hyväksymispäivä arvosana

arvostelija

Ohjelmistoarkkitehtuurin sisällyttäminen ketteriin ohjelmistotuotantomenetelmiin

Tero Huomo

Helsinki 9.12.2012

Kandidaatin tutkielma

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Tero Huomo

Työn nimi — Arbetets titel — Title

Ohjelmistoarkkitehtuurin sisällyttäminen ketteriin ohjelmistotuotantomenetelmiin

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatin tutkielma	9.12.2012	19 sivua

Tiivistelmä — Referat — Abstract

Ketterän ohjelmistokehitys myötä ohjelmistosuunnittelu ja ohjelmistoarkkitehtuurin merkitys on muuttunut. Perinteisessä vesiputousmallissa suunnittelulle ja määrittelylle varattiin kuukausia ennen ohjelmoinnin aloittamista. Ketterässä ohjelmistotuotannossa ei tehdä raskasta suunnitteluprosessia etukäteen, vaan järjestelmän rakenne muotoutuu asiakkaan ja kehitysryhmän valitsemien toteutettavien ominaisuuksien mukaan. Arkkitehtuuri muodostuu inkrementaalisesti kehitysjaksosta toiseen. Tarpeen vaatiessa järjestelmän rakennetta refaktoroidaan, eli sitä muutetaan yksinkertaisemmaksi ja selkeämmäksi muuttamatta järjestelmän toiminnallisuutta.

Osa ohjelmistokehittäjistä pitää ketterän ohjelmistokehityksen suhdetta suunnitteluun haastavana. Järjestelmän rakenteen luominen inkrementaalisesti vaatii tietoa ja taitoa, ja kehitysympäristön mahdolliset rajoitteet tulevat esiin vasta ohjelmoinnin aikana. Ketterässä kehityksessä ei ole luonnollista tilaa ohjelmistoarkkitehtuurille. Ketterä ohjelmistokehitys antaa liian vähän neuvoja rakenteen suunnittelulle ja toteutukselle.

Toisinaan ketterää ohjelmistokehitystä käyttävät yhtiöt ovat sopeutuneet erilaisilla käytännöillä, joilla oppikirjan mukaisia kehitystapoja on muokattu huomioimaan paremmin ohjelmistoarkkitehtuurin haasteet. Tässä tutkielmassa käsitellään viittä käytäntöä, joita suositellaan tai käytetään ohjelmistoarkkitehtuurin sisällyttämiseksi ketteriin ohjelmistotuotantomenetelmiin. Käytännöt antavat vaihtoehtoisia tapoja järjestelmän arkkitehtuurin tuottamiseen, mutta eivät välttämättä ole linjassa ketterien periaatteiden kanssa.

ACM Computing Classification System (CCS):
D.2.11 [Software Architectures]
D.2.9 [Management - Software Process Models]

Avainsanat — Nyckelord — Keywords
Ohjelmistoarkkitehtuuri, ketterä ohjelmistokehitys, suunnittelu

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Sisältö

1 Johdanto

Ohjelmiston arkkitehtuuri voidaan ajatella ohjelman pohjapiirustuksina. Arkkitehtuuri on korkean tason kuvaus järjestelmän komponenteista, komponenttien yhteyksistä sekä järjestelmän osien fyysisestä sijainnista [Iee00]. Arkkitehtuuri käsittää myös järjestelmän rakennetta koskevat päätökset sekä järjestelmän laajuiset ohjelmointikäytänteet. Vesiputousmalli on ohjelmistotuotannon prosessimalli, jossa arkkitehtuuri on suuressa osassa. Ohjelmisto suunniteltiin kokonaisuudessaan etukäteen. Järjestelmän rakenteen ja sen yksityiskohtien suunnitteluun varattiin kuukausien mittainen suunnitteluvaihe. Tarkan suunnitelman pohjalta ohjelmoinnin oli tarkoitus olla lähes mekaanista työtä.

Vesiputousmallin käsitys ohjelmistotuotantoprosessista on ongelmallinen. Ongelmia erittelee esimerkiksi Craig Larman [Lar03]. Vesiputousmallin ongelmiin ratkaisuja tarjoaa ketterä ohjelmistokehitys, jonka myötä suunnittelun rooli ohjelmistotuotannossa on muuttunut. Ketterän ohjelmistokehityksen julistus [BBB01a] kuvaa ketterän ohjelmistokehityksen arvot. Ketterässä ohjelmistotuotannossa on lähtökohtana jakaa projekti kehitysjaksoihin, joissa keskitytään vain pieneen osaan järjestelmän toiminnallisuutta. Tavoitteena on tuottaa tasaisin väliajoin asiakkaalle arvokasta ohjelmistoa. Tärkeä ketterä arvo on muutosvalmius [BBB01b]. Jatkuviin muutoksiin varautumista arvostetaan muuttumatonta suunnitelmaa enemmän.

Ketterää ohjelmistokehitystä ja arkkitehtuurilähtöistä ohjelmistotuotantoa pidetään usein vastakkainasetteluna. Ketterä ohjelmistokehitys hylkää perinteisen arkkitehtuurisuunnittelun ja järjestelmän rakentamisen lähes kokonaan, sillä tiukka suunnitelmallisuus ei ole ketterien arvojen mukainen. Esimerkiksi Extreme Programming kieltää tuottamasta ylimääräistä ohjelmakoodia, jos sitä ei tarvita juuri kyseisellä hetkellä [BeC04]. Osa ohjelmistokehittäjistä pitää ketterälle ohjelmistokehitykselle tyypillistä inkrementaalista järjestelmän rakentamista haastavana. Haasteiksi nähdään esimerkiksi järjestelmän skaalautuvuus, kehityksen jakaminen usean kehitysryhmän välille sekä järjestelmän suunnittelun vaikeus [Boe02].

Ketterän ohjelmistokehityksen arvoihin [BBB01b] kuuluu projektiryhmien mukautuvuus ympäristön vaatimuksiin. Projektiryhmät ovat omaksuneet erilaisia tapoja huomioida ohjelmistoarkkitehtuuri ja arkkitehtuurisuunnittelu ketterässä ohjelmistokehityksessä. Tapoja ovat eritelleet erityisesti Pekka Eloranta ja Kai Koskimies [ElK13]. Tässä tutkielmassa kuvataan näitä tapoja ja muita vaihtoehtoisia käytänteitä, joilla ketterässä ohjelmistokehityksessä voidaan ottaa ohjelmistoarkkitehtuuri

paremmin huomioon.

Käytänteiden esittelyn lisäksi tarkastellaan kritiikkiä, jota arkkitehtuuriyhteisö esittää puhdasta oppikirjojen mukaista ketterää ohjelmistokehitystä vastaan. Ketterän ohjelmistokehityksen rajoituksien tarkastelu auttaa hahmottamaan sitä, miten käytänteet vastaavat osaan esitetystä kritiikistä. Käytänteiden yhteydessä käsitellään kuitenkin myös sitä, mitä ketterän kehityksen perimmäisiä arvoja uudet tavat mahdollisesti rikkovat tai saattavat rikkoa väärin käytettyinä.

Luvussa 2 eritellään tarkemmin suunnittelun eri osat ketterässä ohjelmistokehityksessä. Käsittely on rajattu Extreme Programmingiin sekä Scrumiin. Luvussa 3 käsitellään kritiikkiä, joka koskee luvussa 2 esiteltäviä suunnittelumenetelmiä ja niiden rajoittuneisuutta ohjelmistoarkkitehtuurin kannalta. Luvussa 4 kuvataan käytäntöjä, joilla arkkitehtuuri ja arkkitehtuurin suunnittelu on mahdollista liittää osaksi ketterää tuotantoprosessia kritiikin kumoamiseksi. Yhteenveto aiheesta esitetään luvussa 5.

2 Suunnittelu ketterässä ohjelmistotuotannossa

Tässä luvussa käsitellään karkeasti suunnittelua suosituimmissa ketterän ohjelmistokehityksen suuntauksissa. Tutkielmassa keskitytään Scrumiin [ScB01] ja Extreme Programmingiin (XP) [BeC04], jotka ovat vuoden 2011 käytetyimmät ketterän ohjelmistokehityksen mallit [Ver11]. Yhteensä kahta lueteltua menetelmää käytti yli 60% ketteriä menetelmiä hyödyntävistä projekteista. Scrumia ja Extreme Programmingia käytetään erikseen ja yhdistettynä. Osa harvinaisemmin käytetyistä ketteristä menetelmistä saattaa antaa toisenlaisia ohjeita suunnittelulle, mutta ketterän ohjelmistokehityksen julistuksen [BBB01a] pohjalta ne noudattavat useita samoja tapoja.

Ketterissä menetelmissä projekti tuotetaan iteraatioissa. Pitkä projekti jaetaan lyhyisiin tuotantojaksoihin. Scrumissa *pyrähdykseksi* (sprint) kutsutun jakson suositeltava pituus on tyypillisesti kahdesta neljään viikkoon [ScB01]. Extreme Programmingissa puolestaan suositellaan pituudeksi 1-3 viikkoa [BeC04, s. 120]. Jokaisen tuotantojakson tavoitteena on toimiva ja käytettävä ohjelma. Tämä mahdollistaa ohjelmiston käyttöönoton jo tuotannon varhaisessa vaiheessa, vaikka järjestelmä sisältäisi vasta murto-osan lopullisen tuotteen toiminnallisuudesta.

Tuotantojaksoissa toteutetaan aina muutamia asiakkaan vaatimia ominaisuuksia.

Scrumissa toteuttavat vaatimukset valitsevat tuotantoryhmä ja asiakas yhteistyössä. Samoin toimii myös Extreme Programming. Valinnassa asiakkaan mielipide on tärkein, sillä hän asettaa vaatimuksensa tärkeysjärjestykseen. Järjestelmä rakentuu inkrementaalisesti, kun tuotantojaksoon valitut toiminnot liitetään osaksi aiemmissa jaksoissa toteutettua järjestelmää. Järjestelmän laajetessa ja vaatimusten muuttuessa ohjelman sisäistä rakennetta muokataan, jos rakennetta voidaan siten selkeyttää ja parantaa.

Ketterässä ohjelmistokehityksessä tapahtuvan suunnittelun voi jakaa karkeasti kolmeen osaan:

- *Julkaisun suunnittelu* (release planning)
- *Iteraation suunnittelu* (iteration planning)
- Kehitysjakson aikana tapahtuva suunnittelu

2.1 Julkaisun suunnittelu

Julkaisun suunnittelusta käytetään Scrumissa sekä Extreme Programmingissa nimeä release planning. Molemmissa suuntauksissa järjestetään projektin aluksi lyhyt suunnitteluvaihe, jossa kerätään, kartoitetaan ja määritellään asiakkaan vaatimukset. Vaatimukset kerätään usein *käyttäjätarinoina* (user story), joista yksittäinen tarina selvittää yhden ohjelmalta vaaditun toiminnallisuuden. Esimerkiksi "opiskelija kirjautuu järjestelmään"ja "opiskelija ilmoittautuu laskuharjoitusryhmään"voisivat olla kaksi erillistä käyttäjätarinaa ilmoittautumisjärjestelmässä.

Käyttäjätarinat kerätään tärkeysjärjestyksessä pidettävään työlistaan, jota Scrumissa kutsutaan nimellä backlog [Sch04, s. 10]. Listan alkupäähän sijoitetaan tärkeimmät ja yksityiskohtaisimmin määritellyt ominaisuudet. Tärkeysjärjestyksen päättää asiakas. Suunnittelutilaisuudessa arvioidaan myös käyttäjätarinoiden ohjelmoimiseen kuluvaa aikaa [Kni07]. Ajan perusteella voidaan arvioida ja suunnitella projektin alustava aikataulu.

Julkaisun suunnittelu on projektin lähtöpiste, mutta sitä ei lopeteta aloituksen jälkeen. Asiakkaalla on aina mahdollisuus priorisoida työlistaa uudestaan sekä lisätä listaan uusia vaatimuksia. Ketterien menetelmien olettamus on, ettei asiakas välttämättä tiedä vielä projektin alussa, mitkä ovat hänen tarkat vaatimuksensa järjestelmältä. Vaatimukset tarkentuvat ja lisääntyvät projektin edetessä. Ketterien kehitysryhmien tulee varautua alati muuttuviin vaatimuksiin [BBB01b].

2.2 Iteraation suunnittelu

Iteraation suunnittelu on toiselta nimeltään *pyrähdyksen suunnittelu* (sprint planning). Ennen jokaista kehitysjaksoa Scrumissa järjestetään lyhyt kaksiosainen suunnittelutilaisuus [Kni07]. Extreme Programming ei tee kaksiosaista jakoa, mutta Beck ja Fowler [BeF00] kuvailevat hyvin samankaltaisen lähestymistavan. Ensimmäisessä osassa on tarkoitus selvittää seuraavan tuotantojakson tavoite. Seuraavaan tuotantojaksoon valitaan toteutettavat käyttäjätarinat, joiden määrittelyä tarkennetaan tarpeen vaatiessa. Käyttäjätarinoita voi myös pilkkoa tai järjestää uudestaan, jotta asiakas ja kehitysryhmä pääsevät yhteisymmärrykseen toteutettavista ominaisuuksista [Kni07]. Kehitysryhmä sitoutuu valittujen vaatimusten toteuttamiseen.

Käyttäjätarinat voivat selkeästi liittyä toisiinsa ja muodostaa järjestelmän osia. Iteraation suunnittelun ensimmäisessä osassa voidaan kiinnittää huomiota siihen, että kehitysjaksoon otetut käyttäjätarinat liittyisivät selkeästi toisiinsa ja muodostaisivat isompia kokonaisuuksia järjestelmästä.

Kehitysjakson suunnittelun toinen vaihe käytetään suunnitellessa sitä, miten kehitysjakson tavoitteet saavutetaan [Kni07]. Kehitysryhmä suunnittelee ja miettii tarvittavalla tasolla, miten vaatimukset toteutetaan ja mitä niiden toteuttaminen vaatii. Scrumissa on myös tapana pilkkoa käyttäjätarinat vielä pienemmiksi, maksimissaan yhden työpäivän vieviksi työtehtäviksi (task). Työtehtäviä voi olla esimerkiksi yksittäisen komponentin toteuttaminen, yksikkötestien kirjoittaminen tai tietokantapalvelimen pystyttäminen.

2.3 Kehitysjaksossa tapahtuva suunnittelu

Kehitysjakson aikana tapahtuvaan suunnitteluun ketterät ohjelmistokehitysmetodiikat antavat vähiten ohjeita. Scrum-oppaat eivät ota lainkaan kantaa kehitysjakson sisäiseen suunnitteluun tai suunnittelutilaisuuksiin. Scrum luottaa ryhmän itseorganisoituvuuteen ja yksilöiden taitoihin. Itseorganisoituvuus ja ryhmään luottaminen ovat osa ketterän julistuksen periaatteeita [BBB01b].

Ketterän kehityksen julistuksen perusperiaatteissa luetellaan myös yksinkertaisuuden vaaliminen. Extreme Programmingin ohje kehitysjaksossa tapahtuvaan suunnitteluun on turhan työn minimointi. Kent Beck painottaa järjestelmän rakentamisessa mahdollisimman yksinkertaista lähestymistapaa [BeC04]. Kun kehittäjä saa työtehtäväkseen uuden toiminnallisuuden, tulisi se liittää järjestelmään aluksi niin yksinkertaisesti kuin mahdollista. Ylimääräistä työtä ei saa tehdä, vaikka ylimääräiset

lisäykset saattaisivat vaikuttaa tulevaisuuden kannalta hyödyllisiltä.

Extreme Programming korostaa myös refaktoroinnin tärkeyttä. Refaktorointi on järjestelmän rakenteen muokkaamista yksinkertaisemmaksi ja selkeämmäksi ilman, että järjestelmän toiminnallisuus muuttuu. Martin Fowlerin mukaan jatkuvalla refaktoroinnilla saavutetaan mahdollisimman yksinkertainen rakenne [Fow01]. Järjestelmään voi toteuttaa esimerkiksi tunnetun suunnittelumallin, jos se tekee ohjelmasta selkeämmän ja ylläpidettävämmän.

3 Arkkitehtuuriyhteisön kritiikki

Ketterän kehityksen ja arkkitehtuurivetoinen ohjelmistokehityksen vastakkainasettelu tiedostetaan useassa lähteessä. Varsinaisia syitä vastakkainasettelulle esitetään kuitenkin harvoin. Esimerkiksi vastakkainasettelua käsittelevät Abrahamsson ja kumppanit [ABK10] sekä Kruchten [Kru10] kyseenalaistavat koko vastakkainasettelun tarpeellisuuden. Artikkelit keskittyvät siihen, miten vastakkainasettelua pitäisi tutkia, tai onko ajatus vastakkainasettelusta tarpeeton.

Tässä luvussa keskitytään arkkitehtuuriyhteisön kritiikkiin ketterästä ohjelmistokehityksestä, jotta syyt luvussa 4 esiteltävien käytäntöjen taustalla ovat selvät. Käsite arkkitehtuuriyhteisö tarkoittaa tässä yhteydessä ohjelmistokehittäjiä, jotka painottavat kehitystyössä ohjelmistoarkkitehtuurin ja suunnitelmallisuuden merkitystä, mutta joiden mielestä ketterä kehitys ei kiinnitä niihin huomiota riittävästi. Tutkielmassa sivuutamme ketterän ohjelmistokehityksen puolestapuhujien vastaukset esitettyyn kritiikkiin.

Ohjelmistoarkkitehtuurille ei ole tarkkaa yksiselitteistä ja kaikkien hyväksymää määritelmää. Yksi tapa määritellä ohjelmistoarkkitehtuuri on kutsua sitä järjestelmän pohjapiirustuksiksi. IEEE:n standardin [Iee00] mukaan arkkitehtuuriin kuuluu järjestelmän korkean tason rakenne, joka sisältää järjestelmän osat, osien väliset suhteet sekä tiedon osien yhteistyöstä. Arkkitehtuuri keskittyy järjestelmän keskeisiin osiin ja teknisiin kysymyksiin, kuten järjestelmän hajautukseen, skaalautuvuuteen sekä tietoturvaan. Arkkitehtuuri määrittelee järjestelmän suuret linjat.

Ketterän kehityksen pääpaino on toteuttaa järjestelmän rakenne inkrementaalisesti pienissä osissa. Barry Boehm pitää järjestelmän pilkkomista ja osista yhtenäisen järjestelmän koostamista haastavana taitona [Boe02]. Ketterä kehitys vaatii tuntemusta ohjelmoitavasta laitealustasta, suunnittelumalleista sekä yleisimmistä järjes-

telmärakenteista. Ilman erillistä suunnitteluvaihetta projekti vaatii kokeneita työntekijöitä. Boehm muistuttaa, että 49,9% ohjelmoijista ovat keskimääräistä huonompia. Myös Larry Constantine yhtyy ajatukseen siitä, että maailmassa on vain rajattu määrä kokeneita ja osaavia ohjelmoijia [Con01].

Ohjelmointia edeltäneessä suunnitteluvaiheessa on yleensä selvitetty rakenteen ja vaatimusten lisäksi ympäristön ja kohdelaitteiston asettamat rajoitteet. Suunnitteluvaiheen puuttuessa rajoitteet ja vaatimukset tulevat esille vasta kehityksen aikana. Järjestelmän rakenteen suunnitteluun ja rajoitteiden kartoittamiseen varatun suunnitteluvaiheen puuttuminen onkin yhtiöille suurin syy olla siirtymättä ketterään kehitykseen [Ver11].

Erillisen suunnitteluvaiheen puute voi tehdä projektista vaikean jakaa usealle kehittäjäryhmälle. Kehityksen aikana eri ryhmille jaetuista osista voi paljastua yllättäviä riippuvaisuuksia. Kehitysryhmien välisten riippuvaisuuksien vuoksi ryhmät joutuvat odottamaan tai jäljentämään toisen kehitysryhmän työtä. Esimerkki tapauksesta on MSLite-projekti [NOS12]. Se jouduttiin aloittamaan alusta yllättävien riippuvuussuhteiden vuoksi.

Ohjelmiston alustavan rakenteen puuttuminen voi luoda myös tarpeen suurelle määrälle muokkaamista ja uudelleen tehtävää työtä. Mahdollisimman yksinkertainen arkkitehtuuri ei välttämättä skaalaudu asiakkaan tarpeisiin. Vastaavasti lisätyötä tuottaa se, jos rakenne pidetään liian joustavana ja skaalautuvana jatkokehitystä tai uusia omanaisuuksia varten [Boe02]. Ketterät menetelmät usein arvostavat muokkaavaa työtä refaktoroinnin muodossa, mutta järjestelmän laajuinen refaktorointi vie aikaa ja voi viivästyttää projektia.

Extreme Programming painottaa myös turhan työn minimointia, kuten esitettiin luvussa 2. Tapaa perustellaan ennalta-arvaamattomilla muutoksilla ja asiakkaan usein huonosti määrittelemillä vaatimuksilla. Boehmin [Boe02] mukaan käytäntö on kuitenkin huono silloin, jos vaatimukset ovat tarkasti tiedossa. Käytäntö hylkää aikaisin tehtävät järjestelmän kannalta tärkeät lisäykset, vaikka niiden lisääminen myöhemmin olisi varmaa. Boehm huomauttaa myös ongelmista asiakkaiden kanssa, jos kehitysryhmä ei luota asiakkaan vaatimusten muuttumattomuuteen. Tämä voi herättää asiakkaassa vastaavaa epäluottamusta kehitysryhmää kohtaan.

Turhan työn välttäminen voi tavoitella myös sitä, että tärkeät päätökset jätetään viimeiseen mahdolliseen hetkeen [BeC04, s. 23]. Kun päätökset tehdään viimeisenä mahdollisena hetkenä, on kohdelaitteistosta ja järjestelmästä eniten tietoa. Uuden tiedon perusteella tehdään todennäköisimmin oikeat valinnat. Abrahamsson ja

kumppanit huomauttavat, että liian myöhäiset päätökset järjestelmän rakenteesta voivat johtaa projektin kaaokseen [ABK10].

Eloranta ja Koskimies uskovat, että ketterästä ohjelmistokehityksestä puuttuu luonnollinen tila järjestelmän rakenteen suunnittelulle [ElK11]. Järjestelmän sisäinen rakenne ei ole asiakkaalle suoraan hyödyksi, joten ketterä ohjelmistokehitys ei anna ohjelmistoarkkitehtuurille sen vaatimaa arvoa. Järjestelmäsuunnittelun puuttuminen olisi syy kehitysryhmien luomiin ja omaksumiin käytänteisiin, jotka ottavat huomioon myös ohjelmistoarkkitehtuurin merkityksen ohjelmistokehityksessä.

4 Käytänteet arkkitehtuurin sisällyttämiseen

Yhtenä ketterän ohjelmistokehityksen julistuksen kahdestatoista periaatteista ryhmän olisi mukauduttava ollakseen tehokkaampi [BBB01b]. Jokainen kehitysryhmä ja projekti on yksilöllinen - kaikille projekteille ei sovi samanlaiset kehitystavat. Periaate antaa luvan muokata ryhmän toimintaa tarpeen vaatiessa projektin ja ympäristön vaatimuksiin soveltuvaksi. Periaatetta voi hyödyntää myös ratkaisuna kritiikkiin, jota arkkitehtuuriyhteisö esittää ketterää ohjelmistokehitystä vastaan.

Tässä luvussa esitellään käytäntöjä tai metodiikkojen laajennoksia, joilla ohjelmistoarkkitehtuurin työn voi sisällyttää osaksi ketterää ohjelmistokehitystä. Käsittelyssä otetaan huomioon käytänteiden ja ketterien periaatteiden väliset eroavaisuudet sekä ristiriidat. Esiteltävät käytänteet eivät ole täysin ristiriidattomia suosituimpien ketterien ohjelmistokehitysmetodiikkojen tai ketterän ohjelmistokehityksen julistuksen kanssa. Käytänteiden yhteydessä eritellään tärkeimmät periaatteet ja ohjenuorat, joita käytänteet rikkovat.

Käsittelyssä käytetään hyväksi erityisesti Elorannan ja Koskimiehen kyselytutkimuksen raporttia [ElK11] sekä sen pohjalta kirjoitettua artikkelia [ElK13], jotka listaavat kolme tämän tutkielman käsittelemistä viidestä tavasta.

4.1 Sprint 0

Ketterän kehityksen lyhyitä kehitysjaksoja kutsutaan Scrumissa pyrähdyksiksi eli sprinteiksi. Ne numeroidaan järjestyksessä yhdestä eteenpäin. Rawsthorne esitteli vuonna 2007 käsitteen Sprint 0 [Raw07]. Se on projektin aloitustoimiin suunnattu pyrähdys ennen varsinaisia kehitysjaksoja.

Rawsthornen ohjeissa Sprint 0 on kehitystyön alustamiseen tarkoitettu jakso. Ennen ensimmäistä varsinaista kehitysjaksoa siinä voidaan esimerkiksi konfiguroida työntekijöiden kehitystietokoneet, asentaa tarpeelliset palvelimet sekä hoitaa muut projektin aloituksen kannalta tärkeät toimenpiteet. Rawsthorne määritteli jakson noin viikon pituiseksi. Rawsthorne suositteli myös muutaman konkreettisen koodirivin kirjoittamista, jotta kehitystyö saadaan alkuun.

Kirjoituksen kommenteissa Rawsthornen lähestymistapa herättää kysymyksiä ja kritiikkiä. Miten alustusjaksossa tehtävät toimet eroavat niistä, mitä normaalisti tehdään ensimmäisen kehitysjakson aikana? Architech Solutionsin toisenlaisessa näkökulmassa Sprint 0 voidaan suunnata etukäteen suoritettavaan järjestelmän rakenteen suunnitteluun [Arc10]. Sprint 0:ssa luodaan korkean tason suunnitelma ohjelmiston arkkitehtuurista.

Elorannan ja Koskimiehen tutkimuksessa Architech Solutionsin kaltainen Sprint 0 oli käytössä useassa yhtiössä [ElK11]. Käytännössä Sprint 0 rajataan usein samaan 2-4 työskentelyviikkoon kuin muut tuotantojaksot. Ero Vesiputousmallin pitkään ja tarkkaan suunnitteluvaiheeseen on myös se, ettei arkkitehtuurin ole tarkoitus olla lopullinen. Tarpeen vaatiessa sitä laajennetaan ja muutetaan myöhemmissä kehitysjaksoissa. Lisäksi varhaisen arkkitehtuurin luo erillisten ohjelmistoarkkitehtien sijasta sama kehitysryhmä, joka vastaa ohjelmistojärjestelmän toteuttamisesta.

Käytäntöä seuraavat yhtiöt väittävät, että ilman sitä ne olisivat pulassa. Jaksossa käytetään aikaa esimerkiksi entuudestaan tuntemattoman laitteiston vaatimusten ja rajoitusten kartoittamiseen. Hieman vastaavaa käytäntöä on käytetty myös Lean-ohjelmistotuotantoa (Lean software development) hyödyntäneessä MSLite-projektissa. Siinä varhainen arkkitehtuurisuunnitelma oli myös tapa koordinoida järjestelmän osien kehitys useilla projektiryhmille [NOS12].

Suosiostaan huolimatta Sprint 0 on Scrumin ja XP:n käytänteiden vastainen. Esimerkiksi Ken Schwaber ei tunnista termiä lainkaan [Sch10]. Schwaberin mielestä Sprint 0 on tekosyy viivästyttää projektia. Kaikki Sprint 0:ssa tehtävä työ voidaan aivan yhtä hyvin tehdä myöhemmissä iteraatioissa. Schwaberin mukaan paras tapa tutustua kohdelaitteistoon tai uuteen teknologiaan on työskennellä sen parissa.

Ketterän kehityksen julistuksen periaatteiden mukaisesti asiakkaalle tulisi toimittaa toimivaa järjestelmää mahdollisimman aikaisin [BBB01b]. Sprint 0 ei vielä tuota toimivaa ohjelmistoa, toisin kuin ensimmäiset kehitysjaksot normaalisti. Ohjelman sisäinen arkkitehtuuri ei tuota asiakkaalle suoraa arvoa, joten Sprint 0 ei täytä ketterän tuotantojakson vaatimuksia. Etukäteen tehtävä arkkitehtuurisuunnittelu voi

myös väärinkäytettynä johtaa liian lopulliseen järjestelmän rakenteeseen. jolloin rakenteen muuttamisesta tulee työlästä. Kehitysryhmä ei välttämättä pysty vastaamaan projektin vaatimusten muutoksiin.

4.2 Arkkitehtuuri erillisenä prosessina

Ketterässä ohjelmistokehityksessä järjestelmän rakenne muodostuu usein kehitysjaksojen aikana. Ohjelmistoarkkitehtuuri voidaan myös eriyttää omaksi prosessikseen [Lef07, s. 209]. Arkkitehtuurin suunnittelee ja mahdollisesti myös osin toteuttaa erilliset arkkitehdit. Arkkitehtiryhmä noudattaa omaa prosessimalliaan ja aikatauluaan. Arkkitehtiryhmä sitoutuu kehitysryhmän aikataulun perusteella määriteltyihin tarkistuspisteisiin, jolloin tietty osa järjestelmän rakenteesta täytyy olla valmis kehitysryhmää varten.

Arkkitehtiryhmän huomio keskittyy vaatimuksiin, jota ohjelmistolta halutaan myöhemmissä kehitysjaksoissa. Ryhmän vastuu on suunnitella järjestelmän rakenne, joka tukee seuraavia lisättäviä toimintoja. Dean Leffingwellin mallissa arkkitehtuuriryhmä ei yritä luoda järjestelmää kerralla, vaan valmistelee sitä juuri riittävästi tukemaan seuraavia toiminnallisuuksia [Lef07]. Järjestelmän rakenne muodostuu toiminnallisuuksien tapaan iteratiivisesti ja inkrementaalisesti. Ennen rakenteen toteuttamista arkkitehdit todistavat suunnitelmansa sopivaksi esimerkiksi prototyypeillä tai rakennetta demonstroivalla ohjelmakoodilla.

Tyypillisesti osa arkkitehtiryhmän jäsenistä kuuluu myös toiminnallisuuksia ohjelmoivaan kehitysryhmään [ElK13]. Siten molemmat ryhmät pysyvät ajantasalla siitä, mitä järjestelmän rakenteelta vaaditaan, ja miten rakenne rajoittaa ohjelmointia. Jos arkkitehtuurin implementoinnista järjestelmään huolehtii kehitysryhmä, molempiin ryhmiin kuuluva jäsen tietää, miten rakenne toteutetaan seuraavassa kehitysjaksossa.

Arkkitehtiryhmän ei tarvitse noudattaa ketterää prosessimallia, kunhan se pystyy noudattamaan kehitysryhmän prosessimallin määrittämiä aikatauluja. Yksi tapa on kuitenkin se, että myös arkkitehtuuriryhmä noudattaa esimerkiksi Scrumia. Tällöin ohjelmiston toiminnallisuuden kehittäjät ovat ikään kuin asiakkaan roolissa järjestelmän rakenteesta vastaavalle ryhmälle.

Abrahamssonin ja kumppanien [ABK10] mukaan jopa pienimmät päätökset nähdään usein virheellisesti järjestelmän rakenteen kannalta merkittävinä. Eriytetyssä arkkitehtuuriprosessissa on tärkeää ymmärtää, mitkä ovat arkkitehtuurisesti mer-

kittäviä kysymyksiä ja päätöksiä. Siten saadaan eroteltua arkkitehtuuriryhmän ja kehitysryhmän vastuualueet.

Koskimiehen ja Elorannan tutkimuksissa esiintyi muutamia projekteja, joissa käytettiin arkkitehtuuri oli eriytetty omaksi prosessikseen. Yhtiöt käyttivät lähestymistapansa perusteluna sitä, etteivät ne halunneet sotkea arkkitehtuurilla oppikirjan mukaista Scrumin prosessimallia [ElK13]. Scrumia käyttävä kehitysryhmä pystyi täysin keskittymään prosessimallin noudattamiseen sekä ohjelman toiminnallisuuksiin.

Extreme Programmingia käyttäessä arkkitehtuurin eriyttäminen on vaikeaa, sillä XP:n mukaan [BeC04] järjestelmän rakenne ilmaantuu ohjelmoitaessa. Järjestelmä ei enää todennäköisesti olisi mahdollisimman yksinkertainen, jos sitä suunnittelee toinen ryhmä. Toisaalta arkkitehtiryhmä vastaa siitä, että järjestelmä tukee seuraavia laajennoksia, joten kehitysryhmällä on pienempi tarve lisätä ohjelmaan yksinkertaisuuden periaatteen vastaista tulevaisuuteen varautuvaa ohjelmakoodia.

Arkkitehtuuri eriytettynä prosessina varaa ohjelmistoarkkitehtuurille aikaa huomattavasti tavallista ketterää kehitystä enemmän. Arkkitehtiryhmän on mahdollista tuottaa ohjelmalle useita vaihtoehtoisia arkkitehtuureja [ElK11]. Vaihtoehdoista valitaan tilanteeseen sopivin ennen seuraavan kehitysjakson alkua. Parhaimmassa tapauksessa noudatetaan Extreme Programmingin neuvoa viimeisestä mahdollisesta hetkestä tärkeille valinnoille.

Eriytetty arkkitehtuuriprosessi voi kuitenkin johtaa siihen, ettei se huomioi riittävän hyvin järjestelmän muuttuvia vaatimuksia. Asiakas voi vaatia seuraavaan kehitysjaksoon toteutettavaksi vaatimukset, joihin suunniteltu järjestelmän rakenne ei ota kantaa. Jos ohjelmiston kehittäjiä ei kuulu arkkitehtiryhmään, voi kehitysryhmä ja arkkitehtiryhmä ymmärtää vaatimukset tai ryhmien vastuut eri tavoin. Tämä voi luoda järjestelmälle rakenteen, joka rajoittaa kehitysryhmän mahdollisuuksia tai tekee vaatimusten täyttämisen mahdottomaksi.

Ketterän ohjelmistokehityksen julistus [BBB01a] painottaa, että parhaat arkkitehtuurit ja suunnitelmat kumpuavat itseorganisoituvista kehitysryhmistä. Lisäksi pitäisi luottaa, että kehitysryhmät hoitavat työn, johon ne ovat sitoutuneet. Jos arkkitehtuuri on eriytetty kokonaan omaksi prosessikseen, ei järjestelmän rakenteessa ole välttämättä lainkaan kehitysryhmän työpanosta. Järjestelmän rakenteen suunnittelua ei tällöin ole luotettu kehitysryhmälle.

4.3 Suunnittelupiikit

Suunnittelupiikit (design spike) ovat iteraation aikana suoritettavia työjaksoja, joissa ratkotaan järjestelmän laajentamisesta koituvia ongelmia, suunnittelun haasteita sekä uuden teknologian vaikutusta järjestelmään. Dean Leffingwell määrittelee suunnittelupiikit kohdeltaviksi kuten käyttäjätarinat - ne priorisoidaan, estimoidaan ja suoritetaan [Lef07, s. 209]. Suunnittelupiikit auttavat ymmärtämään käytettävän teknologian yksityiskohtia. Suunnittelupiikkejä käytetään myös tutkittaessa vaihtoehtoisia lähestymistapoja.

Extreme Programming hyväksyy suunnittelupiikit käytäntönä. Wells luettelee piikit yhtenä XP:n säännöistä [Wel99]. Wells määrittelee piikit yksinkertaisina ohjelmina, joilla kokeillaan potentiaalisia ratkaisuja vaikeisiin teknisiin tai rakenteellisiin ongelmiin. Tavoitteena on vähentää teknisten ongelmien riskiä sekä lisätä käyttäjätarinoiden aika-arvioiden luotettavuutta. Suunnittelupiikkiin voi muutama kehittäjä varata aikaa jopa yhdeksi tai kahdeksi viikoksi.

Piikit eivät suoraan ole ristiriidassa ketterän kehityksen periaatteiden kanssa. Tutkimustyö ja piikkien käyttö kuitenkin vie kehitysryhmältä resursseja [Lef07]. Ryhmän työvauhti laskee yksittäisissä jaksoissa, jos liian moni kehittäjä osallistuu suunnittelupiikkeihin. Jos resursseja uhrataan liikaa, voi kehitysjakson tavoitteet jäädä toteuttamatta. Tavoitteiden viivästyessä usein koko projekti viivästyy, jos suunnittelupiikki ei auta kirimään menetettyä aikaa.

ChannelAdvisor on esimerkki yhtiöstä, jonka onnistuneessa ketterässä ohjelmistoprojektissa sallittiin piikit [Ish08]. Projektissa piikkeihin käytetty aika rajattiin. Lisäksi piikeille määriteltiin selkeä hyväksymistavoite, joka piti piikit organisoituina ja tuottavina. Eloranta ja Koskimies eivät erikseen huomioineet artikkelissaan suunnittelupiikkien käyttöä.

4.4 Arkkitehtuurijaksot

Leffingwell esittelee kokonaisten kehitysjaksojen käytön järjestelmän rakenteen suunnitteluun, kehitykseen ja toteuttamiseen [Lef07, s. 209-212]. Tarvittaessa tavallisen kehitysjakson sijasta suoritetaan *arkkitehtuurijakso* (architecture sprint). Arkkitehtuurijaksoa voidaan pitää laajennettuna variaationa suunnittelupiikistä. Etuna on koko kehitysryhmän sitoutuminen toimintaan, jolloin noudatetaan ketterien periaatteiden mukaista luottamusta ryhmään ja sen itseorganisoituvuuteen.

Eloranta ja Koskimies eivät havainneet yhtiöitä, jotka käyttäisivät ohjelmistotuotannossaan tätä käytäntöä. Hieman vastaavaa käyttäytymistä kuitenkin havaittiin yhtiössä, joka oli ottanut käyttöönsä niin kutsutun team sprintin [ElK11]. Team sprintissä asiakas ei valitse kehitysjaksossa toteutettavia vaatimuksia. Valinnat suorittaa kehitysryhmä. Nämä kehitysjaksot ovat olleet ensisijaisesti tapa antaa vapaus kehitysryhmän innovaatiolle ja luovuudelle. Team sprint -kehitysjaksoon valittiin usein työtehtäviä, jotka jäisivät muuten tärkeysjärjestyksen vuoksi kokonaan toteuttamatta. Jos kehitysryhmä ottaisi team sprintissä toteutettavakseen ainoastaan järjestelmän rakenteen kannalta keskeisiä vaatimuksia, olisi team sprint arkkitehtuurijaksoa vastaava jakso.

Arkkitehtuurijaksojen ongelmat ovat yhtäläiset kuin luvun 4.1 Sprint 0:ssa. Arkkitehtuurijakson lopputuloksena on vähäisesti tai ei lainkaan asiakasta hyödyttäviä konkreettisia ominaisuuksia. Asiakas voi myös olla vaikeasti suostuteltavissa ohjelman sisäiseen rakenteeseen keskittyviin tuotantojaksoihin, jotka viivästyttävät ominaisuuksien valmistusta.

4.5 Arkkitehtuuritarinat

Vaatimukset määritellään ketterissä menetelmissä usein käyttäjätarinoina. Konkreettisessa maailmassa ne ovat pieniä lappuja, joihin kirjoitetaan toiminnallisuus ja toiminnallisuuden toteuttamiseen kuluva aika. Aika-arvion tekee kehitysryhmä. Yksittäinen käyttäjätarina pitää määrittelee pienen toiminnallisuuden. Käyttäjätarinoissa on perinteisesti esitetty nimenomaan järjestelmän eri käyttäjien konkreettisia toimintoja. Tarinat pilkotaan edelleen pienemmiksi työtehtäviksi, kun ne otetaan työstettäväksi kehitysjaksossa.

Arkkitehtuuritarinat (architecture stories) tai *kehittäjätarinat* (developer stories) ovat käyttäjätarinoita vastaavia vaatimuksia, mutta niitä kirjaavat asiakkaan sijasta järjestelmää kehittävä ryhmä [JMS06]. Arkkitehtuuritarinat kuvaavat järjestelmän rakenteen vaatimuksia, jotka eivät näy asiakkaalle. Esimerkkeinä arkkitehtuuritarinoista on jonkin järjestelmän komponentin suunnittelu, järjestelmän osien refaktorointi tai suunnittelumallien toteuttaminen järjestelmään.

Arkkitehtuuritarinoiden viemä aika estimoidaan käyttäjätarinoiden tavoin. [JMS06] Ne priorisoidaan samaan työlistaan kuin käyttäjätarinat. Toisinaan suositellaan myös kokonaan omaa työlistaa arkkitehtuuritarinoille [BNO10]. Arkkitehtuuritarinoiden ideana on tuoda esille järjestelmän refaktoroinnin ja suunnittelun konkreettiset vaa-

timukset. Ne toimivat kehitysryhmälle myös muistutuksena ja tietona järjestelmän rakenteesta [JMS06].

Käyttäjätarinoiden aika-arvioiden perusteella pystytään arvioimaan kehitysryhmän työvauhtia (velocity). Ensimmäisen kehitysjakson jälkeen voidaan laskea toteutuneiden käyttäjätarinoiden aika-arviot yhteen, jolloin saadaan tietää ryhmän työvauhti [Kni07]. Työvauhdin avulla arvioidaan, kuinka monta käyttäjätarinaa seuraavassa kehitysjaksossa ehditään toteuttaa. Toteutettavia vaatimuksia valittaessa ryhmän työvauhti voidaan jakaa käyttäjä- ja arkkitehtuuritarinoiden välille [NOS12]. Esimerkiksi käyttäjätarinoiden toteuttamiselle voidaan varata kaksi kolmasosaa ja arkkitehtuuritarinoille yksi kolmasosa ryhmän kokonaistyövauhdista. Lähestymistavan etuna on se, että järjestelmän rakenteelle ja toiminnallisuudelle varataan aikaa tasaisesti jokaiseen kehitysjaksoon. Työvauhtia jakaessa järjestelmän rakenteen vaatima aika ei jää huomioimatta.

Oleellinen osa arkkitehtuuritarinoita on työtehtävien näkyvyys työtiloissa. Ketterissä menetelmissä suosittu väline on Kanban-taulu. Se on valkotaulu, joka jaetaan osiin [Aro01]. Taulun vasemmalle puolelle sijoitetaan kehitysjaksoon valitut vaatimukset ja niistä pilkotut työtehtävät. Seuraavat taulun osat ovat kyseisellä hetkellä työstettävät tehtävät sekä testaukseen ehtineet tehtävät. Aivan taulun oikeassa reunassa ovat testauksen läpäisseet, valmistuneet työtehtävät. Käyttäjätarinoita ja työtehtäviä kuvaavat laput kiinnitetään taululle ja niitä siirretään oikeille paikoilleen kehitysjakson edessä. Lappujen on tarkoitus kulkea taulun vasemmasta reunasta oikeaan.

Ipek Ozkaya ja Robert Nord korostavat arkkitehtuuritarinoiden näkyvyyden tärkeyttä Kanban-taululla [BNO10]. Näkyvyys auttaa erottamaan selkeästi järjestelmän rakenteen kannalta oleelliset työtehtävät. Erottamiseen on muutamia tapoja [NOS12]. Ensimmäinen tapa on jakaa taulu vaakaviivalla kahteen osaan, joista ylempään liitetään tavalliset käyttäjätarinat ja alempaan arkkitehtuuritarinat. Toinen tapa on värikoodata työtehtävät. Esimerkiksi käyttäjätarinat voitaisiin kirjoittaa keltaisille lapuille ja rakenteeseen liittyvät tarinat sinisille.

Jensen ja kumppanit [JMS06] väittävät, ettei kehittäjätarinat ole ristiriitaisia Extreme Programmingin käytänteiden kanssa. Nord ja Ozkaya suosittelevat myös arkkitehtuuritarinoita, joita voidaan työstää useassa kehitysjaksossa [NOS12]. Suosituimpien ketterien ohjelmistokehityksen metodiikkojen määritelmä käyttäjätarinoille on kuitenkin se, että niiden tulisi olla toteutettavissa yhden kehitysjakson aikana. Tarpeen vaatiessa käyttäjätarinoita tulisi pilkkoa pienemmiksi. Sama ohje on sovellet-

tavissa myös arkkitehtuuritarinoihin.

Arkkitehtuuritarinat eivät ole ketterän ohjelmistokehityksen julistuksen vastaisia. Ne saattavat jopa auttaa kiinnittämään huomiota jatkuvaan tekniseen erinomaisuuteen sekä yrityshenkilöstön ja ohjelmistokehittäjien yhteistyöhön, joita ketterät periaatteet [BBB01b] vaalivat. Arkkitehtuuritarinat voivat selkeyttää myös asiakkaalle sitä, millaista lisätyötä hänen vaatimuksensa tuottavat. Teknisesti taitamatonta asiakasta voi olla hankala saada sopimaan menetelmästä, jossa työlistaan otetaan myös asiakkaan kannalta merkityksettömiä työtehtäviä, tai jaksojen työtehtävät valittaisiin tasaisesti käyttäjä- ja arkkitehtuuritarinoista.

4.6 Yleistä käytännöistä

Käytäntöjen vertailu tai paremmuusjärjestyksen tekeminen on haastavaa, ellei mahdotonta. Yhtä oikeaa valintaa ei ole. Käytäntöjen käyttöä tai niiden välttämistä vertailevaa empiiristä tutkimusta ei ole. Elorannan ja Koskimiehen tutkimuksissa käytänteitä hyödyntäneet projektiryhmät olivat myös pääosin tyytyväisiä valintoihinsa. Tutkimukset eivät kuitenkaan maininneet kehitysryhmistä, jotka olisivat kokeilleet useita vaihtoehtoisia tapoja. Käytäntöjen käsittelyn yhteydessä mainittiin lisäksi joitakin yksittäisiä projekteja, jotka käyttivät tapoja onnistuneesti. Vastaavasti on varmasti yhtä paljon tapauksia, joissa projektit ovat onnistuneet puhtaalla ketterällä ohjelmistokehityksellä.

Käytännöt eivät useinkaan ole keskenään ristiriidassa. Projektissa olisi mahdollista hyödyntää montaa käytäntöä. Esimerkiksi kehitysryhmästä eriytetty arkkitehtuuriryhmä voisi todistaa suunnittelemansa rakenteen toimivaksi prototyypein ja ohjelmakoodein, mutta siirtää vastuun arkkitehtuurin toteuttamisesta kehitysryhmälle arkkitehtuuritarinoiden muodossa. Sprint 0:lla aloittanut ryhmä voisi tukeutua suunnittelupiikkeihin, jos alustavaa arkkitehtuuria täytyy muuttaa.

Käytäntöjen yhdistelemisestä ei kuitenkaan ole tutkimuksia. Liian monen käytännön sisällyttäminen ohjelmistotuotantoprosessiin ei välttämättä ole enää lainkaan ketterän ohjelmistokehityksen periaatteiden mukainen, vaan ketterien periaatteiden hukkuessa se voi olla enemmänkin arkkitehtuuriorientoitunut prosessimalli. Vaihtoehtoisesti ketterien menetelmien ja käytänteiden yhdistelmästä voisi luoda täysin uudenlaisen prosessimallin. Yhdistelevä tuotantoprosessi voi myös lukeutua Lean-ohjelmistotuotannoksi, jos se noudattaa sopivasti Lean-ohjelmistotuotannon periaatteita.

5 Yhteenveto

Arkkitehtuurivetoinen ohjelmistokehitys ja ketterä ohjelmistokehitys nähdään usein vaihtoehtoisina tai ristiriitaisina lähestymistapoina. Ketterä ohjelmistokehitys luottaa siihen, että sopiva järjestelmän rakenne ja selkeät osien yhteydet kumpuavat kehittäessä ohjelmaa vastaamaan niitä vaatimuksia, joita sen hetkiseen kehitysiteraatioon on valittu. Usein sopivaan arkkitehtuuriin ohjeistetaan yksinkertaisuuden vaalimisella ja turhan työn välttämisellä. Osa ketterän kehityksen metodiikoista, esimerkiksi Scrum, ei ota järjestelmän rakenteen suunnitteluun lainkaan kantaa.

Arkkitehtuuriyhteisön mielestä ketterän kehityksen suhde järjestelmän rakenteen suunnitteluun on haastava. Arkkitehtuuriyhteisö vaatiikin ketterältä prosessilta aikaa myös järjestelmän rakenteen määrittelyyn, suunnitteluun ja toteuttamiseen. Ketterä kehitys ei huomioi riittävästi järjestelmän ei-toiminnallisia vaatimuksia. Järjestelmän laajetessa vaaditaan huomattavasti refaktorointia, jotta järjestelmä pysyy ylläpidettävänä ja riittävän laajennettavana.

Ilman suunnitteluvaihetta kehitysympäristön ja kohdelaitteiston rajoitukset eivät aina ole selvät. Järjestelmän rakenteen kehitys ja suunnittelu on haastavaa erityisesti kokemattomammille ohjelmistokehittäjille, jotka eivät tunne kohdelaitteistoa tai kaikkia yleisesti käytettyjä suunnittelumalleja. Projektin aloitusta edeltävän suunnitteluvaiheen puuttuminen on suurimpia syitä olla vaihtamatta ketteriin ohjelmistokehitysmetodeihin.

Ketterän ohjelmistokehityksen periaatteisiin kuuluu kehitysryhmän mukautuvuus. Käytännön työelämässä kehitysryhmät ovatkin luoneet käytänteitä, joilla arkkitehtuurin suunnittelun ja toteutuksen voi liittää osaksi ketterää ohjelmistokehitystä. Niistä osa tapahtuu varsinaisten ketterien kehitysjaksojen ulkopuolella, osa taas on toteutettavissa itse iteraatioissa.

Sprint 0 on perinteistä suunnitteluvaihetta korvaava jakso, joka on usein tavallisen kehitysjakson mittainen. Siinä voidaan suunnitella ohjelmalle rakenne, jonka ei ole tarkoitus olla lopullinen. Rakennetta muutetaan tarpeen vaatiessa, mutta se toimii alustavana runkona projektin alkaessa. Rakenteen suunnittelee ja toteuttaa yleensä samat ohjelmiston kehittäjät, jotka toteuttavat myös järjestelmän toiminnallisuudet.

Arkkitehtuuri eriytettynä prosessina siirtää vastuun järjestelmän rakenteesta erilliselle arkkitehtiryhmälle. Se voi noudattaa omaa prosessimalliaan, mutta välietapit on synkronoitava toiminnallisuuksista vastaavan ryhmän kanssa. Kommunikoinnin helpottamiseksi arkkitehtiryhmään kuuluu yleensä muutama henkilö kehitysryhmäs-

tä. Arkkitehtiryhmä voi toteuttaa myös useita vaihtoehtoisia arkkitehtuureja, joista kehitysryhmä valitsee sopivimman.

Suunnittelupiikit ovat kehitysjakson sisäisiä lyhyitä ajanjaksoja, joissa suunnitellaan ja kokeillaan vaihtoehtoja vaativille teknisille tai rakenteellisille ongelmille. Piikeissä voidaan testata useita vaihtoehtoisia järjestelmän rakenteita ja teknologioita pienten esimerkkiohjelmien avulla. Yleensä suunnittelupiikit rajataan ajallisesti. Aikarajat vaihtelevat suuresti, kahdesta päivästä jopa viikkoihin. Suunnittelupiikkeihin osallistuu usein vain muutama kehitysryhmän jäsen, jotta piikit eivät vaadi liikaa resursseja.

Arkkitehtuurijaksot ovat jatkettu varitaatio suunnittelupiikeistä. Arkkitehtuurijaksoissa suunnittelemiseen ja testaamiseen osallistuu koko ryhmä. Arkkitehtuurijaksoissa myös yleensä refaktoroidaan testatut rakenteelliset muutokset järjestelmään. Arkkitehtuurijaksoja järjestetään tarpeen vaatiessa normaalin kehitysjakson sijasta.

Arkkitehtuuritarinat ovat ei-toiminnallisten vaatimusten vastine toiminnallisille vaatimuksille eli käyttäjätarinoille. Toiminnallisuuden sijasta arkkitehtuuritarinoissa ilmaistaan järjestelmän teknisiä ja rakenteellisia vaatimuksia. Arkkitehtuuritarinat auttavat varaamaan riittävästi aikaa järjestelmän haasteille, jotka eivät näy asiakkaalle. Tarinoihin liittyy oleellisesti niiden näkyvyys ja erottaminen normaaleista käyttäjätarinoista. Usein käytetty menetelmä on pitää työtehtävistä kirjaa Kanbantaululla, johon arkkitehtuuri- ja käyttäjätarinoista kirjoitetut laput liitetään. Arkkitehtuuritarinoiden näkyvyyttä voi parantaa esimerkiksi arkkitehtuuritarinoille varatulla omalla työlistalla tai värikoodaamalla arkkitehtuuritarinoiden laput.

Ei ole olemassa yhtä ohjetta sille, mikä käytänteistä on paras tai hyödyllisin. Käytänteet ovat usein jonkin verran ristiriidassa suosituimpien ketterien menetelmien tai ketterän ohjelmistokehityksen julistuksen periaatteiden kanssa. Siitäkin huolimatta käytänteitä käyttävät monet ohjelmistoyritykset, jotka ovat olleet pääasiassa tyytyväisiä valintoihinsa. Tieteellistä vertailua käytäntöjen paremmuudesta tai vaikutuksista ketterään ohjelmistokehitykseen ei juuri ole.

Ketterän ohjelmistokehityksen julistus painottaa projektiryhmän ja yhtiön muutosvalmiutta ja sopeutuvuutta. Käytännöt voivatkin olla suunnitteluorientoituneiden tai isojen yhtiöiden tapa sopeutua prosessimallin muutoksiin ja ketterän ohjelmistokehityksen haasteisiin.

Lähteet

ABK10 Abrahamsson, P., Babar, M. A. ja Kruchten, P., Agility and architecture: Can they coexist? *IEEE Softw.*, 27,2(2010), sivut 16–22.

Arc10 Sprint zero - where your software project begins, 2010. URL `http://www.architech.ca/wp-content/uploads/2010/09/Sprint-Zero-Where-Your-Software-Project-Begins.pdf`. Haettu 3.12.2012.

Aro01 Arora, N., *Kanban Guide: Demand Scheduling for Lean Manufacturing.* Add ValueConsulting Inc., 2001.

BeC04 Beck, K. ja Andres, C., *Extreme Programming Explained: Embrace Change (2nd Edition).* Addison-Wesley Professional, 2004.

BBB01a Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. ja Thomas, D., Manifesto for agile software development, 2001. URL `http://www.agilemanifesto.org/`. Haettu 3.12.2012.

BBB01b Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. ja Thomas, D., Principles behind the agile manifesto, 2001. URL `http://agilemanifesto.org/principles.html`. Haettu 3.12.2012.

BeF00 Beck, K. ja Fowler, M., *Planning Extreme Programming.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, ensimmäinen painos, 2000.

BNO10 Brown, N., Nord, R. ja Ozkaya, I., Enabling agility through architecture.

Boe02 Boehm, B., Get ready for agile methods, with care. *Computer*, 35,1(2002), sivut 64–69.

Con01 Constantine, L., Methodological agility. *Software Development.*

ElK11 Eloranta, V.-P. ja Koskimies, K., Sulava scrum survey. Tekninen raportti, Department of Software Systems, 09 2011.

ElK13 Eloranta, V.-P. ja Koskimies, K., Software architecture practices in agile enterprises. sivut 230–249.

Fow01 Fowler, M., Extreme programming examined. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001, luku Is design dead?, sivut 3–17.

Iee00 Ieee recommended practice for architectural description of software-intensive systems. *IEEE Std 1471-2000*, sivut 1–23.

Ish08 Isham, M., Agile architecture is possible – you first have to believe! *Proceedings of the Agile 2008 Conference*. IEEE Computer Society, 2008, sivut 484–489.

JMS06 Jensen, R. N., Møller, T., Sönder, P. ja Tjørnehøj, G., Architecture and design in extreme programming; introducing "developer stories". *Proceedings of the 7th international conference on Extreme Programming and Agile Processes in Software Engineering*, XP'06, Berlin, Heidelberg, 2006, Springer-Verlag, sivut 133–142.

Kni07 Kniberg, H., *Scrum and XP from the Trenches: Enterprise Software Development*. Lulu.com, 2007.

Kru10 Kruchten, P., Software architecture and agile software development: a clash of two cultures? *ICSE (2)*, 2010, sivut 497–498.

Lar03 Larman, C., *Agile and Iterative Development: A Manager's Guide*. Pearson Education, 2003.

Lef07 Leffingwell, D., *Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series)*. Addison-Wesley Professional, 2007.

NOS12 Nord, R. L., Ozkaya, I. ja Sangwan, R. S., Making architecture visible to improve flow management in lean software development. *IEEE Software*, 29,5(2012), sivut 33–39.

Raw07 Rawsthorne, D., 2007. URL `http://blogs.collab.net/agile/sprint-zero`. Haettu 3.12.2012.

ScB01 Schwaber, K. ja Beedle, M., *Agile Software Development with Scrum.* Prentice Hall PTR, Upper Saddle River, NJ, USA, ensimmäinen painos, 2001.

Sch04 Schwaber, K., *Agile Project Management With Scrum.* Microsoft Press, Redmond, WA, USA, 2004.

Sch10 Schwaber, K., 2010. URL `http://scrumorg.squarespace.com/assessmentdiscussion/post/1317787`. Keskustelua Scrumin käytänteistä, Haettu 3.12.2012.

Ver11 VersionOne, 6th annual state of agile survey, 2011. URL `http://www.versionone.com/pdf/2011_State_of_Agile_Development_Survey_Results.pdf`. Haettu 3.12.2012.

Wel99 Wells, D., Create a spike solution, 1999. URL `http://www.extremeprogramming.org/rules/spike.html`. Haettu 3.12.2012.