Jaso Ristimäki

KETTERIEN MENETELMIEN SKAALAAMINEN OHJELMISTOARKKITEHTUURIIN

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2018

TIIVISTELMÄ

Ristimäki, Jaso

Ketterien menetelmien skaalaus ohjelmistoarkkitehtuuriin

Jyväskylä: Jyväskylän yliopisto, 2018, 25 s.

Tietojärjestelmätiede, kandidaatin tutkielma

Ohjaaja(t): Kollanus, Sami

Ketterät menetelmät ovat olleet vahvaasti esillä ohjelmistokehittämisessä sekä -projekteissa jo vuosikausia. Niiden saavuttama suosio ja hyöty on saanut monet yritykset ja organisaatiot yrittämään tai ainakin miettimään kuinka näitä ketterien menetelmien hyötyjä voitaisiin skaalata laajemmin koko ohjelmistoarkkitehtuurin tasolle. Perinteisesti ketteryys on näkynyt yritysten ja organisaatioden yksittäisten osastojen tai pienten tiimien sisällä. Tämän laajentaminen tiimien ja osastojen välille on usein hankalaa. Skaalausta helpottamaan on kuitenkin konsulttien toimesta kehitetty erinäisiä viitekehyksiä, jotka useimmiten pohjautuvat Scrumiin ja Lean-ajattelumalliin. Tutkimuksessa käydään tunnetuimpia viitekehyksiä lävitse, sekä selvitetään niiden eroavaisuudet ja ominaisuudet. Ketterien menetelmien skaalaaminen on usein niin laaja projekti, että ulkopuolinen apu saattaa olla toivottua. Lisäksi skaalautumiseen liittyy erinäisiä rooleja, jotka käydään myös tarkemmin lävitse. Viitekehysmalleista, konsulteista ja rooleista huolimatta ketterien menetelmien skaalaamiseen ei ole vielä vakiintuneita käytänteitä ja toimenpiteeseen liittyy paljon haasteita. Tutkimuksen päätavoitteena on saada selkoa kirjallisuuskatsauksen muodossa skaalautumiseen liittyvistä osatekijöistä ja tunnistaa siihen liittyvät haasteet.

Asiasanat: ketterä ohjelmistoarkkitehtuuri, ketterien mentelmien skaalaus, roolit ketterissä menetelmissä

ABSTRACT

Jaso, Ristimäki

Scaling the Agile Methods into the Software Architecture.

Jyväskylä: University of Jyväskylä, 2018, 25 pp.

Information System Science, Bachelor's thesis.

Supervisor(s): Kollanus, Sami

Agile methods have been strongly involved in software development and projects for years. Their popularity and benefits have led many companies and organizations to try, or at least think, of how these benefits of agile methods could be scaled more broadly to the overall software architecture level. Traditionally, agility has appeared within individual departments or small teams of companies and organizations. Extending it between teams and departments is often difficult. To facilitate scalability, however, consultants have developed a number of reference frameworks that are mostly based on Scrum and Lean thinking. The research is conducted through the most well-known reference frameworks as well as their differences and characteristics. Scaling the agile methods is often such a large project that outsourcing may be desirable. In addition, scalability involves a number of roles, which are also going through more detail. Despite the reference framework models, consultants and roles, there are still no well-established practices for scaling up agile methods and there are many challenges involved. The main objective of the study is to find out in the form of a literature review the scaling elements and to identify the challenges involved.

Keywords: agile software architecture, scaling agile methods, roles in scaling agile

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ketterät menetelmät ovat olleet kuuma aihe 90-luvun loppupuolelta lähtien. Ne siirtävät järjestelmäkehitystä keskitetyistä prosesseista ja työkaluista yksilöihin ja vuorovaikutuksiin, dokumentaatiosta työohjelmille, sopimusneuvotteluista asiakasyhteistyöhön, seuraamalla suunnitelmaa vastatakseen muutoksiin (Jensen ym., 2008). Useiden erilaisten ketterien menetelmien syntyminen on räjähtänyt viime vuosien aikana, eikä se näytä merkkejä lopettamisesta (Abrahamsson ym., 2003). Ketterät menetelmät nähdään usein menestyvinä ja parhaiten soveltuvina pienten nopeisiin muutoksiin kykenevien tiimien yksinoikeutena. Useat organisaatiot haluavat kuitenkin hyödyntää ketterien menetelmien hyviä puolia laajemmin toiminnoissaan, osa-alueissaan ja projekteissaan. Tähän skaalaukseen on kehitetty avuksi useita erilaisia viitekehyksiä. Tutkielmassa käydään näistä viitekehyksistä tarkemmin lävitse muutama yleisesti tunnetuin. Ketterien menetelmien skaalaus on usein suuri muutos ja siihen saatetaan käyttää myös yrityksen ulkopuolista apua. Tutkielmassa syvennytään erilaisiin skaalaukseen liittyviin rooleihin ja mahdollisia ketterien menetelmien skaalaukseen liittyviä yritysten kokemia haasteita käydään myös lävitse. Puhuttaessa skaalaukseen liittyvästä laajemmasta kokonaisuudesta ohjelmistoarkkitehtuurissa, niin tutkielmassa tarkoitetaan pääasiassa Dikert ym. (2016) mukaista ajatusta siitä, että laajamittaisen ketteryyden omaksumisen (large-scale agile adopting) määritelmän täyttämiseksi sovelluskehitysorganisaatiossa on oltava vähintään 50 henkilöä tai vähintään kuusi tiimiä.

Tutkimuksessa pyritään vastaamaan seuraavaan tutkimuskysymykseen: Mitkä asiat vaikuttavat ketterien menetelmien onnistuneessa skaalauksessa? Vastaus tullaan etsimään aiempien tutkimusten ja tieteellisten artikkeleiden pohjalta systemaattisena kirjallisuuskatsauksena.

Johdannon jälkeen seuraavassa kappaleessa tullaan käymään lävitse ketterää kehittämistä perinteisimpien menetelmien Scrumin ja Extreme Programmingin (XP) kanssa, sekä käsitellään Lean-ajattelutapaa. Näiden avulla ymmärretään paremmin ketterien menetelmien ajatusmaailmaa ja toimintatapoja ja se auttaa ymmärtämään millaisia asioita, menetelmia ja

ajtuksia skaalaamisessa tulisi saada siirrettyä ohjelmistoarkkitehtuurin tasolle. Tämän jälkeen seuraavassa kappaleessa tullaan käsittelemään ketteryyden skaalautumista ohjelmistoarkkitehtuuriin viitekehysten ja erilaisten skaalautumiseen liittyvien roolien avulla, sekä käydään lävitse skaalautumiseen liittyviä haasteita. Tämän jälkeen on pohdintaosio ja yhteenveto, jossa on pyritty löytämään vastauksia tutkimuskysymystä varten. Lopuksi kirjallisuuskatsauksessa käytet lähteet.

2 KETTERÄ KEHITTÄMINEN

Ketteryyteen liitetään usein notkeus, ripeys ja nopeus. Nämä mielikuvat näkyvät myös ketterissä menetelmissä. Raskaiden elementtien pois karsiminen ja nopea reagointi muuttuvaan ympäristöön ja asiakkaiden vaatimuksiin ovat ominaista ketterille menetelmille (Erickson ym., 2005). Yleisesti ajatellaan ketterän ohjelmistosuunnittelun alunperin saaneen alkunsa Yhdysvalloissa vuonna 2001 julkaistusta ketterästä manifestistä (Agile Manifesto). Ketterien menetelmien keskeisimmät ominaisuudet Jalalinin & Wohlinin (2010) mukaan ovat: jatkuva vaatimusten kerääminen, kasvokkain tapahtuva viestintä, pariohjelmointi, refaktorointi, jatkuva integrointi, varhainen asiantuntijapalaute ja minimaalinen dokumentaatio. Ketterät menetelmät ovat kehittyneet vastauksena aiemmin suosittujen vesiputousmallien (Waterfall Model) epäkäytännöllisyyteen (Hibbs & Jewett, 2010). Tässä luvussa käsitellään kaksi yleisesti ohjelmistokehityksessä käytössä olevaa menetelmää, Scrum ja Extreme Programming (XP). Luvun toinen osio käsittelee Japanista lähtöisin olevaa Lean-ajattelutapaa.

2.1 Scrum

Scrum on laajalti tunnettu ketterä ohjelmistoprosessimalli, joka on erityisesti suunniteltu ei-teknisiin toimintoihin ohjelmistokehityksessä (Hurtado Alegria ym., 2010). Scrumin kehittäjinä pidetään Jeff Sutherlandia, Ken Schwaberia, John Scumniotalesia ja Jeff McKennaa, saman tyylisiä työskentelymetodeja tai ajatuksia niistä on esitetty jo aiemmin vuonna 1986 Hirotaka Takeuchin ja Ikujiro Nonakan toimesta. Varsinaisen Scrum-viitekehyksen katsotaan kuitenkin syntyneen vuonna 1993 ensin mainittujen henkilöiden toimesta. Scrum on suunniteltu lisäämään energiaa, keskittymistä, selkeyttä ja läpinäkyvyyttä projektin suunnitteluun ja täytäntöönpanoon, sen avulla voidaan:

- Lisätä kehityksen nopeutta.

- Kohdentaa yksittäiset ja yrityksen tavoitteet.
- Luoda kulttuuria, jota ohjaa suorituskyky.
- Antaa tukea osakkeenomistajien arvon luomiselle.
- Saavuttaa vakaa ja johdonmukainen suorituskyvyn viestintä kaikilla tasoilla.
- Parantaa yksilön kehitystä ja elämänlaatua. (Sutherland, 2007).

Scrum on saanut nimensä englantilaisesta rygby urheilulajista. Rising & Janoff (2000) kuvaavatkin rygbyn ja scrumin yhtäläisyyksiä seuraavasti; tiimit toimivat tiukkoina, integroituna yksikköinä joissa jokaiselle tiiminjäsenellä on tarkoin määritelty rooli ja koko tiimi keskittyy yhteen tavoitteeseen. Jokaisen tiimin jäsenen täytyy tietää myös oma roolinsa ja tehtävänsä jokaisessä inkrementissä. Työskentely Scrumissa on jaettu kehitysjaksoihin, joita kutsutaan sprinteiksi, yhden sprintin kesto on tavallisesti viikosta kuukauteen. Sprintin sisältö sovitaan aina etukäteen suunnittelupalaverissa ja sprintin lopuksi aikaansaannokset esitellään sidosryhmille ja otetaan palaute vastaan. Sprintin tavoitteita ei voi muuttaa keken sprintin, mutta jokaisen jakson loputtua tuotteenomistaja (PO) voi lisätä uusia ominaisuuksi projektiin, vaikka niitä ei oltaisi aiemmin määritelty (Srivastava ym., 2017).

Scrumissa työskentelevät henkilöt muodostavat scrum-tiimin. Nämä scrum-tiimit on suunniteltu maksimoimaan joustavuus ja tuottavuus, ne ovat itsenäisiä ja työskentely tapahtuu iteraatioissa (Hurtado Alegria ym., 2010). Scrum-tiimiin kuuluu tuotteenomistaja, scrummaster ja kehitystiimi. Sutherland (2007) luettelee edellämainittujen roolien vastuut seuraavasti: tuoteenomistajan päävastuisiin kuuluu tuotteen ominaisuuksien määrittäminen, julkaisupäivämäärän päättäminen ja sisältö. Lisäksi hän vastaa tuotteen kannattavuudesta (return on investment, ROI) ja priorisoi ominaisuudet markkina-arvon mukaan. PO joko hyväksyy tai hylkää työn tulokset. Scrummasterin vastuulla on varmistaa, että tiimi on täysin toimiva ja tuottava. Hän myös mahdollistaa tiiviin yhteistyön kaikissa tehtävissä ja toiminnoissa, sekä myös poistaa mahdolliset esteet. Lisäksi scrummaster suojaa tiimin ulkopuolisista häiriöistä, sekä valvoo prosessin noudattamista. Hänen tehtäviinsä kuuluu myös kutsua koolle sprintin suunnittelu- ja katsauskokoukset. Kehitystiimi muuttaa jokaisen sprinttiin valitut asiat mahdollisiksi julkaisukelpoisiksi valmiiksi tuotteiksi. Ainoastaan kehitystiimin jäsenet osallistuvat tuoteversion kehitykseen Kehitystiimi työskentelee sprintin päämäärän saavuttaakseen ja heillä on oikeus tehdä kaikki projektin suuntaviivojen rajoissa tehtävät toimenpiteet tavoitteen saavuttamiseksi. Kehitystiimi järjestää itesensä ja työtehtävät, sekä vastaa tulosten esittelystä tuotteenomistajalle.

Scrumin tuotokset voidaan jakaa kolmeen osaan, tuotteen kehitysjono (product backlog), sprintin tehtävälista (sprint backlog) ja edistymiskäyrä (burndown chart). Hurtado Alegria ym. (2012) määrittelee osioiden tehtävät seuraavasti: kehitysjono on priorisoitu luettelo tuotteen ominaisuuksista, sprintin tehtävälista on luettelo tehtävistä suoritettavaksi sprintissä, tuottaen lisäyksen potentiaaliseen valmiiseen tuotteeseen kehitysjonosta. Edistymiskäyrä

kuvaa tuotteen ja/tai sprintin jäljellä olevaa aikaa. Viimeisten vuosien aikana scrumista on tullut ketterien menetelmien kasvot ja usein näitä käytetään synonyymeina toisilleen. Yksi syy tähän saavutettuun suosioon on se, että scrum antaa tuotteenomistajan aloittaa projektin ilman laajaa etukäteissuunnittelua (Srivastava ym., 2017).

2.2 Extreme Programming

Extreme programming (XP) on yksi ketteristä tekniikoista. Ohjelmistokehittäjien keskuudessa se on yleinen käytäntö yhteisen ymmärtämisen rakentamiseen ja tiedon jakamiseen tiimin jäsenten välillä (Sadath ym., 2018). XP sai alkunsa 90-luvun loppupuolella Kent Beckin, Ward Cunninghamin ja Ron Jeffriesin toimesta ja se on kehitetty sopimaan erityisesti riskialteille projekteille, joissa vaatimukset vaihtuvat useasti. XP:lle ominaista on, että ohjelmistokehittäjillä olisi mahdollisimman vähän tarpeetonta työtä ja he pystyisivät paremmin keskittymään olennaiseen, joten esimerkiksi kattavaa dokumentointia on karsittu paljon (Beck, 1999). Päämääränä on tuottaa mahdollisimman yksinkertainen lopputulos, joka kuitenkin täyttää aina kaikki sille asetetut sen hetkiset vaatimukset. Refaktoroinnissa on XP:n keskuudessa vahva merkitys, tällä tarkoitetaan ohjelmistokoodin muuttamista parempaan muotoon, ilman että itse ominaisuuksia muutetaan. XP-tiimit muodostuvat yleensä 6-12 hengen ryhmästä, viestintä pidetään optimaalisena ja yksinkertaisena, jotta muuttuviin vaatimuksiin pystytään reagoimaan (Holzinger ym., 2005). Lisäksi asiakas on jatkuvasti mukana kehitysprosessissa jatkuvalla palautteella.

XP koostuu viidestä perusarvoista, joita ovat kommunikointi, yksinkertaisuus, palaute, rohkeus ja kunnioitus, sekä 12 käytänteestä. Lindstrom & Jeffries (2004) avaavat tarkemmin perusarvoja seuraavasti:

• Kommunikointi. XP korostaa asianmukaisen viestinnän merkitystä. Keskustelut tapahtuvat kasvotusten, mutta myös muita mekanismeja suositaan, esimerkiksi piirtämistä.

• Yksinkertaisuus. Tarkoituksena on välttää jätettä ja tehdä vain ehdottoman välttämättömiä asioita, kuten pitää järjestelmän suunnittelu mahdollisimman yksinkertaisena, jotta sitä on helpompi ylläpitää, tukea ja tarkistaa. Yksinkertaisuus tarkoittaa myös vain vaatimuksia, joita tiedetään, ei yritetä ennustaa tulevaisuutta.

• Palaute. Tiimeillä on jatkuvasti palautetta aikaisemmista ponnisteluistaan, ja he voivat tunnistaa parannuksia ja muuttaa toimintatapojaan. Palaute tukee myös yksinkertaista muotoilua. Tiimi rakentaa jotain, kerää palautetta suunnittelusta ja toteutuksesta ja säätä tuotteen eteenpäin.

• Rohkeus. Tarvitaan rohkeutta nostaa esille organisaation ongelmia, jotka vähentävät tiimin tehokkuutta. Tarvitaan rohkeutta lopettaa jotain, joka ei toimi

ja kokeilla jotain muuta. Tarvitaan rohkeutta hyväksyä ja toimia palautetta vastaan, vaikka sitä on vaikea hyväksyä.

• Kunnioitus. Tiimin jäsenten on kunnioitettava toisiaan, jotta he voivat kommunikoida keskenään, tarjota ja hyväksyä asiakassuhdetta kunnioittavaa palautetta ja työskennellä yhdessä tunnistamaan yksinkertaiset mallit ja ratkaisut.

Yksi XP:n kehittäjistä, Kent Beck, listaa XP:n käytänteiksi, suunnittelupelit, pienet julkaisut, järjestelmän metaforan, yksinkertaisen rakenteen ja suunnittelun, testauksen, uudelleenrakentamisen, pariohjelmoinnin, yhteisomistajuuden, jatkuvan integroinnin, 40-tuntisen työviikkon, paikan päällä olevan asiakkaan ja koodausstandardit (Beck, 1999). Näitä käytänteitä ei kuitenkaan avata tässä tutkielmassa tarkemmin. Tiivistettynä XP perustuu yksinkertaisuuden, viestinnän, palautteen ja rohkeuden arvoihin. Se toimii tuomalla koko tiimin yhteen yksinkertaisten käytäntöjen läsnäollessa, ja siinä on tarpeeksi palautetta, jotta ryhmä voi nähdä missä ovat ja miten käytäntöjä mukautetaan heidän ainutlaatuiseen tilanteeseensa (Lindstrom & Jeffries, 2004).

2.3 Lean

Lean-ajattelutapa on lähtöisin Japanista Toyotan tehtaalta Toyota Production Systemin (TPS) pohjalta 1950-luvulta. Lean-termi yleistyi käytössä tosin vasta 1990-luvulla, samaan aikaan kuin sen menetelmiä ruvettiin soveltamaan useammalla toimialalla. Toyota oli 1990- luvun alkuun mennessä 60% tuottavampi ja 50% vähävikaisempi kuin sen Lean-ajattelutapaa toteuttamattomat kilpailijat. Tämän nähtiin johtuvan pääasiassa siitä, että Toyota toteutti hyvin Leania, pyrkimällä eliminoimaan kaiken ylimääräisen (Hibbs & Jewett, 2010). Taiichi Ohno on tutkinut ja kirjoittanut paljon Toyotan Product Systemistä ja hän onkin kuvannut sitä osuvasti ”absoluuttiseksi turhuuden eliminointi systeemiksi”.

Viisi keskeisintä ajatusta Lean-ajattelutavan takana Razzakin (2016) mukaan ovat:

• Arvo (Value), jonka määrä perustuu asiakkaan näkemykseen.

• Arvoketju (Value Stream), sekä kaiken arvoa tuottamattoman poistaminen.

• Imuohjaus (Flow), asiakkaan tarpeiden perusteella muodostettu arvoketju.

• Eteenpäin suuntaus (Pull), työntekijöiden osallistamisen kehittämiseen.

• Täydellisyys (Perfection), toimintaa täytyy kehittää jatkuvasti eteenpäin.

Leanin pohjalla on myös seitsemän periaatetta:

• Poista jätteet.

• Rakenna laatua.

• Luo tietoa.

- Viivytä sitoumuksia.
- Toimita nopeasti.
- Kunnioita ihmisiä.
- Optimoi kokonaisuus. (Poppendieck, 2007).

Kuten aiemmin todettiin, Leanin keskiössä on hukan tunnistaminen ja poistaminen nopeasti, jotta voidaan parantaa laatua sekä pienentää kustannuksia. Leanissa hukka tarkoittaa kaikkea mikä ei lisää suoraan asiakkaiden arvoa tai lisää tietoa siitä, miten tuottaa tätä arvoa enemmän (Poppendieck & Cusumano, 2012).

3 KETTERYYDEN SKAALAAMINEN OHJELMISTOARKKITEHTUURIIN

Tällä hetkellä maailmanlaajuisesti ketteräohjelmistokehitys on tullut Shameemin ym. (2017) mukaan entistä haasteellisemmaksi maantieteellisten rajojen hälventyessä ja sosiokultturisten erojen sekoittuessa. Tämän kokonaisuuden hallitsemiseen on kehitetty useita viitekehyksiä, joista muutama yleisin käydään tarkemmin lävitse tämän luvun toisessa osiossa. Ensimmäisessä osiossa käydään yleisellä tasolla lävitse, mikä ja mitä on ohjelmistoarkkitehtuuri. Kolmas osio käsittelee erilaisia rooleja ja niiden merkitystä ketterien menetelmien skaalaamisessa. Erilaisista viitekehyksistä huolimatta käytännössä kohdataan hankalasti hallitsettavia haasteita ja näitä haasteita käydään tarkemmin lävitse tämän luvun viimeisessä osiossa.

3.1 Ohjelmistoarkkitehtuuri

Arkkitehtuuri on organisaation esitys, joka mahdollistaa organisaation muutosten suunnittelun. Se sisältää nykyiset ja tulevaisuuden liiketoimintatavoitteet, visiot, strategiat, liiketoimintaprosessit, ihmiset, organisaatiorakenteet, sovellusjärjestelmät ja teknologiset infrastruktuurit (Pereira & Sousa, 2005). Koko yrityksen tasolla tätä voidaan kutsua kokonaisarkkitehtuuriksi tai yritysarkkitehtuuriksi. Ohjelmistoarkkitehtuuri on oma osansa yritysarkkitehtuurissa. Tässä luvussa ja tutkielmassa käsiteltävät asiat käsittelevät arkkitehtuuria juuri ohjelmistoarkkitehtuurin tasolla.

Sana arkkitehtuuri viittaa rakennusteollisuudessa perusrakenteeseen tai rakennuksen päärakenteeseen, se sisältää käyttö- ja laatuvaatimukset, käyttöiän, vaatimukset rakennusmateriaaleista sekä valmiiksi valmistetut liitoskomponentit (Gong ym., 2010). Ohjelmistoarkkitehtuuri viittaa ohjelmiston perusrakenteeseen ottaen huomioon ohjelmistovaatimukset ja toiminnalliset vaatimukset. Ohjelmistoarkkitehtuuri on kuin rakennelma joka koostuu ohjelmien tai järjestelmien osista, niiden keskenäisistä suhteista, lisäksi sen suunnittelua ja kehitystä ohjaavat periaatteet ja ohjeistukset (Dobrica &

Niemelä, 2002). Ohjelmistoarkkitehtuurin voidaan nähdä olevan tuotteen tai palvelun keskiö, joka saattaa yhteen tekniset vaatimukset sekä liiketoiminnan tarpeet. Hyvältä ohjemistoarkkitehtuurilta vaaditaan suorituskykyä ja turvaa, sen on oltava myös helposti hallittava kokonaisuus, joka on valmiina muutoksiin.

Ohjelmistoarkkitehtuuria on mahdollista havaita myös useista näkökulmista, joista kukin paljastaa erilaisia näkökohtia. Christensen ym. (1999) esittelevät kolme näkökulmaa: moduulinäkymä, prosessinäkymä ja käsitteellinen näkymä. Moduulinäkymässä kuvataan mitkä ovat tärkeät moduulit ja osa-moduulit. Prosessinäkymässä kuvataan käynnissä olevat prosessit ja niiden väliset suhteet. Käsitteellinen näkymä kuvaa arkkitehtuurin tärkeimmät komponentit kehittäjien mielestä, tästä on hyötyä kun yritetään ymmärtää ongelma-alueita ja nykyisen järjestelmän suhteita niihin (Christensen ym., 1999).

3.2 Viitekehysmallit

Ketterien menetelmien skaalaukseen on kehitetty useita erilaisia, yleensä konsulttien kehittämiä viitekehysmalleja. Tässä tutkielmassa keskitytään niistä viiteen yleiseen. Nämä viitekehykset ovat Scrum of scrums (SoS), Scaled Agile Framework (SAFe), Large-Scale Scrum (LeSS), Disciplined Agile Delivery (DAD) ja Lean Scalable Agility for Engineering (LeanSAFE). TAULUKKO 1 osoittaa näiden viiden ketterän viitekehyksen eroavaisuudet ja pääpiirteet.

SoS on viitekehyksistä laajimmiten käytössä (Ebert & Paasivaara, 2017). Se on perinteisen Scrumin tapaan erittäin joustava tapa toimia ja se soveltuu hyvin niin ohjelmistokehitykseen kuin isoihin järjestelmiinkin. Scrum of scrums ei ole asetelmaltaan kovin monimutkainen ja sen kustannuksetkin ovat melko pieniä. Sen soveltaminen on mahdollista myös maailmanlaajuisesti jakaantuneissa tiimeissä. SoS:n pitäisi mahdollistaa Scrumin laajentamisen kaikkiin tilanteisiin kaiken kokoisena. Paasivaara ym. (2012) kuitenkin toteaa, että SoS-kokoukset näyttäisivät toimivan huonosti, silloin kun tiimissä on liian monta osallistujaa joilla on eriävät mielenkiinnot ja huoleenaiheet, kun taas tilanteet joissa tiimillä on yhtenäiset tavoitteet ja intressit, näyttäisivät olevan kykenevämpiä havaitsemaan menestystekijät.

SAFe julkaistiin vuonna 2011 ja se on yksi tunnetuimmista skaalaukseen liittyvistä viitekehyksistä. Sen pohjalla on Scrumin lisäksi viitteitä myös muista ketterien menetelmien periaatteista, sekä Lean-ajattelutavasta. SAFe:n kohteena ovat suuret yritykset, tosin usein se koetaan hieman turhan raskaaksi viitekehykseksi. SAFe myös määrittelee paljon rooleja ja ohjenuoria ja osa sen käyttäjistä on kokenut sen jopa lisäävän byrokratiaa ja näin luoden uuden vesiputousmallin (Ebert & Paasivaara, 2017).

LeSS julkaistiin vuonna 2008 Craig Larmanin ja Bas Vodden toimesta. Sitä voidaan kuvailla skaalauksessa Scrumin laajennukseksi sääntöjen ja ohjeistuksien osalta, eikä siinä tarvitse kuitenkaan luopua Scrumin

alkuperäisistä päämääristä ja tavoitteista. Tiimien roolit on LeSSissä toteutettu niin, että tiimien vetäjien ja projektipäälliköiden roolit on poistettu kokonaan. Tavallinen LeSS-viitekehys tarjoaa ohjeet ja tekniikat ketterään kehitykseen kokoonpanolle joka muodostuu alle kymmenestä erillisestä tiimistä (Uludag ym., 2017).

Disciplined Agile Delivery (DAD) sai alkunsa vuonna 2007 Scott W. Amblerin toimesta hänen työskennellessään IBM:llä. DADia voidaan kutsua hybridimalliksi, sillä siinä yhdistyvät Lean-ajattelutavan ja ketterien menetelmien yleisimmät käytänteet (Ambler & Lines, 2012), tästä johtuen se on myös koettu osittain hieman monimutkaiseksi, sillä se yhdistelee useita malleja (Ebert & Paasivaara, 2017). DAD:lle ominaisia arvoja ovat ihmiset ensin (people first), selkeä skaalaustuki (explicit scaling support) ja tavoitteellisuus (goal-driven). Brown ym. (2013) avaavat näitä arvoja seuraavasti: ihmiset ensin, tarkoittaa että tiimin sisällä ei pitäisi olla hierarkioita ja tiimin jäseniä kannustetaan toimimaan myös muissa tehtävissä kuin erikoisaloissaan. Selkeällä skaalaustuella taas tarkoitetaan, että on olemassa useita skaalaustekijöitä, joita tiimi voi joutua käsittelemään. Nämä skaalaustekijät ovat tiimin koko, maantieteellinen jakelu, lainsäädännön noudattaminen, tekninen monimutkaisuus, organisatorinen monimutkaisuus ja yrityksen kurinalaisuus. Tavoitteellisuudella tarkoitetaan, että vältetään räätälöityjä haasteita, johon liittyy useampia määritteleviä menetelmiä. Sen sijaan yritetään räätälöidä valintoja ohjaavat skaalaustekijät, jotka kohtaavat tiimin taitojen ja kulttuurin kanssa (Brown ym., 2013).

TAULUKKO 1 Viitekehysten erot.

Kriteeri	Viitekehys				
	Scrum of Scrums (Sos)	Scaled Agile Framework (SAFe)	Large-Scaled Scrum (LeSS)	Disciplined Agile Delivery (DAD)	Lean Scalable Agility for Engineering (LeanSAFE)
Laajuus	Ohjelmistot, laitteistot ja järjestelmät; joustava	Ohjelmistot	Ohjelmistot	Ohjelmistot	Ohjelmistot, laitteitot ja järjestelmät
Differoija	Mahdollistaa Scrumin kaikissa tilanteissa ja eri kokoisina	Monimutkainen, monia artifakteja, rooleja ja ohjesääntöjä	Mahdollistaa joustavuuden tarjoamalla vain ehdoituksia	Monimutkainen, kattaa monet mallit	Toimii kriittisten järjestelmien kanssa
Teknologia	Scrum	Scrum ja muut ketterät menetelmät, Lean	Scrum	Scrum, Lean	Scrum, Lean
Omaksuminen	Käytössä monissa yrityksissä	Käytössä useissa yrityksissä	Käytössä useissa yrityksissä	Käyttö on alkunut	Käyttö on alkanut
Skaalaus	Joustava ja sopeutuu eri asetuksiin	Kohdistettu suurille yrityksille, mutta koetaan raskaaksi	Omaksuttavissa eri asetuksille	Omaksuttavissa eri asetuksille	Omaksuttavissa eri asetuksille
Mutkikkaisuus	Matala	Korkea	Keskitaso	Keskitaso	Keskitaso
Hinta	Matala	Korkea	Keskitaso	Keskitaso	Matala
Tiimien maailmanlaajuinen jakaantuminen	Mahdollinen	Mahdollinen	Mahdollinen	Vaikea	Mahdollinen

Lean Scalable Agility for Engineering eli LeanSAFE on kustannuksiltaan näistä viitekehyksistä edullisin. LeanSAFE käyttää hyväkseen malleja ja tekniikoita sekä Scrumista, että Lean-ajattelumallista. Se määrittelee säännöksiä paljon muita viitekehyksiä vähemmän ja antaa enemmän vapautta tapauskohtaiseen räätälöintiin, joka johtaa siihen, että yritysten on vähitellen koottava omat puitteet ja empiirisesti mitattava millä käytännöillä on parhaat arvot ja sopivuus (Ebert & Paasivaara, 2017).

3.3 Roolit

Yritykset joutuvat toimimaan ennustamattomassa ja erittäin kilpailullisessa ympäristössä, jossa asiakkaiden vaatimukset saattavat muuttua hetkenä minä hyvänsä, eikä virheisiin ole juuri varaa. Tälläisessä ympäristössä myös ohjelmistokehittäminen kohtaa nämä haasteet ja muutokset joko suoraan tai epäsuorasti (Uluda ym., 2017). Ketterät menetelmät tuovat ratkuisuja tämän kaltaisiin tilanteisiin. Niiden skaalaaminen ohjelmistokehityksestä ohjelmistoarkkitehtuurin tasolle ei kuitenkaan ole täysin haasteetonta. Perinteiset ketterät menetelmät, kuten aiemmin esitellyt Extreme Programming ja Scrum eivät vaadi erilaisia arkkitehtuurisia rooleja, mutta skaalautumisessa erilaiset roolit tuovat helpotusta haasteiden kohtaamiseen. Tässä luvussa käymme lävitse ketterien menetelmien skaalautumiseen liittyviä keskeisempiä rooleja ja niiden tehtäviä. Edellisessä osiossa esiteltiin muutamia yleisimpiä ketterien menetelmien skaalautumiseen tarkoitettuja viitekehysmalleja. Niistä osassa ei ole määritelty arkkitehtuurisia rooleja ja osassa on esitetty tarkempia rooleja eri arkkitehdeille. Nämä roolit käsittävät muun muassa yritys-, ohjelmisto-, ratkaisu- ja tieto- tai informaatioarkkitehdin roolit (Uludag ym., 2017).

Seuraavaksi käydään tarkemmin lävitse nämä neljä roolia, mitä ne pitävät sisällään ja mitä heidän vastuullaan on. Yritysarkkitehdin (Enterprise Architect, EA) rooliin kuuluu sekä johtamispuoli, että teknologinen ymmärtäminen (Mapingire ym., 2018). Vaikka skaalautumisissa on paljon yhtäläisyyksiä, niin jokainen tapaus on kuitenkin yrityskohtainen. Näin ollen myös yritysarkkitehdin täytyy hallita useita rooleja, sillä Van den Bergin & Van Vlietin (2016) mukaan tilanteet joissa EA työskentelee vaihtelevat erilaisen sisällön, osaamisen ja osallistumisen välillä. EA: päällimmäisinä vastuina on ylläpitää korkeatasoista ja kokonaisvaltaista näkemystä yrityksen ratkaisuista ja kehityshankkeista, sekä työskenteleminen yrityksen liiketoiminnallisten sidosryhmien kanssa, pyrkien samalla ajamaan teknologista toteutusta yrityksen arvojen mukaisesti (Uludag ym., 2017). Tiivistettynä voitaisiinkin todeta, että yritysarkkitehtinä työskentelvän henkilön täytyy hallita yrityksen strategia, ymmärtää läpi yrityksen kulkevia erilaisia teknologisia puolia ja tehdä teknologisia päätöksiä jotka noudattavat yrityksen omaa linjaa.

Ohjelmistoarkkitehtuuri on olennainen osa ohjelmistokehitystä; sen integrointi kehitysprosessiin on tullut haastavammaksi, kun siirrytään

perinteisestä ketterään kehitystekniikkaan, ja kun arkkitehdeistä on tullut paljon enemmän kuin vain teknisiä suunnittelusta vastaavia asiantuntijoita (Sherman & Hadar, 2015). Ohjelmistoarkkitehtinä työskentelevä henkilö on usein päävastuussa teknisen arkkitehtuurin hyväksymisestä ja valvomisesta. Hänen toimiinsa kuuluu myös teknisten riskien tunnistaminen ja niihin varautuminen. Ohjemistoarkkitehti määrittelee myös järjestelmäarkkitehtuurin johon sisältyy tekninen viitekehys, joka esittää ratkaisun vaatimuksiin ja antaa tarkan kuvan tämän ratkaisun rakenteesta (Uludag ym., 2017). Hyvän ohjelmistoarkkitehtuurin työssä ei ole Erderin & Pureurin (2017) mukaan tärkeintä hänen henkilökohtainen tietämys vaan hänen kykynsä soveltaa vaadittuja välineitä juuri oikeana ajankohtana. Arkkitehtejä ja varsinkaan ohjelmistoarkkitehtejä ei enää nähdä kuitenkaan täysin yksinäisinä päätöksentekijöinä. Varsinkin ketterien menetelmien skaalautumisen yhteydessä, jotta voidaan hyötyä itse ketteryydestä niin tiimien oletetaan saavan myös autonimisiä piirteitä ja pystyvän itse vaikuttamaan päätöksiin ja ohjelmistoarkkitehti toimii ensisijaisesti mentorina tukemalla tiimejä (Britto ym., 2016).

Ratkaisuarkkitehti tukee ratkaisunhallintaa hallinnoimalla yrityksen arvovirtoja. Lisäksi hän keskustelee johdon ja omistajien kanssa tulevista mahdollisuuksista ja yhdenmukaistaa siihen teknisen toteutuksen (Uludag ym., 2017). Ratkuisiarkkitehdin päätehtävänä on ottaa kokonaisvastuu järjestelmäratkaisuista luomalla optimaalisin ratkaisu ottamalla huomioon vaatimukset, asiakkaan ympäristön , rajoitukset ja resurssit. Hän myös avustaa käyttäjäystävällisen käyttöliittymän suunnittelussa sekä varmistaa laadukkaan järjestelmän, joka tarjoaa hyvän suorituskyvyn, tehokkaan ja joustavan käyttöliittymän, mikä ottaa huomioon myös tulevaisuudessa tulevat muutokset (Tandon, 2007).

SAFe on ainoa skaalauksen viitekehyksistä, joka määrittelee roolin tietoarkkitehdille. SAFe:ssa tietoarkkitehti osallistuu kehitykseen tarjoamalla nopeasti saatavaa asiantuntemustaan järjestelmän tai ratkaisun alueella, joka vaatii erityista osaamista tai jotakin tiettyä taitoa (Uludag ym., 2017). Yleensä tietoarkkitehti työskentelee tiiviisti muiden arkkitehtien, scrum masterien ja yhden tai useamman ketterän tiimin kanssa. IT-hallintotavasta (IT governance) tiedottaminen on yleensä tietoarkkitehdin vastuulla. Hän myös toimii passiivisena apuna ja tukena ketterille tiimeille ja sitä kautta ohjelmistokehitykselle. Tietoarkkitehti on myös vastuussa IT-hallintovaatimusten asettamisesta ketterille tiimeille (Uludag ym., 2017).

Osa rooleista ei ole aivan täysin sidoksissa pelkkästään tutkielman teemaan, eli ohjelmistoarkkitehtuuriin. Ne eivät kuitenkaan ole siitä täysin myöskään ulkona ja jokainen rooli kuitenkin sitä koskettaa tavalla tai toisella. Myös ketterien menetelmien skaalaamiseen kehitetyt viitekehykset määrittelevät näitä rooleja enemmän tai vähemmän. Usein arkkitehtuurisissa rooleissa olevat henkilöt IT-alalla eivät toimi yksin vaan arkkitehtien roolit ja vastuut menevät päällekkäin ja he huolehtivat niistä tiimeinä.

3.4 Haasteet

Ketterien menetelmien skaalaukseen liittyy monia haasteita, kuten useiden ketterien tiimien välinen koordinointi, arkkitehtuurin puutteellisuus, vaatimusten analysoinnin puute sekä hajautettujen hankkeiden muut haasteet, kuten suurten organisaatioden hajautuminen (Paasivaara ym., 2018). Tässä kappaleessa käydään lävitse yritysten kohtaamia haasteita skaalatessa ketteriä menetelmiä sovellusarkkitehtuuriinsa.

Tutkijat Reifer, Maurer ja Erdogmus toteuttivat Kanadassa tutkimuksen johon osallistui 35 alan tutkijaa ja liike-elämän asiantuntijaa. He kokoontuivat keskustelemaan ketterien menetelmien skaalautumisen ongelmista ja haasteista. Keskusteluissa nousi esiin seitsemän haasteellisuutta aiheuttavaa kohtaa, joita seuraavaksi tarkastellaan lähemmin. Nämä seitsemän haastetta olivat:

- Epäpuhtaat ketterät menetelmät
- Suuntaviivojen luominen ei-kiinnostaville ketterille hankkeille
- Kasvavien ketterien käytänteiden sovittaminen suuriin hankkeisiin
- Integraatio-ongelmien käsittely kettärässä projektissa
- Ketteryyden skaalaaminen läpi yrityksen sovellusten. (Scaling agile in an enterprise across applications).
- Hajautuvan kehityksen käsittely ketterässä projektissa
- Testauksen integrointi kun järjestelmät kasvavat. (Reifer ym, 2003).

Epäpuhtaat ketterät menetelmät. Tutkimuksen keskusteluihin osallistuivat olivat yhtä mieltä siitä, että ketterät menetelmät soveltuvat pieniin projekteihin, joissa mittakaavaongelmat ovat vähäisiä. Suurin osa oli myös samaa mieltä siitä, että ketterien menetelmien on toimittava maailmassa, jossa molempia, sekä ketteriä, että perinteisiä menetelmiä voidaan käyttää rinnakkain. Tästä päästäänkin kysymykseen: Kuinka skaalata ketteriä menetelmiä ilman, että joudutaan uhraamaan taustalla vaikuttavia ketterän manifestin (Agile Manifesto) periaatteita.

Suuntaviivojen luominen ei-kiinnostaville/sopiville ketterille hankkeille. Yleisesti sopiva ketterä projekti on melko pieni, itseorganisoiva alle 20 kehittäjästä koostuva tiimi, sekä vähintään yksi paikan päällä oleva asiakkaan edustaja. Yleensä tiimi työskentelee yhdessä nopeasti muuttuvalla alueella, sovellukseen kohdistuvat vaatimukset muuttuvat tai nousevat nopeasti. Kommunikointi tapahtuu usein myös suoraan naamakkain (face to face) asiakkaan tai muiden tiimin jäsenten kanssa. Skaalatessa edellämainittuja kohtia suurempaan kokonaisuuteen asiat monimutkaistuvat. Tutkimuksen keskusteluun osallistuneista osa oli sitä mieltä, että ketteriä menetelmiä ei pitäisi käyttää niiden alueiden ulkopuolella joihin ne on jo todettu toimiviksi. Toiset osallistujista taas ehdottivat, että sovellusten kehittäjät skaalaisivat ketteriä menetelmiä suurempiin projekteihin siitä huolimatta tahtooko projektin edustajat sitä vai ei. Kaikki olivat kuitenkin tutkimuksessa yhtä mieltä siitä, että suuntaviivoja ja ohjeistusta skaalaamiseen tarvitaan.

Kasvavien ketterien käytänteiden sovittaminen suuriin hankkeisiin. Keskustelijat pohtivat minkälaisia lisäkäytänteitä skaalaus saattaisi vaatia. Osa ehdotti, että päivittäiset tiimien kokoukset voisi laajentaa koskemaan saman projektin muiden tiimien kanssa käytäviä yhteisiä kokouksia. Suuri osa keskustelusta liittyi skaalauksen vaikutuksista juuri kokouksiin ja sitä kautta vaatimuksiin.

Integraatio-ongelmien käsittely kettärässä projektissa. Useimmat sovellukset käyttävät jo olemassa olevia arkkitehtuureja, sekä osia tai osioita vanhoista ohjelmistoista tai jopa suoraan kaupanhyllystä löytyviä paketteja. Reiferin ym. tutkimuksessa nousi esille kuinka nämä uudelleenkäytettävät komponentit rajoittavat ketterien menetelmien käyttöä skaalauksessa. Keskustelua herätti myös, se kuinka tätä voidaan koordinoida tiimien välillä skaalauksessa. Eli, kuinka tiimit pysyvät toistensa perässä siitä mitä ja minkä verran vanhoja komponentteja kukin tiimi käyttää ja pitäisikö siihen olla jonkinlaiset suuntaviivat tai standardit.

Ketteryyden skaalaaminen läpi yrityksen sovellusten. (Scaling agile in an enterprise across applications). Keskusteluun osallistuvat olivat yhtä mieltä siitä, että sovelluksen kehittäminen ei ole, eikä saa olla erillään muusta yrityksestä. Esiin nousi myös ketterien menetelmien mukanaan mahdollisesti tuomat kommunikointiesteet yrityksen sisällä, sillä sovellukset tuotetaan lyhyissä iteratiivisissa sykleissä.

Hajautuvan kehityksen käsittely ketterässä projektissa. Keskustelua aiheutti tiimin koko ja kuinka se pystyy työskentelemään hajautuneena usein jopa maantieteellisesti eri paikoista. Tutkimukseen osallistuneet eivät olleet yksimielisiä siitä kuinka hoitaa kommunikointiesteet hajaantuneen tiimin kohdalla. Osa kannatti selvää hierarkiaa tiimien välillä sekä vaiheittaista käyttöönottoa. Esimerkiksi niin, että ensiksi arkkitehtuuritiimi, sitten sovelluksen ominaisuuksiin keskittyvä tiimi ja lopuksi integraatiotiimi. Osa taas koki tämän sotivan pahasti itse ketteryyttä vastaan. Yksittäisten tiimien synkronisointi mahdollistaisi sen, että hitain tai heikoin tiimi määrittäisi koko projektin vauhdin ja koko hankkeen kohtalon. Tämä koettiin liian suureksi riskiksi.

Testauksen integrointi kun järjestelmät kasvavat. Yhteisymmärrystä keskustelijoiden kesken oli siitä, että ketterät menetelmät joissa asiakas saadaan kirjoittamaan hyväksymistestejä, kun uusia ominaisuuksia toteutetaan, maksaa itsensä takaisin pitkässä juoksussa. Tästä huolimatta suurissa projekteissa ei ole realistista olettaa asiakkaan riittävää osallistumista jatkuvasti. Keskusteluissa ehdotettiin myös, että jokin tiimi voisi ottaa tehtäväkseen kirjoittaa asiakkaiden hyväksymistestejä.

4 POHDINTA JA YHTEENVETO

Tämä tutkielma on toteutettu systemaatisena kirjallisuuskatsauksena. Kirjallisuuskatsauksen avulla hahmotetaan tutkielman aihepiirin kokonaisuutta. Sen avulla saadaan tietoa siitä, miten paljon tutkimustietoa on olemassa, millaisesta näkökulmasta aihetta on aiemmin tutkittu ja millaisin menetelmin (Hirsjärvi ym., 2009). Isojärven (2017) mukaan hyvin suunniteltu ja toteutettu kirjallisuushaku on edellytys onnistuneelle menetelmäarvioinnille tai muulle systemaattiselle katsaukselle. Systemaattinen kirjallisuushaku:

- On suunniteltu huolellisesti
- On toteutettu käyttäen useaa tiedonlähdettä
- Pyrkii löytämään kaikki relevantit tutkimukset
- On vinoutumaton: ei rajaa pois maita, kieliä jne.
- On raportoitu niin, että se on helposti toistettavissa. (Isojärvi, 2017).

Valtaosaan tämän tutkimuksen lähdemateriaalin hakemiseen on käytetty Google Scholar alustaa, sekä IEEE Xplore digitaalista kirjastoa. Hakusanoina on käytetty muunmuassa seuraavia hakuja: agile software architecture, scaling agile, scaling agile methods ja roles in scaling agile. Pohjustuskappaleisiin, jossa käydään lävitse ketterää kehittämistä ja ohjelmistoarkkitehtuuria yleisesti, on lähteeksi otettu myös vanhempaa tutkimusta, sillä tämän kaltainen tieto ei ole oleellisesti muuttunut tähän päivään mennessä. Varsinaiseen sisältölukuun on lähteitä käytetty lähinnä 2010-luvulla julkaistuja artikkeleita ja tutkimuksia. Vanhempaakin tietoa löytyi jonkin verran, mutta ketterien menetelmien skaalauksen ollessa sen verran tuoretta, varsinkin tutkimuksen osalta, että tutkimuksen kannalta oleellisimmat lähteet ovat julkaisuja aivan viime vuosien ajalta.

Tutkielman tarkoituksena oli käydä läpi millaisia erilaisia ketteriä viitekehyksiä on saatavilla ketterien menetelmien skaalaamiseen sovellusarkkitehtuuriin. Tutkimuskysymyksenä oli, mitä on otettava huomioon onnistuneessa ketterien menetelmien skaalaamisessa? Tutkielman alussa käytiin lävitse ketteristä menetelmistä Scrum ja Extreme Programming sekä Lean-ajattelutapa. Näiden lävitse käymisen tarkoituksena oli helpottaa

skaalautumiseen kehitettyjä viitekehyksiä. Tämän jälkeen käytiin lävitse näiden viitekehysten eroavaisuudet ja kuinka ne kukin tahoiltaan soveltuvat perinteistä sovelluskehitystä laajempaan kokonaisuuteen. Lisäksi selvitettiin erilaisia skaalautumiseen liittyviä rooleja. Tutkimuksessa esiteltiin myös aiempien tutkimuten pohjalta haasteita, joita ketterien menetelmien skaalaamiseen on kyseisissä tutkimuksissa noussut esille sovellusarkkitehtuuritasolle asti. Ketterien menetelmien skaalautuvuus on haastavaa ja lähes jokaiselle organisaatiolle tapauskohtaista. Päällimmäinen havainto on, kuten aiemmin on todettu, että ketterät menetelmät on kehitetty ja ne soveltuvat parhaiten pienten tiimien käyttöön. Näiden pohjalta luodut skaalaukseen soveltuvat viitekehykset voivat kuitenkin tuoda suuria etuja myös suuremmassa mittakaavassa, tai ainakin ketteriä piirteitä, jotka voivat skaalausta helpoittaa. Onnistuneissa ketterien menetelmien skaalauksissa yhteistä oli henkilöstön kouluttaminen hyvissä ajoin, ihmisten tiedottaminen, muutos agentin osallistuminen projektiin sekä ulkopuolisen konsultin palkkaus neuvomaan, kouluttamaan ja tukemaan (Paasivaara, 2017). Voitaisiinkin siis todeta, että samankaltaiset asiat jotka ovat avainasemassa missä tahansa organisaatiota koskevassa suuremmassa muutoksessa, ovat tärkeässä roolissa myös ketterien menetelmien skaalauksessa. Perinteiset ketterät menetelmät, kuten XP tai Scrum, eivät vaadi arkkitehtien roolia. Ketterien menetelmien skaalauksessa taas Uludagin ym. (2017) mukaan arkkitehdeillä ja ulkopuolisilla asiantuntijoilla on suuri rooli onnistuneeseen skaalaukseen. IT-ammattilaisilla on myös vielä opittavaa ”siviili” puolen insinöörien toiminnasta, jossa arkkitehtien ja insinöörien roolit ja vastuut on selkeasti määritelty (Tandon, 2007). Näin se pitäisi olla myös IT-puolella, mutta usein selvät roolit ovat tarkemmin määritelty vain yleisimmille rooleille, kuten ohjelmoijalle tai projektipäällikkölle.

Ketterien menetelmien skaalaus vaatii organisaatiolta tehokkaan rakenteen (Eklund & Berger, 2017). Myös Dikert ym. (2016) ottavat asian esille puhuessaan organisaation sisäisistä siiloista. Näillä siiloilla he tarkoittavat tilanteita, joissa yrityksen sisällä pidetään jotakin erikoistunutta osaamista tai tietoa pidetään tietyn tiimin tai tiimien sisällä. Tämä luo myös haasteita skaalautumiselle ja tämän kaltaiset haasteet voivat myös olla vaikeasti havaittavissa, varsinkin ilman ulkopuolista asiantuntijaa. Palataan varsinaiseen tutkimuskysymykseen, eli mitä on otettava huomioon ketterien menetelmien onnistuneessa skaalaamisessa? Tutkielman pohjalta voidaan todeta, että ei ole olemassa mitään tiettyä kaavaa joka takaisi skaalautumisen onnistumisen, vaan se on monen tekijän summa ja haasteita riittää. Onnistumisen todennäköisyyttä voidaan parantaa valmistautumisella, koulutuksella ja jopa organisaation rakenteella ja hierarkialla voi olla vaikutusta. Avuksi skaalautumisen onnistumiseen on saatavilla apua erilaisista rooleista joko yrityksen sisältä tai skaalautumiseen erikoistuneelta ammattilaiselta ulkouolelta. Viitekehykset ovat apuna skaalauksessa ja valinnaksi jää vain se, että mitä niistä lopulta halutaan lähteä toteuttamaan. Jatkotutkimuksissa voisi yksi suunta olla juuri, se millä perusteilla skaalaukseen valitaan viitekehys, vaikuttaako siihen yrityksessä jo

olevat roolit, aiempi kokemus jostakin viitekehyksestä, konsulttien puheet vai kilpailijoiden onnistumiset tai epäonnistumiset.

LÄHTEET

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. Proceedings of the 25th International Conference on Software Engineering, 244-254. IEEE.

Ambler, W.S., Lines, M. (2012). Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise. International Business Machines Press, 2012.

Beck, K. (1999). Embracing change with extreme programming. Computer, volume 32, issue 10, 1999. IEEE.

Beck, K. (1999). Extreme Programming Explained: Embracing Change. Addison Wesley. Boston (MA). 1999.

Britto, R., Smite, D., Damm, L. (2016). Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study. IEEE Software, Volume 33, Issue 6, 2016. IEEE Computer Society.

Brown, A.W., Ambler, S., Royce, W. (2013). Agility at scale: Economic governance, measured improvement, and disciplined delivery. 2013 35th International Conference on Software Engineering (ICSE). IEEE.

Chistensen, M., Damm, C.H., Hansen, K.M., Sandvad, E., Thomsen, M. (1999). Design and evolution of software architecture in practice. Proceedings Technology of Object-Oriented Languages ans Systems. TOOLS 32. IEEE.

Dobrica, L., Niemela, E, (2002). A survey on software architecture analysis methods. IEEE Transactions on Software Engineering, Volume 28, Issue 7, 2002. IEEE.

Dikert, K., Paasivaara, M., Lassenius, C. (2016). Challenges and success factors in large-scale agile transformations: A systematic literature review. Journal of Systems and Software, volume 119, pages 87-108. 2016.

Ebert, C., Paasivaara, M. (2017). Scaling Agile. IEEE Software, Volume 34, Issue 6. November/December 2017.

Eklund, U., Berger, C. (2017). Scaling Agile Development in Mechatronic Organizations – A Comparative Case Study. 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track.

Erder, M., Pureur, P. (2017). What Type of People Are Software Architects?. IEEE Software, Volume 34, Issue 4, 2017. IEEE.

Erickson, J., Lyytinen, K., Siau, K. (2005). Agile Modeling, Agile Software Development and Extreme Programming: the State of Research. Journal of Databse Management, Vol 16. Iss.4.

Gong, W., Liu, Y., Bi, X., Shi, Z. (2010). Basic contents of software architecture design. 2010 International Conference on Computer Application and System Modeling (ICCASM 2010). IEEE.

Hibbs, C., Jewett, S. (2010). Lean Software Development: One Step at a Time. Systems & Software Technology Conference 2010. IEEE.

Hirsjärvi, S., Remes, P., Sajavaara, P., Sinivuori, E. (2009). Tutki ja kirjoita. Helsinki, Tammi 2009.

Holzinger, A., Errath, M., Searle, G., Thurnher, B., Slany, W. (2005). From extreme programming and usability engineering to extreme usability in software engineering education (XP+UE /spl rarr/ XU). 29th Annual International Computer Software and Applications Conference (COMPSAC'05). IEEE.

Hurtado Alegria, J., A., Bastarrica, M., C., Bergel, A. (2010). Analyzing the Scrum Process Model with AVISPA. 2010 XXIX International Conference of the Chilean Computer Science Society. IEEE.

Isojärvi, J. (2017). Kirjallisuushaku. Versio 1.1. HTA-opas. Helsinki: Suomalainen Lääkäriseura Duodecim. 2017.

Jalali, S., Wohlin, C. (2010). Agile Practices in Global Software Engineering – A Systematic Map. International Conference on Global Software Engineering (ICGSE). IEEE.

Jensen, R.N., Platz, N., Tjørnehøj, G. (2008). Developer Stories: Improving Architecture in Agile Practice. In: Filipe J., Shishkov B., Helfert M., Maciaszek L.A. (eds) Software and Data Technologies. Communications in Computer and Information Science, vol 22. Springer, Berlin, Heidelberg.

Lindstrom, L. Jeffries, R. (2004). Extreme Programming and Agile Software Development Methodologies. Information Systems Management, Vol 21, Issue 3, 2004.

Mapingire, K., Deventer, P., Van der Merwe, A. (2018). Positioning the role of the enterprise architect: An independent study in a mobile telecommunications organisation. 2018 Conference on Information Communications Technology and Society. IEEE.

Paasivaara, M. (2017). Adopting SAFe to Scale Agile in a Globally Distributed Organization. 2017 IEEE 12th International Conference on Global Software Engineering (ICGSE).

Paasivaara, M., Behm, B., Lassenius, C., Hallikainen, M. (2018). Large-scale agile transformation at Ericsson: a case study. Empirical Software Engineering 2018.

Paasivaara, M., Lassenius, C., Heikkilä, V.T. (2012). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE Intenational Symposium on.

Pereira, C.M., Sousa, P. (2005). Enterprise Architecture: Business and IT Alignment. 2005 ACM Symposium on Applied Computing.

Poppendieck, M. (2007). Lean Software Development. 29th International Conference on Software Engineering (ICSE'07 Companion). IEEE.

Poppendieck, M., Casumano, M.A. (2012). Lean Software Development: A Tutorial. IEEE Software, Volume 29, Issue 5, 2012. IEEE Computer Society.

Razzak, M.A. (2016). An Empirical Study on Lean and Agile Methods in Global Software Development. 2016 IEEE 11th International Conference on Global Software Engineering Workshops (ICGSEW). IEEE.

Reifer, D., J., Maurer, F., Erdogmus, H. (2003). Scaling Agile Methods. IEEE Computer Society. 2003.

Rising, L., Janoff, N, S. (2000). The Scrum software development process for small teams. IEEE Software, Volume 17, Issue 4, 2000. IEEE Computer Society.

Sadath, L., Karim, K., Gill, S. (2018). Extreme programming implementation in academia for software engineering sustainability. 2018 Advances in Science and Engineering Technology International Conferences (ASET). IEEE.

Shameem, M., Kumar, C., Chandra, B., Khan, A.A. (2017). Systematic Review of Success Factors for Scaling Agile Methods in Global Software

Development Environment: A Client-Vendor Perspective. Software Engeneering Conference Workshop (APSECW), 2017 24th Asia-Pacific.

Sherman, S., Hadar, I. (2015). Toward Defining the Role of the Software Architect. 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE.

Srivastava, A., Bhardwaj, S., Saraswat, S. (2017). SCRUM model for agile methodology. 2017 International Conference on Computing, Communication and Automation (ICCCA). IEEE

Sutherland, J. (2007). The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process. Wolrwide Scrum Consulting Practice, PatientKeeper inc. 2007.

Tandon, R. (2007). The Role of Solution Architects in Systems Integration. IT Professional, Volume 9, Issue 2, 2007. IEEE Computer Society.

Uludag, O., Kleehaus, M., Xu, X., Matthes, F. (2017). Investigating the Role of Architects in Scaling Agile Frameworks. 2017 IEEE 21st International Enterprise Distributed Object Computing Conference.

Van den Berg, M., Van Vliet, H. (2016). Enterprise Architects Should Play Different Roles. 2016 IEEE 18th Conference on Business Informatics (CBI). IEEE.