

Aalto-yliopiston teknillinen korkeakoulu
Informaatio- ja luonnontieteiden tiedekunta
Informaatioverkostojen tutkinto-ohjelma

Eevert Saukkokoski

Ketterän ohjelmistokehityksen kasvuhaasteiden hallinta ohjelmistoarkkitehtuurikeskeisillä toiminnoilla

Kandidaatintyö, joka on jätetty opinnäytteenä tarkastettavaksi tekniikan kandidaatin tutkintoa varten

Espoo, 28. marraskuuta 2010

Työn ohjaaja: Tomi Männistö

Sisältö

1	Näkökulmien ristiriidat ketteryyden skaalautuvuuden esteenä	1
2	Ketterä ohjelmistokehitys	2
2.1	Ketteryyden taustat ohjelmistoliiketoiminnan arvonluontiprosessissa .	2
2.2	Ketteryys ideologiana	4
2.3	Ketterissä malleissa käytetyt menetelmät	6
3	Ohjelmistoprojektin kasvu ja kasvun seuraukset	8
3.1	Projektin ja projektiympäristön muuttujat	8
3.2	Projektin kasvu vauhdittaa teknisen riskin kertymistä	9
4	Näkökulmia ohjelmistoarkkitehtuuriin	11
4.1	Arkkitehtuuri osana ohjelmistoa ja sen kehitystä	11
4.2	Arkkitehtuurin käsittely ketterissä menetelmissä	13
5	Perinteisestä ohjelmistoarkkitehtuurista ketterään	15
5.1	Arkkitehtuuri joukkona suunnittelupäätöksiä	16
5.2	Suunnittelupäätökset ja ketteryys	17
6	Ketterä arkkitehti ja arkkitehtuurikeskeiset toiminnot	18
6.1	Ketterä arkkitehti tarjoaa arvokasta palvelua	18
6.2	Arkkitehtuurikeskeiset toiminnot ketterän prosessin aikana	20
6.3	Ketterän arkkitehtuurin kokonaiskuva	21
7	Pohdinta	22
8	Yhteenveto	24
	Viitteet	25

1 Näkökulmien ristiriidat ketteryyden skaalautuvuuden esteenä

Ohjelmistokehitysalalla ketterät menetelmät ovat nykyään laajalti käytettyjä (Falesi et al., 2010) ja ohjelmistoarkkitehtuuri hyväksytty käsite (Jansen & Bosch, 2005; Kruchten et al., 2006). Ketteryys priorisoi jatkuvan arvon tuottamisen asiakkaalle (Abrahamsson et al., 2010) ja ohjelmistoarkkitehtuurikeskeiset menetelmät huomioivat monimutkaisen projektiympäristön vaatimukset sekä niiden heijastumisen ohjelmistossa (Lago et al., 2010). Käytännössä osapuolten näkemykset toisistaan ovat poikenneet lähtökohdiltaan riittävästi estääkseen menetelmien yhdistämisen (Nord & Tomayko, 2006). Tämä on valitettavaa, sillä arkkitehtuurikysymysten ohittaminen voi ketterässäkin ohjelmistokehityksessä johtaa umpikujaan, jossa muutosten tekeminen ohjelmistoon muuttuu hyvin vaikeaksi (Kruchten, 2007). Ketteryyttä on toisaalta pidetty soveltuvana vain tiettyihin projektiympäristöihin (Boehm & Turner, 2003) ja erityisesti pienille kehitysryhmille (Abrahamsson et al., 2002). Tässä tutkimuksessa selvitetään, millä tavoin ohjelmistoarkkitehtuuri voi muodostua ongelmaksi ketterän projektin kasvun myötä ja miten ohjelmistoarkkitehtuurikeskeisiä toimintoja voidaan soveltaa tinkimättä ketteristä arvoista.

Pohjimmiltaan ristiriidassa on kyse suunnitelmallisuutta ja mukautuvuutta kannattavien prosessien eroista (Abrahamsson et al., 2010). Ohjelmistoarkkitehtuurin keinot on liitetty suunnitelmavetoisuuteen, joka ei sovi yhteen suunnittelua välttäviin ketteriin prosesseihin, minkä lisäksi arkkitehtuuria itsessään ei ketteryyden piirissä ole pidetty kriittisenä huolenaiheena (Abrahamsson et al., 2010). Ohjelmistoarkkitehtuuria ja ketteryyttä on pyritty yhdistämään vasta hiljattain - esimerkiksi Abrahamsson et al. (2010), Blair et al. (2010) ja Madison (2010) ovat käsitelleet aihetta, mutta sitä ei toistaiseksi ole laajalti tutkittu.

Tämän tutkimuksen tarkoituksena on etsiä kirjallisuudesta näkökulmia, joiden kannalta ohjelmistoarkkitehtuuri ja ketterät menetelmät on mahdollista yhdistää ja tuottaa ohjeita ja suosituksia, jotka tekevät mahdolliseksi ohjelmistoarkkitehtuurin huomioinnin ketterän prosessin aikana sen ehtoja noudattaen, jatkuvasti ja luonnollisesti prosessiin integroituen. Tutkimus ei käsittele ohjelmistoarkkitehtuurin keinoja esimerkiksi suunnittelun tai esittämisen osalta sikäli kuin se ei ole välttämätöntä tutkimustulosten esittämisen kannalta. Sen sijaan pyritään etsimään joukko ohjelmistoarkkitehtuurikeskeisiä toimintoja, joita ketterissä prosesseissa voidaan soveltaa

ohjelmistoarkkitehtuurinäkökulman huomioimiseksi.

Ohjelmistoarkkitehtuurikeskeisillä toiminnoilla tarkoitetaan tämän tutkimuksen yhteydessä mitä tahansa ohjelmistoprojektin aikana toteutettua tehtävää, jonka pääasiallinen tarkoitus on vaikuttaa ohjelmiston arkkitehtuuriin tai arkkitehtuurillisten päämäärien toteutumiseen. Toimintojen toteutustavat, esimerkiksi keinot perustelujen suunnittelupäätösten saavuttamiseksi, on siis jätetty avoimeksi. Ohjelmistokehitysmallilla tarkoitetaan kuvausta tietyntylaisesta kehitysprosessista ja menetelmällä taas mallin ilmentymää käytännössä, mallin pohjana toiminutta prosessia tai sen osia.

2 Ketterä ohjelmistokehitys

Tässä luvussa esitellään ketterä ohjelmistokehitys ja sen asiakasprojektiliiketoiminnassa saavuttaman suosion syitä. Ketteryyttä käydään läpi sekä konkreettisenä, seurattavana prosessimallina että alla piilevien motivaatioiden kautta. Tarkoitus on korostaa ketteryyden oleellisia osatekijöitä, jotta niiden vaikutusta sekä kasvuhaasteisiin että ohjelmistoarkkitehtuurin soveltamiseen voidaan käsitellä myöhemmin. Esimerkkeinä ketteristä kehitysmalleista käytetään Scrum- (Schwaber, 1995) ja Extreme Programming (Beck & Fowler, 2000) -malleja, jotka yhdessä kattavat kehitysprojehtin koko aktiviteettispektrin projektinhallinnasta ohjelmointiin (Abrahamsson et al., 2002).

2.1 Ketteryyden taustat ohjelmistoliiketoiminnan arvonluontiprosessissa

Tässä kappaleessa esitellään muutamia perusteltuja syitä, joiden ansiosta ketteriin menetelmiin on päädytty. Tarkoitus on osoittaa ketteryyden olevan nykyisellään oikeutetusti oleellinen osa asiakasprojektiliiketoiminnan standardityökaluja.

Suunnittelu- ja kehitystyön tarkoitus yleisesti on tuottaa sijoitukselle lisäarvoa (Boehm & Sullivan, 2000). Ohjelmistokehitysprojekteissa otetun riskin ja saatavan lisäarvon suhde on ajoittain ollut hyvin huono, eikä riskejä ole voitu ennakoita hyväksyttävällä tasolla (Boehm & Sullivan, 2000). Merkittävä osa ohjelmistokehitysprojekteista on jouduttu päättämään ennen niiden valmistumista (Sutherland, 2001). IT-alalle ase-

tetut vaatimukset ovat kasvaneet ja nykyään ohjelmistoprojekteja on vaikea käynnistää, elleivät ne kykene tuottamaan hyvin määriteltyä lisäarvoa nopeasti - sijoituksen kannattavuus laskee sen mukaan, mitä kauempana tulevaisuudessa hyötyä saadaan (Denne & Cleland-Huang, 2005).

Perinteiset, raskaat ohjelmistokehitysmallit olettavat, että ohjelmistolle asetettavat vaatimukset voidaan selvittää etukäteen, että vaatimukset eivät muutu projektin aikana ja että ohjelmistokehitysprosessi on pohjimmiltaan ennakoitava ja toistettavissa (Sutherland, 2001). Perinteisyydellä viitataan mallien tutkittuuteen ja kypsyyteen (Huo et al., 2004). Hallittuun kehitysprosessiin keskittymisen vuoksi malleja voidaan kutsua myös suunnitelmavetoisiksi (Abrahamsson et al., 2002).

Yleisiä merkittäviä tekijöitä projektien epäonnistumisessa ovat olleet epäselvästi kommunikoidut tai tarpeettomat vaatimukset, käyttötarkoitukseen soveltumattomat tai toimimattomat ohjelmistot ja aikatauluihin ja laajuuteen sitoutuminen ennen toteutukseen sisältyvien riskien ymmärtämistä (Lindstrom & Jeffries, 2004). Epäonnistumisriskiä on pyritty hallitsemaan tarkemmilla vaatimusmäärittelyillä, paremmalla suunnittelulla ja toteutuksella ja tehokkaammalla testauksella - eli keskittymällä kuhunkin kehitysprosessin osaan erikseen (Lindstrom & Jeffries, 2004). Perinteisissä malleissa ohjelmisto suunnitellaan, toteutetaan ja testataan kokonaisuudessaan kerralla ja järjestyksessä (Sutherland, 2001; Huo et al., 2004).

Toteutuksen aikana lähtökohdissa voi kuitenkin tapahtua merkittäviä muutoksia (Highsmith & Cockburn, 2002). Jonkinlaisia viitteitä täydellisen etukäteissuunnittelun mahdottomuudesta antavat jo matemaattinen todistus siitä, ettei interaktiivista järjestelmää¹ ole mahdollista määritellä täydellisesti ja osoitus siitä, ettei kaikkia ohjelmiston vaatimuksia voida tietää ennen kuin ohjelmistoa on käytetty (Sutherland, 2001). Abrahamsson et al. (2002) huomauttavat tietyissä tutkimuksissa näytetyn, ettei perinteisiä suunnitelmavetoisia kehitysmalleja todellisuudessa noudateta, mikä on arveltu johtuvan mallien jäykkyydestä sekä muutosten että prosessin itsensä suhteen.

Suunnitelmavetoiset mallit eivät osassa ohjelmistoprojekteista ole tarjonneet hyviä vastauksia ohjelmistokehityksen haasteisiin (Lindstrom & Jeffries, 2004). Näin on erityisesti uusilla ja kasvavilla aloilla, kuten internet- ja mobiilisovelluskehityksessä (Abrahamsson et al., 2002). Ohjelmistokehityksiprojekteilta toivottujen ominaisuuksien

¹Interaktiivisia ovat kaikki ympäristön syötteeseen nojaavat ohjelmistot, mikä käsittää suurimman osan kehitettävistä ohjelmistoista (Sutherland, 2001).

sien, kuten pääomasijoitusten lyhyen keston, ohjelmistotuotteiden nopean julkistussyklin ja toiminnan suuren mukautuvaisuuden, saavuttaminen on vaatinut uudenlaisia ohjelmistokehitysmalleja (Denne & Cleland-Huang, 2005). Ketterät ohjelmistokehitysmallit pyrkivät vastaamaan näihin vaatimuksiin keskittymällä kehitysprosessin keventämiseen ja arvontuoton nopeuttamiseen (Abrahamsson et al., 2002). Huo et al. (2004) esittävät ketterien mallien oleellisimmiksi ominaisuuksiksi kyvyn mukautua myös projektin aikana muuttuviin vaatimuksiin ja toimittaa ohjelmistotuotteita lyhyemmässä ajassa ja pienemmällä budjetilla perinteisiin kehitysmalleihin verrattua.

Ketterien projektien toteuttamisesta tehdyt sopimukset perustuvat usein kehityksessä käytettävään aikaan ja resursseihin etukäteen suunnitellun lopputuotteen sijaan (Paetsch et al., 2003). Riskin ja hinnan hallintaan keskittymisen sijaan arvontuotannon toimitustahtia optimoimalla voidaan saavuttaa verrattomia kilpailuetua (Denne & Cleland-Huang, 2005). Suunnitelmavetoisten mallien yhteydessä asiakkaan luottamus on tarkoitus saavuttaa määrittelemällä tarkkaan etukäteen, mitä ja mihin hintaan kehitystyön lopputuloksena on - ketterissä malleissa luottamusta pyritään luomaan integroimalla asiakas kehitystyöhön (Paetsch et al., 2003), jolloin asiakas voi varmistua tuloksista itse (Highsmith & Cockburn, 2002). Asiakas pääsee tällöin myös määrittelemään, mitkä ohjelmistolle asetettavista vaatimuksista ovat kulloinkin arvokkaimpia (Highsmith & Cockburn, 2002). Arvon määrittely mahdollistaa kehitystyön vaiheiden järjestämisen siten, että alkusijoituskustannukset laskevat ja ohjelmisto voi tuottaa arvoa nopeammin (Denne & Cleland-Huang, 2005).

Nykyään ketterät mallit ovat laajalti käytössä (Falessi et al., 2010), mutta niiden sovellettavuuden rajat alkavat olla tiedossa ja suunnitelmavetoisiakin malleja tunnustetaan tietyissä tilanteissa tarvittavan (Boehm, 2002; Boehm & Turner, 2003).

2.2 Ketteryys ideologiana

Tässä kappaleessa valaistetaan ketterää ohjelmistokehitystä ideologisena kokonaisuutena ja käydään läpi ketterien mallien taustalla olevia arvoja. Ideologisen kontekstin ymmärtäminen on tärkeää, jotta jatkossa voidaan tehdä ketteryyden lähtökohtia noudattavia kehitysehdotuksia.

Manifesto for Agile Software Development (Beck et al., 2001) esitteli ketteryysideologian ohjelmistotalle suuressa mittakaavassa (Abrahamsson et al., 2002). Manifes-

tissa ketterien menetelmien käyttäjistä koostuva allianssi kuvasi lyhyesti yhdistäviä periaatteitaan (Fowler & Highsmith, 2001). Jo manifestin tekijöiden edustamien varsinaisten ketterien prosessimallien lukumäärä osoittaa ideologian sisälle mahtuvan reippaasti vaihtelua: manifesti itse luettelee viisi eri mallia (Fowler & Highsmith, 2001). Insinööritieteen sijaan ketteryydessä on kyse enemminkin vapaamuotoisesti osissa ohjelmistokehitysalaa muodostuneesta yhteisestä kulttuurista, josta on monia toisistaan poikkeavia käsityksiä (Kruchten, 2007).

Abrahamsson et al. (2002) tutkivat ketteriksi väitettyjä malleja ja tiivistävät analyysissään mallien yhteiset tekijät seuraaviksi:

Inkrementaalisuus: Ohjelmistoa toimitetaan jatkuvasti lyhyissä sykleissä.

Yhteistyö asiakkaan kanssa: Kehittäjien ja asiakkaan välinen kommunikaatio on jatkuvaa ja esteetöntä.

Yksinkertaisuus: Menetelmä on helppo oppia, hyvin dokumentoitu ja sitä on helppo muokata omiin tarpeisiin.

Muutoksenteon helppous: Prosessissa ei lukita vaihtoehtoja, joten muutoksia suunnitelmiin voidaan tehdä viime hetkiin saakka.

Suunnitelmavetoisen mallin yhteydessä asiakkaan kanssa on etukäteen tarkoitus sopia tarkkaan toimitettavan kohteen yksityiskohdista, ketterässä mallissa sovitaan enemminkin tulevasta yhteistyöstä (Abrahamsson et al., 2002). Ketteryys tunnistaa jokaisen ohjelmistotoimituksen ainutlaatuisuuden (Abrahamsson et al., 2002): ketterän prosessin käyttäjän pitää ymmärtää, että projekti on jatkuva suoritus, jota voidaan ohjata kulloinkin parhaalta näyttävään suuntaan (Beck & Fowler, 2000). Valmiita täsmällisiä kaavoja kunkin projektin viemiseksi alusta loppuun asti ei ole ja sopivat tavat ketteryyden toteuttamiseen riippuvat vahvasti kontekstista (Abrahamsson et al., 2002).

Merkittävä motiivi ketterien menetelmien kehityksessä on ollut ihmiskeskeisyys (Abrahamsson et al., 2002). Ohjelmistoalalla standardikäytännöksi muodostuneisiin epäinhimillisiin ylityötaakoihin kyllästyneet kehittäjät ovat päätelleet ainaisen kiireen johtuvan epärealistisista olettamuksista ja suunnitelmista ja luoneet oman prosessikehyksensä, jossa itse asiantuntijoilla on merkittävästi valtaa prosessin etenemiseen (Glass, 2001). Vallan halutaan olevan yksilöillä, ei jähmeillä prosesseilla (Abrahamsson et al., 2002). Motiivi näkyy selvästi esimerkiksi Extreme Programming -mallin

yhteydessä käytetyssä retoriikassa, jossa korostetaan kehittäjien hyvinvointia ja jakamista (Beck & Fowler, 2000).

2.3 Ketterissä malleissa käytetyt menetelmät

Ketteryys sisältää laajan joukon erilaisia menetelmiä, eikä yksikään tietty menetelmä välttämättä sisälly kaikkiin malleihin (Kruchten, 2007). Ketterä malli voi käytännössä olla esimerkiksi kokoelma löyhiä nyrkkisääntöjä ja matalan tason menetelmiä, jotka yhdessä muodostavat koherentin kokonaisuuden (Beck & Fowler, 2000), tai harkitumpi kuvaus prosessista, joka jättää kehityksen yksityiskohdat käsittelemättä (Schwaber, 1995). Seuraavaksi esitellään oleellisimpia ketterissä malleissa käytettyjä menetelmiä ja niiden roolia kehitysprosessissa.

Ketteryys pyrkii vähentämään etukäteissuunnittelun merkitystä ja lisäämään yhteistyötä asiakkaan kanssa (Abrahamsson et al., 2002). Asiakasta edustavan tahon pitää olla läsnä prosessin aikana pitämässä huolta projektin tavoitteiden täyttymisestä, ohjaamassa kehitystyötä oikeaan suuntaan ja tekemässä täsmennyksiä vaadittujen toiminnallisuuksien yksityiskohtiin (Beck & Fowler, 2000). Tämän henkilön pitää olla todellisuudessa kykenevä tekemään päätöksiä etenemissuuntasta ja käyttämään yhteistyöhön kehittäjien kanssa riittävästi aikaa ja vaivaa (Abrahamsson et al., 2002; Boehm & Turner, 2003).

Toimimatonta sovellusta asiakas ei voi havainnoida eikä käyttää apuna päätöksenteossa, joten kehittämisen on aika ajoin pysähdyttävä (Beck & Fowler, 2000). Ketterissä projekteissa kehitys etenee lyhyehköissä iteraatioissa eli sprinteissä (Schwaber, 1995), joiden lopussa etenemisen pitää olla nähtävissä (Beck & Fowler, 2000). Edistyksen mittana pidetään toimivaa sovellusta (Abrahamsson et al., 2002).

Jokaisen sprintin tavoitteet määritellään, jotta saavutetaan yhteinen näkemys siitä, mitä sprintin aikana pitäisi tapahtua (Schwaber, 1995). Koska sprintit ovat lyhyitä, ei määrittely ja suunnittelu voi viedä merkittävästi aikaa (Beck & Fowler, 2000). Yleensä tarpeen on vain halutun ominaisuusjoukon dokumentoiminen, joka voidaan tehdä kevyillä menetelmillä - esimerkiksi tarinankerronnalla (Beck & Fowler, 2000). Toiminnallisuuksien tarkemmat yksityiskohdat käyvät ilmi sprintin aikana kehittäjien ja asiakkaan keskinäisen kommunikaation kautta (Beck & Fowler, 2000). Sprintin aikana voidaan yhtäaikaisesti tehdä monia tehtäviä, kuten kehitystyötä, aiempien tulosten arviointia ja uusien ominaisuuksien määrittelyä (Schwaber, 1995).

Sprintin määrittäessä tehtävien yhteiskeston pitää tehtävät mitoittaa, jotta sopivan laajuinen joukko tehtäviä voidaan valita toteutettavaksi. Lista halutuista toiminnallisuuksista luodaan asiakkaan kanssa, mutta paras tieto toiminnallisuuksien toteuttamiseen vaadittavasta ajasta on kehittäjillä itsellään (Beck & Fowler, 2000). Asiakasta voidaan toki käyttää apuna riittävien lisätietojen keräämiseen, mutta muut kuin kehittäjät itse eivät voi suoraan vaikuttaa arvioon tehtävien todellisesta kestopista. Edes asiakas ei voi pakottaa kehittäjiä toteuttamaan tehtävää yhdessä sprintissä, ellei tehtävä siihen mahdu. Tämä vähentää kehittäjiin kohdistuvia epärealistisia odotuksia (Glass, 2001). Arvioiden pienen mittakaavan ansiosta riski merkittävien arviointivirheiden tekemiseen laskee - ja mikäli virheitä tapahtuu, ne voidaan havaita ja korjata lyhyiden iteraatioiden ansiosta nopeasti (Beck & Fowler, 2000). Lyhyemmät sprintit vähentävät projektia koskevan epävarmuuden vaikutusta sallimalla tulosten tarkastelun useammin (Cohn et al., 2006).

Sprintin tarkoitus on tuottaa asiakkaalle mahdollisimman arvokkaita ominaisuuksia, joten tehtävien keston arvioinnin jälkeen asiakkaan tehtäväksi jää valita toteutettavaksi tehtävälialta kaikkein arvokkaimmiksi koetut tehtävät (Beck & Fowler, 2000). Tässä vaiheessa voidaan ottaa huomioon myös edellisen sprintin aikana saatu uusi tieto, kuten esimerkiksi havainnot tehdyistä arviointivirheistä tai paljastuneet uudet vaatimukset: tehtäviä voidaan lisätä tai poistaa ja niiden tärkeysjärjestystä voidaan muuttaa (Highsmith & Cockburn, 2002).

Vähäisen dokumentoinnin ansiosta projektia koskevan tiedon on suurimmaksi osaksi välityttävä muulla tavoin. Eksplisiittisen tiedonvälityksen sijaan välitetään projektin jäsenillä olevaa hiljaista tietoa sosialisoinnin avulla (ks. tiedonvälityksen SECI-malli, Nonaka & Konno 1998). Sosialisointi on yhteisessä tiedonjaon tilassa olemista; jotta tiedonkululle olisi mahdollisimman vähän esteitä, tulisi sekä kehittäjien että asiakkaan sijaita fyysisesti samassa tilassa (Beck & Fowler, 2000; Boehm & Turner, 2003). Tarkoituksena on madaltaa kynnyksiä viestimiselle - on vaikeampaa lähettää sähköposti kuin esittää kysymys vierustoverille. Kenties ikävänä sivuvaikutuksena ketteryys on soveltuvimmin pienille, alle kymmenen kehittäjän ryhmille (Abrahamsson et al., 2002).

Ketteryydestä muodostuva kuva vaikuttaa ehkä epämääräisyydessään kaoottiselta - ja niin se tietysti mielessä onkin. Schwaber (1995) väittää Scrum-mallissa, että kaaosta lähestyvä joustavuus kehitysprosessissa parantaa monimutkaisten järjestelmien kilpailukykyä. Yllä esitetyn tiivistelmänä ketteryys on kaoottinen viitekehys, jonka

avulla pieni, asiantunteva ja omistautunut kehitysryhmä kykenee toimittamaan jatkuvaa tulosta myös muuttuvien olosuhteiden alaisuudessa (Boehm & Turner, 2003; Kettunen, 2009). Tarkastellaan seuraavaksi, millaisia ongelmia ketterän ohjelmistokehitysprojektin aikana voidaan kohdata.

3 Ohjelmistoprojektin kasvu ja kasvun seuraukset

Tässä luvussa käydään läpi projektien ominaisuuksia ja niiden toimintaympäristöä sekä reflektoidaan ympäristön vaikutuksia kehitysprosessiin. Lisäksi tuodaan esille riskejä, joita kohdataan projektin kasvaessa. Lähtökohdaksi oletetaan ketteryydelle sopiva projektiympäristö. Edellisen luvun perusteella voidaan todeta, että ketterät menetelmät ovat lähtökohtaisesti mukautuvia, ja muuttuvissa projektiolosuhteissa niiden on tarkoituskin sopeutua. Tarkoituksena on osoittaa käytännön ongelmakohtia, joihin voidaan jatkossa yrittää puuttua ohjelmistoarkkitehtuurin keinoin.

3.1 Projektin ja projektiympäristön muuttujat

Tässä kappaleessa määritellään ketterien projektien näkemyksiä projekteissa vaikuttavista muuttujista ja esitellään projektiympäristön seikkoja, jotka lähtökohtaisesti vaikuttavat prosessimallien soveltuvuuteen ja valikoitumiseen.

Extreme Programming määrittelee ohjelmistoprojektin neljällä muuttujalla: laajuus, aika, hinta ja laatu (Beck & Fowler, 2000). Laajuus määrittää, kuinka paljon toimintoja projekti sisältää. Aika määrää projektiin käytettävissä olevan kalenteriajan. Hinta kertoo, mitä ihmis-, ohjelmisto- ja laitteistoresursseja projektiin voidaan käyttää. Laatu kuvaa kehityksen tuloksia ja jakautuu kahteen puoliskoon: miten asiakas kokee tuotteen ja miten se näyttäytyy kehittäjille. Scrum-mallin (Schwaber, 1995) lähtökohta on liiketoiminnallisempi. Asiakkaan vaatimukset, käytettävissä olevat resurssit ja laatu voidaan yhdistää suoraan edellä mainittuihin ominaisuuksiin. Aika on kuitenkin määritelty kilpailuetumuodossa: miten nopeaa toimintaa tarvitaan kilpailuedun saavuttamiseksi, ja mitä sovellukselta vaaditaan kilpailijoiden lyömiseksi?

Käytetty prosessimalli tekee aina tiettyjä oletuksia kehitysprosessin kontekstista (Boehm & Turner, 2003). Ketterillä ja suunnitelmavetoisilla prosessimalleilla on projektin laadusta riippuvat soveltuvuusalueet: mitä pienempi ja vähemmän kriittinen projekti,

mitä dynaamisemmassa ympäristössä ja vapaammassa yrityskulttuurissa toimitaan ja mitä suurempi osa kehittäjistä on asiantuntijoita, sitä paremmin ketteryys sopii (Boehm & Turner, 2003). Jos kuitenkin lähdetään oletuksesta, että toimitaan ketterille menetelmille soveltuvassa ympäristössä, millaisia muutoksia projektissa ja sen ympäristössä voi todennäköisimmin tapahtua?

Scrum-mallin aikakäsite tarjoaa tavan lähestyä muutosta. Projektin muuttujat ovat toisistaan riippuvaisia (Beck & Fowler, 2000): mikäli sovellukselle asetettuja vaatimuksia on paljon ja ne pitää toteuttaa kilpailutilanteen vuoksi nopeasti, on muiden muuttujien vastattava tilannetta. Käytettävät resurssit määräävät projektin hinnan (Schwaber, 1995) ja kehittäjien määrän lisääminen kasvattaa projektin kehitysnopeutta sillä rajoituksella, ettei muutos ole lineaarinen eikä näy heti (Beck & Fowler, 2000). Myös ohjelmiston laajuus ylittää on suoraan yhteydessä tarvittavien kehittäjien määrään (Boehm & Turner, 2003). Mikäli kehityksen aikana ohjelmistoa laajennetaan tai kehitysnopeutta joudutaan kasvattamaan, voidaan siis päätyä tilanteeseen, jossa tarvitaan alkuperäistä enemmän kehittäjiä.

Jatkossa projektin kasvulla viitataan aiemman nojalla joko kehitysympäristössä syntyvien paineiden tai projektin luonnollisen evoluution aikaansaamaan lisääntyneeseen määrään kehittäjiä.

3.2 Projektin kasvu vauhdittaa teknisen riskin kertymistä

Seuraavaksi käsitellään projektin kasvun vaikutuksia ketteriin menetelmiin. Lisäksi ketteryyden pyrkimys tuottaa jatkuvasti arvokkaita toiminnallisuuksia nostetaan esille mahdollisena teknisen riskin lähteenä.

Ketterien menetelmien skaalaaminen kehittäjämäärän suhteen on koettu haastavaksi (Abrahamsson et al., 2002; Boehm & Turner, 2003), vaikka vastakkaisiakin näkemyksiä on esitetty (Sutherland, 2001). Haasteiden syiksi esitetään kommunikoinnin tarpeen lisääntymistä (Beck & Fowler, 2000) ja ketteryydelle lähtökohtaista hiljaisen tiedon sosialisointiin nojaamista (Boehm & Turner, 2003). Ketterät menetelmät nojaavat ajoittain eksplisiittisesti oletukseen kehitysryhmän pienuudesta. Esimerkiksi Extreme Programming ja Scrum soveltavat molemmat päivittäisiä pikapalavereja, joissa jaetaan oleellisin tieto kehitystyön tilasta (Schwaber, 1995; Beck & Fowler, 2000). Menetelmästä tunnustetaan, että tapaamisten on toimiakseen oltava hyvin lyhyitä (Beck & Fowler, 2000). Rajallisessa ajassa voi kuitenkin olla vain rajallinen

määrä puhujia, joten yhden tapaamisen puhujamäärä on väistämättä rajoitettu.

Kehittäjä määrän kasvu johtaa siis vaikeuksiin projektin sisäisessä tiedonvälityksessä. Onnistunut lopputulos ketterässä prosessissa vaatii lisäksi suhteellisesti suurta määrää asiantuntijoita (Boehm & Turner, 2003), jolloin mahdolliset vaikeudet uusien asiantuntijoiden löytämisessä voivat kasvutilanteessa kasvattaa epäonnistumisriskiä.

Ohjelmiston laadulla on kaksi eri puolta: ulkoinen ja sisäinen (Beck & Fowler, 2000). Ulkoinen laatu kuvastaa asiakkaan saamaa vaikutelmaa ohjelmistosta, sen virheettömyyttä ja käyttökokemusta. Sisäisellä laadulla tarkoitetaan ohjelmiston kehittäjille näkyvää puolta eli esimerkiksi sitä, miten se on sisäisesti suunniteltu. Kehitysnopeutta voidaan pyrkiä maksimoimaan tilapäisesti ohjelmiston sisäisen laadun kustannuksella, mikä voi johtaa hyvin nopeasti kehitystyön jämähtämiseen (Beck & Fowler, 2000). Ketterissä menetelmissä sovelluksen toiminnallisuuteen keskittyminen ja bisnesarvon jahtaaminen saattaa projektin kasvaessa ja muuttuviin vaatimuksiin mukauduttaessa johtaa arkkitehturaalisen ajattelun puutteen myötä teknisen riskin kertymiseen (Kruchten, 2007). Riskillä viitataan mahdollisuuteen, että ohjelmiston ulkoinen julkaisu epäonnistuu eli viivästyy tai estyy (Beck & Fowler, 2000). Projektiympäristön paineet kehitysnopeuden kasvattamiselle voivat siis kerryttää teknistä riskiä.

Hiljaisen tiedon välittymisen vaikeutuminen voi johtaa vakaviin ohjelmistoarkkitehtuurivirheisiin (Boehm, 2002). Suunnitelmakeskeisillä toimilla tiedonvälitysvirheiden mahdollisuutta voidaan vähentää sillä uhalla, että muuttuvissa olosuhteissa suunnitelmat vanhentuvat nopeasti ja niitä on kallis ylläpitää (Boehm, 2002). Tällöin hävitään ketteryydessä. Bisnesarvon saavuttamisen ja teknisen riskin välttämisen välillä pitäisi olla tasapaino (Fowler, 2004), joka on saavutettavissa ketterinkin keinoin (Madison, 2010).

Edellä olevan perusteella projektin kasvu lisää ketterän projektin teknisten riskien kertymistä muun muassa ohjelmistoarkkitehtuurin laiminlyönnin takia. Vaikuttaa siltä, että lisäämällä ohjelmistoarkkitehtuuriseikkoihin kohdistettua huomiota ketterissä malleissa voitaisiin hillitä teknisiä riskejä. Tutustutaan seuraavaksi arkkitehtuuriin ja siitä vallitseviin käsityksiin.

4 Näkökulmia ohjelmistoarkkitehtuuriin

Tässä luvussa käsitellään ohjelmistoarkkitehtuurin taustoja sekä osuutta ohjelmistokehityksessä ja kehityksen tuloksena syntyvässä järjestelmässä. Lisäksi tutkitaan ketterien menetelmien ja ohjelmistoarkkitehtuurin välistä suhdetta. Tarkoituksena on muodostaa pohja ohjelmistoarkkitehtuurin myöhemmälle käsittelylle ja ketteriin menetelmiin yhdistämiselle vastauksena edellisessä luvussa esitettyihin haasteisiin.

4.1 Arkkitehtuuri osana ohjelmistoa ja sen kehitystä

Seuraavaksi esitellään arkkitehtuuri tuotettujen ohjelmistojärjestelmien ominaisuutena ja ohjelmistoarkkitehtuuri alana: mistä se on lähtöisin ja mitä se sisältää? Lisäksi käsitellään arkkitehdin roolia perinteisessä projektioorganisaatiossa.

Arkkitehtuurilla viitattiin tietokoneiden yhteydessä pitkään tietokoneen fyysiseen rakenteeseen (Kruchten et al., 2006). Vasta 90-luvulla muodostui ohjelmistoarkkitehtuurin käsite: “Software Architecture = {Elements, Forms, Rationale}” (Perry & Wolf, 1992). Määritelmään sisältyivät siis järjestelmän osat, niiden muodostamat kokonaisuudet ja tehtyjen valintojen perustelut. Kuvaavaa on, että määritelmään lisättiin oitis vielä “constraints” eli valintoja ohjanneet rajoitukset - ohjelmistoarkkitehtuuria on ollut hyvin vaikea määritellä kokonaisuudessaan tyydyttävästi (Kruchten et al., 2006). Rozanski & Woods (2005) antavat kuitenkin käyttökelpoisen määritelmän:

Ohjelmistointensiivisen järjestelmän arkkitehtuuri on järjestelmän rakenne tai rakenteet, jotka koostuvat ohjelmistoelementeistä, elementtien ulkoisista ominaisuuksista ja elementtien välisistä suhteista.

Arkkitehtuurissa on oleellisimmillaan kyse järjestelmän rakenteesta. Rakennetta on arkkitehtuurin alalla pyritty tarkastelemaan korkeimmalla käsitetasolla (Ellis et al., 2002), mutta mahdottomuus erottaa järjestelmästä yksiselitteinen korkeimman tason käsitteistö on johtanut näkemykseen arkkitehtuurista ennemmin järjestelmän tuntijoiden keskeisenä yhteisymmärryksenä tärkeistä rakenteellisista piirteistä kuin tarkkarajaisena entiteettinä (Fowler, 2003). Kenties määritelmää oleellisempänä voidaan pitää kysymystä siitä, mitä tehtäviä arkkitehtuurilla on ja miten tehtävien toteutumisesta on pyritty varmistumaan.

Arkkitehtuuri eli ohjelmiston rakenne on ominaisuus, joka ei suoraan näy toiminnallisuudessa, mutta jota voidaan käyttää vastaamaan erilaisiin sidosryhmien vaatimuksiin (Lago et al., 2010). Sidoryhmiin lukeutuu myös asiakas. Arkkitehtuuri toimii koko järjestelmän kehittämisen pohjana (Losavio et al., 2003) sekä mahdollistaa uudelleenkäytettävyyden ja muunneltavuuden kaltaisten seikkojen huomioinnin ja toteutumisen (Lundberg et al., 1999). Jokaisessa sovelluksessa on olemassa jonkinlainen arkkitehtuuri (Ellis et al., 2002). Näin on riippumatta siitä, onko arkkitehtuuri syntynyt suunnitellusti vaiko - erityisesti ketterien menetelmien tapauksessa (Abrahamsson et al., 2010) - täysin emergentisti järjestelmän toteutuksen aikana. Kaikki toiminnallisuus järjestelmässä toteutetaan sen arkkitehtuurin asettamissa puitteissa (Nord & Tomayko, 2006).

Arkkitehtuurin avulla voidaan mallintaa järjestelmän rakennetta ja käytöstä, puolustaa rakennetta tahattomalta heikentymiseltä ja lisätä monimutkaisten järjestelmien hallittavuutta (Kruchten et al., 2006). Monimutkaisten järjestelmien hallittavuuden lisäämistä voidaan pitää arkkitehtuurin tehtävistä tärkeimpinä (Lago et al., 2010). Arkkitehtuuria voidaan tutkia eri näkökulmista, jotka *International Federation of Information Processing Working Group 2.10* on määritellyt seuraavasti (Kruchten et al., 2006):

Suunnittelu: Miten arkkitehtuuri tuotetaan?

Analyysi: Mitä järjestelmästä voidaan kertoa sen arkkitehtuurin perusteella?

Realisointi: Miten järjestelmä toteutetaan arkkitehtuurikuvauksen avulla?

Esittäminen: Miten tuotetaan arkkitehtuurin esitysmuotoja arkkitehtuuria koskevan viestinnän mahdollistamiseksi?

Talous: Miten arkkitehtuuriseikat vaikuttavat bisnespäätöksiin?

Kehitystyön onnistumisen kannalta on pidetty erittäin suositeltavana, että projektissa on mukana ohjelmistoarkkitehti, jolla on kokonaiskuva järjestelmästä (Abowd et al., 1997). Vaikka ohjelmistoarkkitehdin tehtävää on tärkeydessään verrattu projektimanageriin (Abowd et al., 1997), on tehtävää osittain pidetty itseselitteisenä ja avoimeksi jäänyt tehtävän määrittely siten ollut arkkitehtien itsensä vastuulla (Gore, 2003). Esimerkiksi arkkitehtuurikeskeisessä RUP-mallissa arkkitehti on kuitenkin oleellisesti järjestelmän toteutuksen pohjana toimivan arkkitehtuurin suunnittelija ja arkkitehtuuria kuvaavan dokumentaation luoja (Kruchten, 2004). Arkkitehtuuri-

suunnittelussa järjestelmä on tarkoitus jakaa ja järjestää osiin siten, että kukin osa vastaa tiettyihin vaatimuksiin - monimutkaisetkin haasteet voidaan ratkoa jakamalla järjestelmä osiin sopivalla tavalla (Lago et al., 2010). Arkkitehtuuria kuvataan yleisesti näkymien avulla (engl. *view*) (Lago et al., 2010), jotka heijastavat järjestelmän jakoa eri vaatimuksiin vastaaviin osiin (Clements et al., 2002; Ellis et al., 2002).

Arkkitehtuurin luominen perustuu järjestelmälle asetettuihin vaatimuksiin vastaamiseen (Abrahamsson et al., 2010). Vaatimukset kumpuavat järjestelmän sidosryhmien tarpeista, järjestelmän toimintaympäristöstä ja realiteeteista ja rajoitteista järjestelmän kehittämisessä, toteutuksessa, ylläpidossa ja käytössä sekä järjestelmän itsensä ominaisuuksissa (Lago et al., 2010). Luotettavuuden, ylläpidettävyyden ja tehokkuuden kaltaisia, järjestelmälle asetettuja laadullisia vaatimuksia toteuttavia arkkitehtuurisia piirteitä voidaan kutsua laatuattribuuteiksi (Losavio et al., 2003).

Tässä kappaleessa esitettyä kuvaa arkkitehtuurista ja arkkitehdin roolista tarkastellessa voidaan havaita, että kyseessä on ollut etupäässä suunnitelmavetoisiin menetelmiin liittyvä käsite: ensin on suunniteltu arkkitehtuuri toteuttamaan asetetut vaatimukset ja sen jälkeen järjestelmä on toteutettu arkkitehtuurin puitteissa. Jo tämän havainnon pohjalta voidaan arvella, että ohjelmistoarkkitehtuurin ja ketteryyden menetelmien suhde ei ole aivan ongelmaton. Tarkastellaan seuraavaksi arkkitehtuuria ketterästä näkökulmasta.

4.2 Arkkitehtuurin käsittely ketterissä menetelmissä

Tässä kappaleessa valaistetaan ketteryyden piirissä vallitsevia käsityksiä arkkitehtuurista ja käsitysten perusteita. Lisäksi esitetään, että arkkitehtuurikeskeisen ajattelun lisääminen ketterissä menetelmissä voisi olla mahdollista ja jopa hyödyllistä.

Arkkitehtuuri voidaan nähdä rakenteina, jotka on pakko tehdä etukäteen - tai joista toivottaisiin saatavan selvyys mahdollisimman aikaisin (Fowler, 2003). Ketterien menetelmien kannattajille arkkitehtuuri merkitsee monesti mittavaa etukäteissuunnittelua dokumentaatioineen (Nord & Tomayko, 2006) ja ominaisuuksia, joita ei todella tarvita (Abrahamsson et al., 2010). Fowlerin (2003) mukaan ohjelmistoarkkitehtuuri eroaa rakennusarkkitehtuurista siinä, ettei ohjelmiston arkkitehtuurissa tehtyjen yksittäisten muutosten ole pakko olla hyvin vaikeita. Ei kuitenkaan vielä osata tehdä arkkitehtuuria, jossa kaikkien mahdollisten muutosten tekeminen olisi helppoa.

Extreme Programming, eräs kypsimmistä ketteristä malleista (Nord & Tomayko, 2006), sisältää käsityksen arkkitehtuurin syntymisestä parhaiten suhtautumalla siihen kuin mihin tahansa ohjelmoinnin ohella tapahtuvaan suunnitteluun ja tuottamalla se toiminnallisuuksien mukana (Fowler, 2004; Blair et al., 2010). Ohjelmaelementtejä, joita eivät suoraan motivoi asiakkaan nykyiset tarpeet, ei tämän ajatusmaailman mukaan kaivata lainkaan. Rakennetta ei suunnitella vastaamaan kaikkiin ennakoituihin tarpeisiin, vaan kasvatetaan ja muokataan jatkuvasti, mikä saattaa toisaalta muodostua ratkaisevaksi heikkoudeksi - oletus, jonka mukaan ohjelmiston rakenne voidaan aina muokata kulloisiakin tarpeita vaativaksi, ei välttämättä pidä paikkaansa (Fowler, 2004).

Arkkitehtuurinäkömät ovat ketterässä projektissa liian raskas dokumentaatiomuoto (Tyree & Akerman, 2005). Näkömien sijaan arkkitehtuurin kuvaamisen käytetään esimerkiksi yksinkertaisia järjestelmämetaforia, joiden on tarkoitus riittää jakamaan projektin visio sen sidosryhmille (Huo et al., 2004).

Käyttäjätarinoiden laatimisessa keskitytään lähtökohtaisesti järjestelmän toiminnallisuuteen, sillä toiminnon suorittaa oletusarvoisesti asiakas (Beck & Fowler, 2000). Arkkitehtuuriseikat unohtuvat usein, sillä niiden huomiointi ei varsinaisesti ole osa kehitysprosessia (Nord & Tomayko, 2006). Rakenteen parantaminen ei välittömästi näy uusina toiminnallisuuksina, joten tehtävä on helppo ohittaa. Kehittäjät saattavat suhtautua erillisestä arkkitehdistä saatavaan hyötyyn skeptisesti ja toisaalta olla itse liian kiireisiä miettimään arkkitehtuuriseikkoja - bisnesprioriteettien nojalla on helppo perustella hyvän arkkitehtuurin tekemättä jättämistä (Madison, 2010).

Arkkitehtuurin roolin vähättely ketteryyden parissa saattaa johtua etukäteissuunnittelun vierastamisen lisäksi osittain myös arkkitehtuurikeskeisen projektiorganisaation järjestäytymisestä. Projektiorganisaatiossa arkkitehdin paikka on usein saatanut olla sidosryhmien ja toteuttajien välissä (Blair et al., 2010). Arkkitehti on laatinut sidosryhmien kanssa vaatimusmäärittelyt, luonut näiden perusteella suunnitteludokumentin ja välittänyt sen kehittäjille. Arkkitehti on siis ollut mahdollinen lähde samantyyppisille muihin kehittäjiin kohdistuville epärealistisille vaatimuksille kuin mitä luvussa 2 liitettiin suunnitelmavetoisiin menetelmiin (Blair et al., 2010). Ketterien periaatteiden mukaisesti taas kommunikoinnin asiakkaan kanssa pitäisi olla mahdollisimman vapaata, kuten luvussa 2 todettiin. Ketterissä malleissa arkkitehdin rooli on ollut huonosti määritelty tai olematon (Blair et al., 2010). Hyvä ensiaskel arkkitehtuurikeskeisyyden lisäämisessä voisi olla arkkitehdin määrittely ketteryyteen

sopivalla tavalla.

Falessi et al. (2010) totesivat kokeneita IBM:n kehittäjiä tutkittuaan, että useimpia arkkitehtuurin tehtäviä pidetään tärkeinä myös ketterissä projekteissa. Arkkitehtuuriin kiinnitetään eniten huomiota projektin monimutkaisuuden kasvaessa - missä monimutkaisuuden mittarina on pidetty vaatimusten tai koodirivien määrää sekä sidosryhmien määrää tai hajanaisuutta. Tärkeimpinä arkkitehtuurin tehtävinä pidettiin sidosryhmien kanssa kommunikointia, tarkemman systemisuunnittelun tai kehityksen auttamista sekä järjestelmän käyttötarkoituksesta ja ympäristöstä tehtyjen oletuksien dokumentointia. Arkkitehtuuri oli vastanneille keino kasvaneiden projektien monimutkaisuuden hallitsemiseen. Huomionarvoista tutkimuksessa on, että vaikka osa vastaajista piti arkkitehtuurillisia periaatteita ketterän ideologian vastaisena, pessimistisimpiä tässä suhteessa olivat juuri ei-ketterät kehittäjät. Toivoa näiden kahden näkökulman yhdistämiselle siis on erityisesti silloin, kun lähtökohtana on ketterä kehitysprosessi.

Edellisen perusteella arkkitehtuuria ei pidetä ketterissä malleissa tärkeänä huolenaiheena ja arkkitehtuurin menetelmät eivät sovi sellaisenaan ketterien menetelmien yhteyteen. Olisi kuitenkin mahdollista kiinnittää enemmän huomiota arkkitehtuuriin, mikäli keksittäisiin keinoja, jotka eivät vaadi laajaa ennakkosuunnittelua, raskaita dokumentteja tai estä projektin aikaista kommunikointia.

5 Perinteisestä ohjelmistoarkkitehtuurista ketterään

Abrahamsson et al. (2010) ovat määritelleet joukon tekijöitä, jotka täytyy määritellä ohjelmistoarkkitehtuurin ja ketterien mallien välisen ristiriidan selvittämiseksi. Tekijöitä ovat semantiikka, laajuus, elinkaari, toimijat, dokumentointi, toteutustavat ja arvon muodostuminen. Kokonaisvaltaisen ratkaisun pitää siis määritellä mitä arkkitehtuurilla tarkoitetaan, miten paljon arkkitehtuuria on tehtävä, milloin sitä tehdään, kuka tekee, miten arkkitehtuuria demonstroidaan ulkoisesti, miten arkkitehtuurin laatiminen etenee ja miten toimintojen synnyttämää arvoa viestitään. Kaikki nämä tekijät pyritään ottamaan huomioon ehdotettaessa ratkaisuja ja näkökulmia tässä ja seuraavassa luvussa. Seuraavaksi pohjustetaan arkkitehtuurin käsittelyä ketteryydessä ideologisella tasolla.

5.1 Arkkitehtuuri joukkona suunnittelupäätöksiä

Arkkitehtuurin käsittelyssä on hiljattain tapahtunut muutos rakenteellisesta kuvauksesta kohti laajempaa näkemystä, joka tarkastelee arkkitehtuurista tietoa yksittäisistä suunnittelupäätöksistä muodostuvana kokonaisuutena (Lago et al., 2010). Perinteisillä arkkitehtuurikeskeisillä menetelmillä² on vaikeuksia viestiä useita arkkitehtuuriin liittyviä seikkoja, kuten tehtyjä muutoksia, päätösten seurauksia ja hylättyjä vaihtoehtoja (Tyree & Akerman, 2005). Tieto näistä sisältyy vain implisiittisesti arkkitehtuurikuvaukseen, mikä johtaa lopulta tiedon katoamiseen ja arkkitehtuurin muutoskustannusten kasvuun, turhaan monimutkaisuuteen ja heikentymiseen jatkuvan kehityksen aikana (Jansen & Bosch, 2005).

Bosch (2004) esittää aiemman arkkitehtuurinäkemysongelmiksi suunnittelupäätöksen käsitteen puuttumisen, tehtyjen suunnittelupäätösten yhteenpunoutumisen, muutostentien vaivalloisuuden, aiemmin tehtyjen suunnittelusääntöjen rikkomisen ja kyvyttömyyden vanhentuneiden suunnittelupäätösten poistamiseen. Arkkitehtuurista on vaikea tai mahdoton päätellä, miksi se on luotu juuri tietynlaiseksi, mikäli päätöksiä ei tunneta. Päätökset vaikuttavat useimmiten useisiin ohjelmiston osiin kerralla, jolloin yksi päätös realisoituu monessa osassa järjestelmää ja arkkitehtuurin muokkaaminen yhden päätöksen osalta jälkeen päin on hyvin vaikeaa. Suunnittelupäätösten implisiittinenkin esitys hajaantuu ja hukkuu suunnitelmaan. Tämä johtaa järjestelmän kehityksen myötä aiemmin tehtyjen päätösten rikkomiseen ja toisaalta mahdottomuuteen poistaa suunnittelupäätöksiä, jotka ovat muodostuneet tarpeettomiksi. Järjestelmän sisäinen laatu heikkenee ja sen muokkaaminen muuttuu yhä vaikeammaksi, kunnes järjestelmä saatetaan lopulta todeta ennenaikaisesti käyttökelvottomaksi.

Arkkitehtuurin ymmärtämisellä suunnittelupäätösjoukkoina Jansen & Bosch (2005) väittävät olevan useita etuja, mille he esittävät myös alustavaa tukea tapaustutkimusten muodossa. Arkkitehtuurin konseptuaalinen integriteetti säilyy paremmin, ilmeisiltä suunnitteluvirheiltä on helpompi välttyä, arkkitehtuurin sekä tehtyjen päätösten analysointi helpottuu ja suunnittelupäätösten suhde järjestelmään ja toisiinsa on helpommin tarkasteltavissa. Oleellisesti järjestelmän evoluutiosta muodostuu tällöin pakollinen osa arkkitehtuurin kuvausta. Mikäli arkkitehtuuri muuttuu, on järjestelmän toteutuksen muututtava vastaamaan tätä - jolloin muutoksen itsensä

²Tyree & Akerman (2005) antavat esimerkkeinä Reference Model for Open Distributed Processing (Raymond, 1995), 4+1 (Kruchten, 1995) ja Rational Unified Process (Kruchten, 2004) -mallit.

dokumentointi auttaa järjestelmän toteutuksessa.

Uudella näkökulmalla on merkittäviä implikaatioita ketterien menetelmien suhteen. Monien yksittäisten päätösten kautta tarkasteltuna arkkitehtuuri muuttuu jo käsitetasolla yhdestä suuresta entiteetistä pienistä osista muodostuvaksi, mikä lisää mahdollisuuksia arkkitehtuurin käsittelyyn ketterissä menetelmissä. Arkkitehtuurin rakentamisesta muodostuu kertaluontoisen tehtävän sijaan prosessi, jota on mahdollista toteuttaa jatkuvasti. Tutkitaan seuraavaksi, miten saatu kuva arkkitehtuurista voidaan sovittaa ketteryteen.

5.2 Suunnittelupäätökset ja ketteryys

Tässä kappaleessa tarkastellaan arkkitehtuurikeskeisiä menetelmiä ja puetaan niiden periaatteet muotoon, joka on yhteensopiva ketteryden kanssa.

Blair et al. (2010) havaitsevat, että myös ketterien menetelmien yhteydessä tehdään suunnittelua. Kappaleessa 5.1 todettiin, että arkkitehtuuri muodostuu arkkitehturaalisista suunnittelupäätöksistä. Arkkitehtuurikeskeisten toimintojen lisäämisessä ketteriin menetelmiin on siis kyse vain päätöstyön järjestelmällistämistä. Blair et al. (2010) soveltavat reaalioptioanalyysia³ ohjelmistotuotantoon ja päättelevät, että päätöksenteon viivyttäminen myöhäisimpään mahdolliseen hetkeen sallii myös suunnittelupäätöksen tekemisen enemmillä tiedolla ja siten todennäköisemmin oikein. Päätelmä oleellisesti mahdollistaa ketterän ja arkkitehtuurillisen ideologian yhdistämisen. Olennaiseksi nostetaan kysymys siitä, milloin on viimeinen vastuullinen hetki tehdä päätös.

Kappaleessa 4.1 määriteltiin arkkitehtuuri vastauksena sidosryhmien vaatimuksiin. Blair et al. (2010) mukaan perinteisesti sovelluksen osoittaminen arkkitehturaalisia laatuvaatimuksia vastaavaksi on tehty etukäteissuunnittelulla ja suunnitelmien dokumentoinnilla. Lähestymistapa on ketterän kehityksen periaatteiden vastainen, mutta Blair et al. (2010) esittävät, ettei sidosryhmiä todellisuudessa kiinnosta *milloin* suunnittelupäätökset tehdään ja *millainen* vaatimukset kattava ratkaisu on, kunhan vaatimuksista lopuksi on pidetty huolta. Madison (2010) esittää, että ketterän kehityksen tarjoama nopea iteraatio mahdollistaa parempien ratkaisujen löytämisen myöhemmin siinä missä niitä ei olisi voitu lainkaan ymmärtää aikaisemmin. Tällöin

³Lähestymistapa on mielenkiintoinen, mutta liian syvälinen käsiteltäväksi yksityiskohtaisemmin tässä yhteydessä. Esimerkiksi Sullivan et al. (1999) perehtyvät aiheeseen tarkemmin.

luotetaan siihen, että projektin edetessä paras kehityssuunta tulee ilmeiseksi. Arkkitehtuuri voidaan siis tehdä iteratiivisesti myös sidosryhmien kannalta, kunhan nämä vakuutetaan siitä, että asetetut vaatimukset toteutuvat projektin aikana. Arkkitehdin tehtäväksi jää määrittellä, milloin vaatimuksiin tullaan vastaamaan eli milloin tietyt arkkitehtuuripäätökset tehdään.

Madisonin (2010) mukaan ketterän kehityksen ja arkkitehtuurin yhdistämisen ydin on arkkitehtuurin arvon esiintuomisessa. Mikäli arkkitehtuurin arvoa ohjelmistotuotteessa ei ymmärretä, saa se luonnollisesti matalan prioriteetin tehtävälisellä. Arkkitehtuuripäätösten ja niiden realisoinnin vaikutukset pitää pystyä ilmaisemaan vaikutuksena bisnesarvoon. Ketteryys tarjoaa arkkitehdille mahdollisuuden kommunikoida arkkitehtuurista kulloinkin saatavaa hyötyä asiakkaalle. Arkkitehti on loistoa-semassa fasilitoimaan molemminpuolisen ymmärryksen saavuttamista - selittämään teknisiä rajoitteita asiakkaille ja kartoittamaan asiakkaan bisnesvaatimuksia kehitysryhmälle selitettäväksi. Erityisesti projektin ensimmäisten sprinttien aikana vaikuttaminen on tärkeää, sillä tällöin arkkitehturaalisesti vaativien ja vaikeasti muutettavien osien toteuttaminen on arvokkaimmillaan (Fowler, 2003). Suurimmassa osassa projekteja merkittävimmät arkkitehturaaliset valinnat tapahtuvat heti alkuun (Abrahamsson et al., 2010). Suunnittelun tuloksena voi kuitenkin olla tietyn lopputuloksen sijaan etenemissuunta (Madison, 2010).

6 Ketterä arkkitehti ja arkkitehtuurikeskeiset toiminnot

Tässä luvussa käsitellään arkkitehdin roolia ketterässä projektissa ja konkreettisia arkkitehtuurikeskeisiä toimintoja, joilla roolia voidaan toteuttaa. Lopuksi muodostunutta kokonaiskuvaa analysoidaan sen soveltuvuudelta esitettyjen kasvuhaasteiden hallitsemiseen luvussa 5 esiteltyjen periaatteiden mukaisesti.

6.1 Ketterä arkkitehti tarjoaa arvokasta palvelua

Abrahamsson et al. (2010) muistuttavat arkkitehtuurin ja arkkitehdin roolin määrittelyn tärkeydestä. Ei kannata olettaa, että kaikilla projektin osapuolilla on valmis ymmärrys aiheesta. Lisäksi kannattaa ottaa huomioon, milloin arkkitehtuuritoimen-

piteet on parasta jättää sivuun kriittisen julkaistavan version stabiloimiseksi, mutta samalla pitää kirjata arkkitehtuurin laiminlyönnin myötä kertyvästä teknisestä riskistä. Arkkitehti kannattaa määritellä ohjelmiston arkkitehtuurin omistajaksi samaan tapaan kuin asiakas on projektin toiminnallisuuksien omistaja ja määrää toiminnallisuuksien tärkeysjärjestyksen. Blair et al. (2010) muistuttavat, että arkkitehti on ratkaisu kahteen eri ongelmaan: kehitysryhmän toimintaan ja sidosryhmien kanssa kommunikointiin eli sisäiseen ja ulkoiseen. Arkkitehdin määrittely arkkitehtuurin omistajaksi vastaa näistä ongelmista ensimmäiseen.

Blair et al. (2010) pitävät perinteistä arkkitehdin roolia esteenä onnistuneelle kommunikoinnille sidosryhmien kanssa. Sen sijaan, että arkkitehti olisi kehittäjien ja sidosryhmien välissä siirtämässä tietoa vaatimuksista arkkitehtuurikuvauksiksi, ehdotetaan arkkitehdin toimimista kehittäjien neuvonantajana sekä kehittäjien ja sidosryhmien välisen viestinnän tehostajana. Tarkoituksena on välttää arkkitehdin henkilökohtaisen tulkinnan vaikutukset vaatimusten ymmärtämisessä ja mahdollistaa suoremman palautteen antaminen vaatimusten toteutettavuudesta ja lopputuloksen laadusta. Madison (2010) huomauttaa, että arkkitehti on uniikissa asemassa ymmärtämään järjestelmien välistä dynamiikkaa ja osoittamaan laajempia mahdollisuuksia tai ongelmia.

Abrahamsson et al. (2010) esittävät, että arkkitehdin tulee olla osa kehitysryhmää, ei sille ulkoinen taho. Madison (2010) yhtyy fyysisen läheisyyden tärkeyteen ja ehdottaa, että arkkitehdin tulisi olla mukana rakentamassa myös ohjelmiston toiminnallisuutta ja tarjota muille kehittäjille mentorointia erityisesti juuri arkkitehturaalisten seikkojen kuten suunnittelumallien ja laatuattribuuttien kehittämistä koskevan työn yhteydessä. Arkkitehti on arvokkaimmillaan projektissa siirtäessään omaa tietouttaan muille kehittäjille mahdollistaakseen näiden itsenäisen päätöksenteon (Fowler, 2003).

Faber (2010) ehdottaa ylempänä kuvaillun kaltaisen, kehitysryhmän kanssa tiivistä yhteistyötä tekevän, ohjelmiston arkkitehtuurista kehitystä ohjaavan, arkkitehtuuri- ja bisnesvaatimusten välillä tasapainottelevan sekä sidosryhmien välistä viestintää avustavan arkkitehdin käsittämistä projektin sisäisen, arkkitehtuuria toteuttavan palvelun tarjoajana. Määrittelemisen tällä tavoin yhdistää aiemmin käsitellyt ulkoiset ja sisäiset roolit.

Arkkitehti palveluntarjoajana on erityisen osuva nimitys kohdistaaessaan huomion palvelusta saatavaan hyötyyn, jonka merkitys nostettiin esille edellisessäkin kappa-

leessa. Arkkitehtuurista muodostuu palveluprosessin tulos, joka riippuu prosessin kaikista osapuolista. Jotta kehittäjät haluaisivat saatavilla olevaa palvelua käyttää, pitää kehittäjien luottaa arkkitehdin kykyyn ratkoa aitoja ongelmia ja siten nähdä palvelu arvokkaana - tämän luottamuksen saavuttaminen jää valitettavasti arkkitehdin tehtäväksi (Faber, 2010).

6.2 Arkkitehtuurikeskeiset toiminnot ketterän prosessin aikana

Madison (2010) esittää ketterien ja arkkitehtuurikeskeisten menetelmien yhdistelmänä hybridimenetelmää, jossa joukko arkkitehtuurikeskeisiä toimintoja on nivottu ketterään muottiin. Arkkitehtuurin tehtävät määritellään viestinnäksi, laatuattribuuttien mittaamiseksi, toteutusta ohjaavien suunnittelumallien (engl. *design patterns*) luonnosteluksi ja projektille sopivan ohjelmiston ja laitteiston valinnaksi. Näitä on tarkoitus toteuttaa etukäteissuunnittelun, tarinankerronnan, sprintin ja sprintin tuloksien tarkastelun aikana.

Nord & Tomayko (2006) ehdottavat arkkitehtuurikeskeisiksi toiminnoiksi projektia edeltävää laatuattribuuttityöpajaa ja projektin aikaista laatuattribuuttivetoista suunnittelua. Työpajan aikana tehtävälstalla olevien tarinoiden ohelle kehitetään lista laatuattribuuttiskenaarioita, jotka määrittävät toiminnallisuuksille asetettavia vaatimuksia. Skenaariot auttavat sidosryhmiä viestimään vaatimuksiaan kehittäjille ja kertaluontoiseen tapahtumaan voidaan kutsua paikalle tahoja, jotka eivät projektin aikana muutoin olisi jatkuvasti läsnä. Laatuattribuuttivetoisella suunnittelulla tarkoitetaan suunnittelun etenemistä projektin aikana siten, että jokaisessa kehitysvaiheessa valitaan joukko arkkitehtuuritaktiikoita, joiden perusteella laatuvaatimukset toteuttavat konkreettiset suunnittelupäätökset voidaan tehdä. Taktiikan voidaan sanoa kehystävän tuleva suunnittelutoiminta muotoon, josta järjestelmän kokonaisrakenne voidaan hahmottaa.

Voidakseen muodostaa osan sprintissä tapahtuvaa kehitystä, pitää arkkitehtuurin kehittämisen jakautua riittävän pieniin osiin mahtuakseen yhden sprintin sisälle. Madison (2010) esittää jakamisen tapahtuvan arkkitehtuurillisesti merkittävien rajojen perusteella: rajat tunnistetaan ja niiden perusteella varmistetaan, että sprintissä ei käsitellä yhtäaikaaisesti asioita liian monien rajojen välillä. Rajojen määrittely tapahtuu niiden toiminnallisuuksien välille, joihin liittyvä laitteisto, ohjelmisto, suunnit-

telumallit tai laatuattribuutit poikkeavat merkittävästi. Liian moni sprintin aikana auki oleva arkkitehtuuriraja aiheuttaa monia yhtäaikaisesti ratkaistavia haasteita ja nostaa projektin riskiä. Tällä tavoin arkkitehtuurirajat saattavat vaikuttaa myös sprinttiin valikoitaviin toiminnallisuuksiin. Valitettavasti rajat voivat poiketa merkittävästi sovelluksesta riippuen, eikä valmiita kaavoja niiden löytämiseen ole, jolloin joudutaan luottamaan yksin kehittäjien asiantuntemukseen.

Kaikkea ei ehditä tekemään yhdessä sprintissä, joten tekemättömistä arkkitehtuuri-toimenpiteistä on pidettävä kirjaa (Abrahamsson et al., 2010; Madison, 2010). Madisonin mukaan avoimet arkkitehtuuritehtävät kannattaa laittaa erilliseen työlistaan ja nostaa projektin varsinaiseen listaan tarvittaessa. Mikäli toimenpiteiden bisnesarvo on kommunikoitu asiakkaalle oikein, johtaa tavallisiin ketteriin periaatteisiin nojaaminen korkea-arvoisimpien arkkitehturaalistenkin tehtävien toteuttamiseen ensin. Abrahamsson et al. (2010) käyttävät vetoketjumetaforaa kuvaamaan toiminnallisten ja arkkitehturaalisten tehtävien projektin aikaista limittymistä, jonka käyttäjälle näkyvien toiminnallisuuksien ja teknisten ominaisuuksien riippuvuuksien ymmärtäminen mahdollistaa.

6.3 Ketterän arkkitehtuurin kokonaiskuva

Tämän ja edellisen luvun ehdotukset muodostavat kokonaisuuden, joilla ohjelmistoarkkitehtuurin ja ketterien mallien ristiriitaa voidaan yrittää helpottaa. Liitetään seuraavaksi luvussa 5 esitetyt määriteltävät tekijät yllä esitettyihin ratkaisuehdotuksiin ja käsitellään sen jälkeen muodostuvaa kokonaiskuva.

Arkkitehtuuri määritellään suunnittelupäätöksinä, jotka tehdään sidosryhmien vaatimuksiin perustuen. Suunnittelupäätöksiä tehdään, kun mahdollisuus päätöksen tekoon on tunnistettu ja päätöksen tekemiseen katsotaan olevan riittävästi informaatiota. Vaikka merkittävimmät suunnittelupäätökset tehdään projektin alussa, arkkitehtuuria voidaan kehittää koko projektin ajan. Arkkitehti pitää kirjaa sekä tehyistä että avoimista suunnittelupäätöksistä ja kommunikoi niiden merkitystä sekä kehitysryhmän sisällä että tarvittaessa sidosryhmille. Arkkitehtuurin ensisijainen esitysmuoto on tehtyjen suunnittelupäätösten joukko. Arkkitehti on vastuussa siitä, että suunnittelupäätökset tehdään ajallaan ja että asiakas ymmärtää sekä päätösten että niiden realisoimiseen vaadittavan työpanosten tuottaman arvon. Arkkitehti ei kuitenkaan sanele päätöksiä, vaan työskentelee yhdessä muun kehitysryhmän kanssa

sekä päätösten tuottamisen että realisoimisen osalta.

Tehtyjen ehdotusten ydin on arkkitehdin roolin esiintuomisessa ja huomion kiinnittämisessä arkkitehturaalisten päätösten tekemiseen, joka ketterissä projekteissa usein tapahtuu huomaamatta. Arkkitehdistä ei kuitenkaan pidä muodostua projektin tahdin määräävää suunnittelupäätöstehdasta, vaan taho, joka osoittaa niin projektin kehittäjille kuin sidosryhmillekin projektissa tehtyjä ja vielä tekemättömiä päätöksiä sekä niihin liittyviä riskejä ja mahdollisuuksia. Teoksessa *Planning Extreme Programming* Beck & Fowler (2000) lainaavat Eisenhoweria:

In preparing for battle I have always found that plans are useless, but planning is indispensable.

Mikäli suunnitelmavetoisia toimintoja sovelletaan, voivat ne itsetarkoituksellisuuden sijaan toimia keinona paremman tiedon saavuttamiselle projektin tilasta. Ketterien periaatteiden mukaisesti keskiössä ovat tällöin suunnitelmia laativat ja toteuttavat ihmiset, ei dokumentteja vaativa prosessi.

7 Pohdinta

Tutkimuksessa selvitettiin ketteryyden asiakaskeskeistä arvонуontimekanismia ja liitettiin tämä teknisen riskin mahdollisuuden kasvamiseen projektin mittakaavan kasvaessa. Mahdollisena ratkaisuna teknisen riskin hillitsemiseen esiteltiin ohjelmistoarkkitehtuuri, jonka käsittely ketterissä malleissa havaittiin vaillinaiseksi. Ratkaisuksi ehdotettiin ohjelmistoarkkitehdin roolin sovittamista ketteriin malleihin sopivalla tavalla ja joukkoa ohjelmistokehitysiteeraatioiden aikana toteutettavia toimenpiteitä. Tutkimuksessa yhdistettiin uudella tavalla arkkitehdin roolin määrittely projektin sisäisenä palveluntarjoajana ja ketterä, suunnittelupäätöksenteon kautta muodostuva arkkitehtuuri.

Ongelmaa tutkittaessa tehtiin oletus tilanteesta, jossa ketteryydelle otollisessa asiakasohjelmistoprojektitympäristössä kohdataan projektin luonnollisen kasvun myötä uusia haasteita. Ehdotetut ratkaisut eivät kata koko suunnitelmavetoisten ja arkkitehtuurikeskeisten kehitysmallien perinteisesti hallitsemaa ympäristöä eivätkä oletettavasti tee ketteristä malleista sopivampia erityisten suurten projektien toteuttamiseen. Sen sijaan ratkaisut ovat kenties relevantteja myös pienemmissä projekteissa, joissa vakavia arkkitehtuuriongelmia ei vielä tavata, sillä pienikin sovellus voi kasvaa

kokoon, jossa arkkitehtuurin huomioinnista on suurta hyötyä.

Tutkimustuloksena tehdyt ehdotukset ovat mainitut oletukset täyttävissä ympäristöissä käytettävissä sellaisinaan tai soveltaen. Täytyy kuitenkin ottaa huomioon, että tässä tutkimuksessa tai pohjaehdotuksia tehneissä teksteissä ei ole kerätty empiiristä aineistoa ehdotusten tueksi. Arkkitehtuurin ja ketteryuden yhdistäminen on aiheena niin uusi, ettei sitä ole ehditty vielä tutkia. Lisäksi ehdotukset ovat ketteryyden ajatusmaailmaa noudatellen abstrakteja ja luottavat viime kädessä arkkitehdin asiantuntemukseen. Ehdotukset toivottavasti kuitenkin luovat kehyksen arkkitehdin toiminnalle ketterässäkin prosessissa.

Suunnitelmavetoisuudesta ketteryyteen siirtymistä on mahdollista tarkastella projektiliiketoiminnan luonteen uudelleenkartoittamisena tuotteesta palveluksi. Ennalta asiakkaan kanssa sovitun lopputuotteen toimittamisen sijaan myydään palvelua projektin asiakkaan bisnesvaatimusten toteuttamiseksi. Ketterässä projektissa ei ole täsmällistä kuvausta asiakkaan vaatimuksista, joten asiakkaan pitää itse olla läsnä palvelutapahtumassa eli kehitysprosessissa. Palvelua ei kuitenkaan voi myydä ellei sitä ole stabilisoitu (Araujo & Spring, 2006). Siispä vaaditaan toiminnalle edes jonkinlainen kehys, mitä virkaa ketterät prosessimallit toimittavat.

Ohjelmistoarkkitehtuurin täytyy kenties käydä läpi samankaltainen paradigmanmuutos ja tarjota tuotteiden - perinteisten arkkitehtuurikuvausten - sijaan projektien sisäistä palvelua arkkitehtuurinäkökulman edistämiseksi. Kuten arkkitehtuurillekin, on palveluille ollut vaikea löytää konsistenttia määritelmää (Araujo & Spring, 2006). Moderni näkemys arkkitehtuurista kuvauksensa sijaan joukkona suunnittelupäätöksiä eli jatkuvana prosessina heijastelee mainittua muutosta, sillä näkemystä perustellaan juuri tuotoskeskeisyyden aiheuttamilla ongelmilla (Jansen & Bosch, 2005). Näkökulma kruunattiin tässä tutkimuksessa käsittelemällä arkkitehtuuria palveluprosessina, jonka tarjoajana arkkitehti toimii.

Ohjelmistoarkkitehtuurin alan tutkimustulokset siirtyvät varsin hitaasti käytännön tasolle (Kruchten et al., 2006), mikä saattaa johtua niiden sidonnaisuudesta ketterien kehittäjien vieroksumaan suunnitelmavetoisuuteen. Kenties esimerkiksi juuri tässä tutkimuksessa esitettyjen kaltaisilla, aidosti ketteristä lähtökohdista tehdyillä ehdotuksilla on paremmat mahdollisuudet tulla käytetyiksi.

8 Yhteenveto

Ketterillä ohjelmistokehitysmenetelmillä tuotettujen ohjelmistojen kasvavaa monimutkaisuutta voidaan pyrkiä hallitsemaan ohjelmistoarkkitehtuurikeskeisillä toiminnoilla. Ohjelmistoarkkitehtuurin ja ketterän ohjelmistokehityksen menetelmien yhdistäminen vaatii ohjelmistoarkkitehtuurikäsitteen nykyaikaistamista. Arkkitehtuuri ja sen tuottaminen voidaan nähdä iteratiivisena prosessina, kun sitä tarkastellaan peräkkäisten suunnittelupäätösten näkökulmasta. Ohjelmistoarkkitehdin rooli sopii ketteriinkin projekteihin, kunhan arkkitehti toimii dokumenttien tuottajan sijaan asiantuntijapalveluna projektiryhmälle ja sen sidosryhmille. Arkkitehti voi olla projektin omistajaan verrattavissa oleva arkkitehtuurin omistaja, jonka tehtävä on ylläpitää listaa tekemättömistä arkkitehturaalisista päätöksistä ja tehtävistä. Toiminnot vaativat kuitenkin arkkitehdiltä kykyä saavuttaa kehitysryhmän luottamus sekä määrittää tehtävien arvo asiakkaalle, jotta asiakas voi ketterien periaatteiden mukaisesti priorisoida tehtävät.

Viitteet

- Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L. & Zaremski, A. (1997), "Recommended Best Industrial Practice for Software Architecture Evaluation", tekn. rap., Software Engineering Institute, Carnegie Mellon University.
- Abrahamsson, P., Babar, M. & Kruchten, P. (2010), "Agility and architecture: Can they coexist?", *IEEE Software*, 27:2, s. 16–22.
- Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002), *Agile software development methods. Review and analysis.*, VTT Technical Research Centre of Finland.
- Araujo, L. & Spring, M. (2006), "Services, products, and the institutional structure of production", *Industrial Marketing Management*, 35:7, s. 797–805.
- Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, B., Jarrick, Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. & Thomas, D. (2001), "Manifesto for Agile Software Development", Saatavilla Internetistä: <http://agilemanifesto.org>, noudettu 2010.10.23.
- Beck, K. & Fowler, M. (2000), *Planning Extreme Programming*, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA.
- Blair, S., Watt, R. & Cull, T. (2010), "Responsibility-Driven Architecture", *IEEE Software*, 27:2, s. 26–32.
- Boehm, B. (2002), "Get ready for agile methods, with care", *IEEE Computer*, 35:1, s. 64–69.
- Boehm, B. & Sullivan, K. (2000), "Software Economics: A Roadmap", teoksessa *Proceedings of the Conference on The Future of Software Engineering*, ACM, s. 319–343.
- Boehm, B. & Turner, R. (2003), "Using risk to balance agile and plan-driven methods", *IEEE Computer*, 36:6, s. 57–66.
- Bosch, J. (2004), "Software architecture: The next step", teoksessa *Proceedings of the First European Workshop on Software Architecture*, Springer, s. 194–199.
- Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J. & Little, R. (2002), *Documenting software architectures: views and beyond*, Pearson Education.
- Cohn, M., Highsmith, J. & Benefield, G. (2006), *Agile estimating and planning*, Prentice Hall Professional Technical Reference.
- Denne, M. & Cleland-Huang, J. (2005), "The incremental funding method: Data-driven software development", *IEEE Software*, 21:3, s. 39–47.

- Ellis, W., Rayford, D., Hilliard, R., Saunders, T., Wade, R., Poon, P. & Sherlund, B. (2002), "Toward a recommended practice for architectural description", teoksessa *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems*, s. 408–413.
- Faber, R. (2010), "Architects as service providers", *IEEE Software*, 27:2, s. 33–40.
- Falessi, D., Cantone, G., Sarcia, S., Calavaro, G., Subiaco, P. & D'Amore, C. (2010), "Peaceful coexistence: Agile developer perspectives on software architecture", *IEEE Software*, 27:2, s. 23–25.
- Fowler, M. (2003), "Who needs an architect?", *IEEE Software*, 20:5, s. 11–13.
- (2004), "Is Design Dead?", Saatavilla Internetistä: <http://www.martinfowler.com/articles/designDead.html>, noudettu 2010.11.20.
- Fowler, M. & Highsmith, J. (2001), "The agile manifesto", *Software Development*, 9:8, s. 28–35.
- Glass, R. L. (2001), "Agile Versus Traditional: Make Love, Not War!", *Journal of Information Technology Management*, 14:12, s. 12–17.
- Gore, M. (2003), "Thoughts on the information system architect role", teoksessa *Proceedings of the International Conference on Information Technology: Computers and Communications*, s. 706–710.
- Highsmith, J. & Cockburn, A. (2002), "Agile software development: The business of innovation", *IEEE Computer*, 34:9, s. 120–127.
- Huo, M., Verner, J., Zhu, L. & Babar, M. (2004), "Software quality and agile methods", teoksessa *Proceedings of the 28th Annual International Computer Software and Applications Conference*.
- Jansen, A. & Bosch, J. (2005), "Software architecture as a set of architectural design decisions", teoksessa *Proceedings of the Working IEEE/IFIP Conference on Software Architecture*.
- Kettunen, P. (2009), "Agile software development in large-scale new product development organization: team-level perspective", väitöskirja, Helsinki University of Technology.
- Kruchten, P. (1995), "The 4+1 view model of architecture", *IEEE Software*, 12:6, s. 42–50.
- (2004), *The rational unified process: an introduction*, Addison-Wesley Professional.
- (2007), "Voyage in the agile memplex", *ACM Queue*, 5:5, s. 38–44.

- Kruchten, P., Obbink, H. & Stafford, J. (2006), "The past, present, and future for software architecture", *IEEE Software*, 23:2, s. 22–30.
- Lago, P., Avgeriou, P. & Hilliard, R. (2010), "Software Architecture: Framing Stakeholders' Concerns", *IEEE Software*, 27:6, s. 20–24.
- Lindstrom, L. & Jeffries, R. (2004), "Extreme programming and agile software development methodologies", *Information Systems Management*, 21:3, s. 41–52.
- Losavio, F., Chirinos, L., Lévy, N. & Ramdane-Cherif, A. (2003), "Quality characteristics for software architecture", *Journal of Object Technology*, 2:2, s. 133–150.
- Lundberg, L., Bosch, J., Häggander, D. & Bengtsson, P. (1999), "Quality Attributes in Software Architecture Design", teoksessa *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, s. 353–362.
- Madison, J. (2010), "Agile-Architecture Interactions", *IEEE Software*, 27:2, s. 41–48.
- Nonaka, I. & Konno, N. (1998), "The concept of Ba", *California management review*, 40:3, s. 40–54.
- Nord, R. & Tomayko, J. (2006), "Software architecture-centric methods and agile development", *IEEE Software*, 23:2, s. 47–53.
- Paetsch, F., Eberlein, A. & Maurer, F. (2003), "Requirements engineering and agile software development", teoksessa *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, s. 308–313.
- Perry, D. & Wolf, A. (1992), "Foundations for the study of software architecture", *ACM SIGSOFT Software Engineering Notes*, 17:4, s. 40–52.
- Raymond, K. (1995), "Reference model of open distributed processing (RM-ODP): Introduction", teoksessa *Proceedings of the third IFIP TC 6/WG 6.1 international conference on open distributed processing, 1994*, Springer.
- Rozanski, N. & Woods, E. (2005), *Software systems architecture: working with stakeholders using viewpoints and perspectives*, Addison-Wesley Professional.
- Schwaber, K. (1995), "SCRUM Development Process", teoksessa *OOPSLA Business Object Design and Implementation Workshop*, s. 10–19.
- Sullivan, K., Chalasani, P., Jha, S. & Sazawal, V. (1999), "Software design as an investment activity: a real options perspective", *Real Options and Business Strategy: Applications to Decision Making*, s. 215–262.
- Sutherland, J. (2001), "Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies", *Journal of Information Technology Management*, 14:12, s. 5–11.

Tyree, J. & Akerman, A. (2005), "Architecture decisions: Demystifying architecture",
IEEE Software, 22, s. 19–27.