pieniä kokonaisuuksia ja tarkemman

suunnitelman avulla toiminnan ohjaaminen muuttuu järjestelmällisemmäksi. (Aitken & Ilango, 2013)

FDD sopii ohjelman tarkan ja korkeatasoisen kuvauksen avulla suuremmille ja monimutkaisille ohjelmistoille, sillä ohjelma ja sen toteuttaminen on helppo jakaa eteenpäin tiimeille. FDD:ssä ei kuitenkaan ole kovin tarkkoja ohjeita muulle toiminnalle. Ohjelmistoa tuotetaan iteratiivisesti ja muuten ketterän kehityksen toimintamallia noudatetaan, mutta tarkkoja ohjeita ei ole. Epätarkkojen ohjeiden takia FDD:ssä voi toiminnan ohjaaminen vaikeutua ja ohjelman suunnitelmasta vastuussa olevan henkilön työmäärä kasvaa kohtuuttomasti. (Aziz Butt et al., 2022)

3.4 Ketterän kehityksen haasteet suurissa ohjelmistoprojekteissa

Ketterän kehityksen toimintatavat nähdään hyödyllisinä, mutta vaikeasti yhdistettävinä suuriin ohjelmistoprojekteihin. Ketterää kehitystä hyödyntävät viitekehykset tuovat paljon hyötyjä muutostenhallinnan ja ohjelmistontuotannon puolesta, mutta jäävät paikoittain jälkeen hallinnollisissa osuuksissa. (Dyba & Dingsoyr, 2009; Aziz Butt et al., 2022) Suurien ohjelmistoprojektien kannalta on hallinnolliset osuudet tärkeitä, sillä suurta työntekijämäärää on vaikea ohjata ja hallita ilman tehokkaita hallintamenetelmiä. Käsitellyistä viitekehyksistä Scrumilla ja XP:llä on eniten haasteita toiminnan skaalan kasvattamisessa, mutta FDD on hieman paremmin skaalautuva (Alqudah & Razali, 2016).

Scrumin ja XP:n skaalaaminen suuriin projekteihin on haasteellista tiimien koon ja toimintatapojen takia. Molemmissa on suositeltavaa pitää yksittäisen tiimin koko alle kymmenessä ja toiminta perustuu erityisesti Scrumissa enemmän itseohjautuvaan toimintaan. Scrumissa tiimi ei myöskään aktiivisesti työskentele ulkopuolisten kanssa, sillä tiimi sisältää jo kaikki projektin toteuttamisen kannalta tarvittavat henkilöt. (Aitken & Ilango, 2013; Alqudah & Razali, 2016; Aziz Butt et al., 2022) Tämä järjestely on haasteellista suurissa projekteissa. Ei ole mahdollista tehdä tarpeeksi isoa yksittäistä Scrum-tiimiä, joka pystyisi tuottamaan koko projektin itse.

Scrumista on toteutettu suurille projekteille sopivampi viitekehys LeSS (eng. Large Scale Scrum). Se toteuttaa Scrumin toimintatapoja pienissä tiimeissä, mutta muokkaa viikoittaiset suunnittelutapahtumat sisältämään henkilöitä jokaisesta projektin tiimistä. Tiimien välisissä pyrähdystapaamisissa ei kuitenkaan ole kaikkia tiimejä kokonaan, vaan jokaisesta tiimistä on pari henkilöä mukana. (Alqudah & Razali, 2016) Tällä muutoksella menetetään hieman ajatuksia ja näkökulmia tapaamisista, sillä kaikki eivät pääse osallistumaan. Toisaalta jos kaikki tiimit osallistuisivat tapaamisiin kokonaan, ei niin

suuren tapaamisen järjestäminen olisi helppoa tai käytännöllistä. Tämän yhteisen tapaamisen avulla saadaan Scrum toimimaan suurissa projekteissa, mutta sen tehokkuus kärsii palavereiden takia. Koska kaikki ihmiset eivät ole jokaisessa tapaamisessa mukana, joudutaan tietoa siirtämään toistamiseen tapaamisten jälkeen työntekijöiden kesken. Tiimien kanssa yhteistyötä tekevä asiakkaan yhteishenkilö on myös yleensä yhteinen tiimien välillä (Alqudah & Razali, 2016). Tämä lisää asiakkaan yhteyshenkilöltä vaadittavaa työpanosta, mikä laskee hänen ja yksittäisen tiimin kanssa tehdyn yhteistyön määrää.

XP:n kohdalla ei ole yhtä selkeää tapaa lähteä toiminnan skaalaa kasvattamaan. XP on toiminnaltaan raskas resurssien ja työntekijöiden kannalta, sillä se vaatii paljon korkeatasoisia ohjelmoijia (Aziz Butt et al., 2022). XP:n teknillisyyden ja ohjelmointi painotuksen takia se on työläs ja resurssi-intensiivistä yhdistää suuriin projekteihin. Jayatilleken ja Lain (2013) tunnistamat kielelliset haasteet ohjelmoijien ja organisaation liiketoimintapuolen välillä tuottavat haasteita XP:n skaalaamisessa. Projektin hallinnolliset osat eivät välttämättä ymmärrä yhtä teknisiä asioita kuin mitä XP:n toiminta tuottaa, mistä johtuen tiimien ja hallinnon rajapinnoissa joudutaan tehdä lisätöitä ymmärryksen varmistamiseksi. Alqudah ja Razali (2016) tunnistivat kuitenkin tutkimuksessaan, ettei XP:tä juurikaan käytetä yksinään tai suoraan muokattunakaan. Heidän mukaansa viitekehyksestä otetaan projekteihin mieluummin jotain toimintatapoja, jotka yhdistetään muuhun toimintaan. Jotkin XP:n osat, kuten ohjelmakoodin yhteisomistus (eng. collective code ownership), vaikuttavatkin sopivan erittäin hyvin suuriin projekteihin. Suurissa projekteissa voi dokumentaation kanssa olla haasteita vain projektin ja ohjelman suuren koon takia, jolloin esimerkiksi ohjelmankoodin omistajuuden avulla voisi olla helppoa ylläpitää ohjelman toteutukseen ja toimintaan tarvittavia tietoja.

FDD on Scrumiin ja XP:hen verrattuna helpommin skaalattavissa suuremmille projekteille. Koska FDD:ssä tehdään tarkka suunnitelma ohjelman toteuttamisesta, voidaan sen pohjilta jakaa pienempiä kokonaisuuksia helposti useiden ohjelmointitiimien välillä. (Aitken & Ilango, 2013) Periaatteiltansa FDD vaikuttaa siis sopivalta suuriinkin ohjelmistoprojekteihin, mutta ylläpidettävän ohjelmistosuunnitelman kanssa pitää olla varovainen. Aziz Buttin et al. (2022) mukaan FDD ei ole paras toimintatapa yksinään suurille projekteille, eikä korkean tason ohjelmiston kuvaus ole tarpeeksi toiminnan tarkalle ohjaamiselle. Koska projektin suunnitelma on yleensä yhden työntekijän päävastuulla, tuottaa se paljon töitä suuressa projektissa ja altistaa vakaville henkilöstömuutosriskeille (Aziz Butt et al., 2022).

Ketterän kehityksen tapa toteuttaa ja painottaa kommunikaatiota toiminnassa on helpompi toteuttaa pienemmissä kuin suurissa projekteissa. Jatkuvasti pidettävät palaverit ovat helpompi toteuttaa pienen tiimin kesken kuin kymmenien tai jopa satojen muiden työntekijöiden kanssa. (Aziz Butt et al., 2022) Tiedon siirtäminen palavereiden välillä vie aikaa muulta toiminnalta projektissa ja tarve tiedon siirtämiselle kasvaa, kun projektiin osallistuvien työntekijöiden määrä kasvaa. Globaalissa toiminnassa on tapaamisten aikatauluttaminen normaalia haasteellisempaa maiden välisten erojen takia, minkä takia kommunikaatio on tunnistettu yleisimmäksi haasteeksi Anwerin et al. (2019) tekemässä tutkimuksessa. Käsitellyt ketterän kehityksen viitekehykset eivät helpota haasteita kommunikaation toteuttamisessa ja se jääkin organisaatiolle itse hoidettavaksi. Ketterää kehitystä täsmällisesti noudattavien projektien tehokkuus vaikuttaa siis laskevan projektin koon kasvaessa, mikä vaikeuttaa niiden toteuttamista suurissa projekteissa.

Vaikka käsitellyt viitekehykset sisältävät kaikki omat haasteensa, ei perinteiset toimintatavat ole niitä suoraan parempia. Petersenin et al. (2009) tekemässä tutkimuksessa tunnistettiin vesiputousmallia noudattavan ohjelmistoprojektin sisältävän paljon erilaisia haasteita ohjelmiston tuotannon eri vaiheissa. Tutkimuksen loppuun kohdeorganisaatio päätyi siirtymään ketterään kehitykseen tunnistettujen haasteiden takia.

Vetämällä käsitellyt viitekehykset yhteen saadaan muodostettua taulukko 2. Taulukossa arvioidaan viitekehysten toimivuutta suurissa ohjelmistoprojekteissa projektinhallinnan, ohjelmiston tuotannon ja muutostenhallinnan näkökulmasta. Taulukossa arviointi on tehty asteikolla 1-3, jossa 1 tarkoittaa hyvää, 2 tyydyttävää ja 3 heikkoa. Vesiputousmalli tunnistettiin toimivaksi projektinhallinan näkökulmasta, mutta jäykäksi muutostenhallinnan ja ohjelmistontuottamisen puolesta. XP ja FDD ovat molemmat hyviä ohjelmiston tuottamisessa, mutta toimintansa puolesta aiheuttavat haasteita muutostenhallinnassa ja erityisesti projektinhallinnassa. Scrum on hyvä ohjelmiston tuottamisessa ja muutostenhallinnassa, mutta sen hallinta on suurissa projekteissa vaikeata. LeSS korjaa Scrumin vaikeuksia projektinhallinnassa, mutta ohjelmistontuotannon tehokkuus laskee kommunikaatiokustannusten takia.

Taulukko 2: Viitekehysten toimivuus suurissa ohjelmistoprojekteissa

Viitekehys	***Projektinhallinta***	***Ohjelmiston tuotanto***	***Muutostenhallinta***
Vesiputous	1	3	3
XP	3	1	2
FDD	3	1	2
Scrum	3	1	1
LeSS	2	2	1

1=Hyvä, 2=Tyydyttävä, 3=Heikko

4. PÄÄTELMÄT

Toteutettavat ohjelmistoprojektit monimutkaistuvat ja niiden ympäristöriippuvuus nousee ohjelmistoalan kasvaessa, aiheuttaen vaikeuksia niiden hallinnassa. Suurien ja pitkien ohjelmistoprojektien ympäristöt eivät pysy vakioina projektien aikana, mikä aiheuttaa muutosriskejä projektien toteutuksessa. Hallitsemattomat muutokset voivat aiheuttaa projekteissa ongelmia, pahimmillaan jopa kaatumisriskejä. Muutosten hallitseminen ei kuitenkaan ole suurissa projekteissa suuren työntekijämäärän ja projektien monimutkaisuuden takia helppoa. Oikeaoppisella muutostenhallinnalla pystyisivät ohjelmistoprojekteja johtavat käyttämään resurssejaan tehokkaammin ja laskien mahdollisia muutosten aiheuttamia riskejä. Tässä työssä tutkittiin kyseisiä ohjelmistoprojektin aikana tapahtuvia muutoksia ja miten näihin muutoksiin voitaisiin suurissa ohjelmistoprojektissa regoida. Tarkoituksena oli vastata alussa asetettuihin tutkimuskysymyksiin:

1. Minkälaisia muutoksia voi ohjelmistoprojektissa tapahtua ja mistä ne johtuvat?
2. Minkälaisia menetelmiä on suurien ohjelmistoprojektien muutosten hallintaan ja minkälaisia riskejä nämä aiheuttavat?

Ohjelmistoprojektin aikana voi tapahtua paljon erilaisia muutoksia. Muutokset ovat joko projektille ulkoisia tai sisäisiä, jotka voidaan vielä jakaa prosessin ja projektin perusteella. Sisäisiin muutoksiin on projekin ja organisaation paljon helpompi vaikuttaa, mutta ulkoiset muutokset ovat paljon haasteellisempia. Ulkoiset muutokset voivat olla erittäin kriittisiäkin ja koska projektit eivät voi vaikuttaa niihin etukäteen, vaativat ne tehokkaat muutostenhallintatavat projektille.

Muutokset saatiin vielä jaoteltua tarkemmin niiden kategorian perusteella henkilöstöön, työkaluihin, teknologiaan ja ympäristöön. Jaottelukategoriat eivät kuitenkaan ole yksiselitteisiä ja kategorioiden välillä on päällekkäisyyksiä. Muutosten kategorisoinnissa ei tarvitse olla täydellinen, mutta ne ovat tärkeitä muutosten lähtökohtien tarkemmassa tutkimisessa, minkä avulla voidaan suurissa projekteissa tehostaa muutostenhallintaa. Suurimmaksi yksittäiseksi muutokseksi tunnistettiin asiakas- ja alkuvaatimusten muuttuminen. Tämä oli kirjallisuudessa selkeästi tutkituin ja yleisin käsitelty muutos.

Koska suurin osa tunnistetuista kriittisistä muutoksista ovat projektin ulkopuolisia, on hyvät muutostenhallintaprosessit tärkeitä projektin onnistumiselle. Useampaa erilaista muutostenhallintaprosessia tutkittiin ja niiden toimivuutta arvioitiin. Tärkeimmiksi osiksi muutostenhallintaprosessissa tunnistettiin toteutettavien muutosten nopea

tunnistaminen ja toteuttaminen sekä muutosten ja niiden vaikutusten jatkuva analysointi prosessin aikana ja sen lopussa. Jatkuvan analysoinnin avulla varmistetaan, ettei tarpeettomia tai haitallisia muutoksia päädytä projektissa toteuttamaan. Analysointi myös auttaa muutosten kriittisyyden jatkuvassa arvioinnissa, minkä avulla varmistetaan muutosten toteutuminen tärkeysjärjestyksessä.

Yleisimmiksi ohjelmistotuotannon viitekehyksiksi tunnistettiin vesiputousmalli, XP, Scrum ja FDD. Vesiputousmalli tunnistettiin jo vanhentuneeksi toimintatavaksi, sillä se ei pysty kankeutensa takia käsittelemään muutoksia tehokkaasti, eikä se ole erityisen tehokas itse ohjelmiston tuottamisessa. Vesiputousmalli on kuitenkin tehokas projektinhallinnassa, minkä takia se on vielä yleisesti käytössä suurissa projekteissa. XP ja Scrum ovat yksinään haasteellisia suurissa ohjelmistoprojekteissa, sillä niiden tiimityöskentely ei mahdollista suurien työntekijämäärien ylläpitoa. FDD toimii suurissa ohjelmistoprojekteissa paremmin, sillä se ei sisällä suoraan rajoitteita työntekijöiden määrälle ja sen avulla ohjelmistoa on helppo jakaa tehtäväksi työntekijöiden välille. FDD:llä on kuitenkin muita haasteita toiminnan kannalta ja on hieman herkempi muutoksille. Ketterät viitekehykset ovat kaikki ohjelmiston tuottamisen puolesta kuitenkin perinteistä toimintaa parempia.

Ketterät viitekehykset eivät ole kovin tehokkaita yksinään suurissa ohjelmistoprojekteissa ja niiden vaatimat kommunikaatiotavat syövät suuremmissa projekteissa suhteellisesti enemmän resursseja perinteiseen toimintaan verrattuna. Parhaalta vaikuttava vaihtoehto suurille ohjelmistoprojekteille on räätälöity yhdistelmä perinteistä toimintaa ketterää kehitystä toteuttavilla tiimeillä. Käsitelty LeSS-viitekehys on hyvä esimerkki suurille projekteille sopivasta perinteisen ja ketterän kehityksen yhdistelmästä. Sen avulla ei yksittäisten tiimien kokoonpanoon tarvitse vaikuttaa, vain viikottaiset tapaamiset muokataan sopimaan suurelle työntekijämäärälle. LeSS:in avulla saadaan ketterän kehityksen tuomat hyödyt projektille menettäen vain vähän tehokkuutta viikottaisten palaverien muokaamisella.

4.1 Rajoitteet ja jatkotutkimusaiheet

Työn suurimmat rajoitteet johtuvat käsitellystä kirjallisuudesta. Käytettyjen hakusanojen kanssa pystyttiin löytämään tämän työn kannalta kattava ja tarpeeksi laaja saanti kirjallisuutta, mutta suoraan ensimmäisillä hakusanoilla löytyi sopivia artikkeleita vähän. Suurin osa käytetystä kirjallisuudesta tuli helmenkalastuksen avulla muutamasta pääartikkelista, joka saattaa laskea käsitellyn kirjallisuuden sisältämien näkökulmien määrää. Työn tarkoituksena oli tehdä vain pintapuolisempi katsaus muutoksiin muutostenhallintaprosessien ja ketterien viitekehysten ohella, minkä takia liian tarkasti

ohjelmiston sisäisiä muutoksia käsitteleviä artikkeleita ei tässä työssä käsitelty. Valitun artikkeleiden käsittelytavan takia saattaa joitakin näkökulmia puuttua työn aiheesta.

Suoraan suurien ohjelmistoprojektien muutostenhallintaa käsitteleviä artikkeleita löytyi myös rajallisesti. Erilaisia hakusanoja pyrittiin etsimään yhdistämään näitä artikkeleita suurilla projekteilla, mutta sopivia hakusanoja ei löytynyt. Tämän takia mahdollisesti työlle sopivia artikkeleita saattoi jäädä löytymättä ja näin käsittelemättä. Vähäinen saanti suuria ohjelmistoprojekteja käsittelevistä artikkeleista saattaa johtua hakusanoista, mutta toinen mahdollinen syy voi olla suurien organisaatioiden haluttomuus julkaista tutkimuksia omista, mahdollisesti epäonnistuneista, ohjelmistoprojekteista.

Työn avulla tunnistetaan selkeä painotus ohjelmakoodimuutoksiin. Löydettyjen artikkeleiden perusteella voisi hyviä jatkotutkimusmahdollisuuksia olla muiden kuin ohjelmakoodimuutoksien tutkiminen syvällisemmin ja tarkemmin suurien ja pienempien ohjelmistoprojektien erojen tutkiminen. Tarkemmalla muutosten tutkimisella voitaisiin tunnistaa projekteille lisää muutosriskejä, joita ei välttämättä olla vielä tunnistettu. Suurien ja pienempien ohjelmistoprojektien vertaamisella voitaisiin myös tunnistaa tarkemmin projektien koon aiheuttamia eroja ja haasteita.

LÄHTEET

Ahimbisibwe, A., Cavana, R.Y. & Daellenbach, U. (2015) A contingency fit model of critical success factors for software development projects. J. Enterp. Inf. Manag. 28, pp. 7–33. https://doi.org/10.1108/JEIM-08-2013-0060

Aitken, A. & Ilango, V. (2013) A Comparative Analysis of Traditional Software Engineering and Agile Software Development, in: 2013 46th Hawaii International Conference on System Sciences. pp. 4751–4760. https://doi.org/10.1109/HICSS.2013.3

Alawairdhi, M. (2016) Agile development as a change management approach in software projects: Applied case study, in: 2016 2nd International Conference on Information Management (ICIM). pp. 100–104. https://doi.org/10.1109/INFOMAN.2016.7477541

Ali, N. & Lai, R. (2016) A method of requirements change management for global software development. Inf. Softw. Technol. 70, pp. 49–67. https://doi.org/10.1016/j.infsof.2015.09.005

Alqudah, M. & Razali, R. (2016) A review of scaling agile methods in large software development. Int. J. Adv. Sci. Eng. Inf. Technol. 6, pp. 828–837. https://doi.org/10.18517/ijaseit.6.6.1374

Anwer, S., Wen, L. & Wang, Z. (2019) A systematic approach for identifying requirement change management challenges: Preliminary results. pp. 230–235. https://doi.org/10.1145/3319008.3319031

Aziz Butt, S., Piñeres-Espitia, G., Ariza-Colpas, P. & Tariq, M.I. (2022) Project Management Issues While Using Agile Methodology, in: Przybyłek, A., Jarzębowicz, A., Luković, I., Ng, Y.Y. (Eds.), Lean and Agile Software Development, Lecture Notes in Business Information Processing. Springer International Publishing, Cham, pp. 201–214. https://doi.org/10.1007/978-3-030-94238-0_12

Bass, J.M. (2012) Influences on agile practice tailoring in enterprise software development. pp. 1–9. https://doi.org/10.1109/AgileIndia.2012.15

Casper, H., Rexford, A., Riegel, D., Robinson, A., Martin, E. & Awwad, M. (2021) The Impact of the Computer Chip Supply Shortage.

Debnath, N.C., Uzal, R., Montejano, G. & Riesco, D. (2006) Software Projects Leadership: Elements to Redefine “risk management” Scope and Meaning, in: 2006 IEEE International Conference on Electro/Information Technology. pp. 280–284. https://doi.org/10.1109/EIT.2006.252148

Drvodelić Cvitak, L.D. & Car, Ž. (2010) Impact of agile development implementation on configuration and change management in telecom domain, in: The 33rd International Convention MIPRO. pp. 377–381.

Dyba, T. & Dingsoyr, T. (2009) What Do We Know about Agile Software Development? IEEE Softw. 26, pp. 6–9. https://doi.org/10.1109/MS.2009.145

Garousi, V., Tarhan, A., Pfahl, D., Coşkunçay, A. & Demirörs, O. (2019) Correlation of critical success factors with success of software projects: an empirical investigation. Softw. Qual. J. 27, pp. 429–493. https://doi.org/10.1007/s11219-018-9419-5

Highsmith, J. & Cockburn, A. (2001) Agile software development: the business of innovation. Computer 34, pp. 120–127. https://doi.org/10.1109/2.947100

Jayatilleke, S. & Lai, R. (2013) A Method of Specifying and Classifying Requirements Change, in: 2013 22nd Australian Software Engineering Conference. pp. 175–180. https://doi.org/10.1109/ASWEC.2013.29

Jones, C. (1996) Software change management. Computer 29, pp. 80–82. https://doi.org/10.1109/2.485858

Nassif, M. & Robillard, M.P. (2017) Revisiting Turnover-Induced Knowledge Loss in Software Projects, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 261–272. https://doi.org/10.1109/ICSME.2017.64

Petersen, K., Wohlin, C. & Baca, D. (2009) The waterfall model in large-scale development. Lect. Notes Bus. Inf. Process. 32 LNBIP, pp. 386–400. https://doi.org/10.1007/978-3-642-02152-7_29

Principles behind the Agile Manifesto (2001) [online] saatavilla: https://agilemanifesto.org/principles.html (käytetty: 12.4.2023).

Rosca, D., Tepfenhart, W. & McDonald, J. (2003) Software engineering education: following a moving target, in: Proceedings 16th Conference on Software Engineering Education and Training, 2003. (CSEE&T 2003). pp. 129–139. https://doi.org/10.1109/CSEE.2003.1191370

Rudic, T. & Sobajic, V. (2012) Change Management Process of Software Projects. Metal. Int. 17, pp. 179–184.