

TEKNILLINEN KORKEAKOULU
Elektroniikan, tietoliikenteen ja automaation tiedekunta

Reko Jokelainen

VAATIMUSTEN JÄLJITETTÄVYYS SCRUM-PROSESSISSA

Kandidaatintyö

Espoo 8.12.2009

Työn ohjaaja:

TkL Kristian Rautiainen

Tekijä: Reko Jokelainen		
Työn nimi: Vaatimusten jäljitettävyys Scrum-prosessissa		
Päivämäärä: 8.12.2009	Kieli: Suomi	Sivumäärä: 5+18
Tutkinto-ohjelma: Automaatio- ja systeemitekniikan laitos		
Vastuupettaja: TkT Pekka Forsman		
Ohjaaja: TkL Kristian Rautiainen		
<p>Vaatimusten jäljitettävyys on tunnustettu hyväksi käytännöksi perinteisten, suunnitelmavetoisten ohjelmistokehitysmenetelmien puolella. Ketterissä menetelmissä jäljitettävyys ei ole saanut yhtä vakiintunutta asemaa. Tässä työssä käsitellään jäljitettävyyttä perinteisissä menetelmissä sekä käytäntöjen soveltamista ketterissä menetelmissä.</p> <p>Monet ketterien menetelmien kannattajat väittävät jäljitettävyyttä turhaksi vedoten siihen, että kehityksen tavoite on tuottaa asiakkaalle suoraa arvoa, jota kyseisellä dokumentaatiolla on vaikea todistaa olevan. Toisaalta jäljitettävyydellä voidaan ylläpitää ketterissä menetelmissä läpinäkyvyyttä. Läpinäkyvyys antaa asiakkaalle paremman mahdollisuuden seurata tuotekehityksen etenemistä ja näin ollen tuottaa arvoa asiakkaan näkökulmasta.</p> <p>Joillakin aloilla, kuten ilmailualalla, vaatimusten jäljitettävyys on pakollinen käytäntö tuotekehitysorganisaatioissa, ja siksi ketteriä menetelmiä ei ole perinteisesti käytetty. Jotta näillä aloilla ketterää kehitystä voidaan käyttää, täytyy jäljitettävyyskäytännöt sovittaa niiden ajatusmalliin.</p> <p>Tutkimuksessa käydään läpi perinteisten menetelmien soveltuvuutta ketterään maailmaan. Lisäksi tutustutaan testivetoisen kehityksen mahdollistamaan automaattiseen jäljitettävyYTEEN.</p>		
Avainsanat: ketterät menetelmät, jäljitettävyys, hierarkia, vaatimusten hallinta		

Sisältö

Tiivistelmä	ii
Sisällysluettelo	iii
Sanasto ja lyhenteet	v
1 Johdanto	1
2 Scrum-prosessi	3
2.1 Yleistä Scrumista	3
2.2 Scrum-ryhmä	3
2.3 Asiakkaan osuus Scrumissa	4
2.4 Scrum-sykli	5
2.4.1 Sprintit	5
2.4.2 Scrum-tapaaminen	5
2.4.3 Iteraation suunnittelu	5
2.5 Vaatimustenhallinta Scrumissa	6
2.5.1 Tuotteen työlista	6
2.5.2 Iteraation työlista	6
3 Vaatimusten jäljitettävyys perinteisissä menetelmissä	8
3.1 Vaatimusten hallinta	8
3.2 Jäljitettävyyden tyypit	8
3.2.1 Vaatimusten jäljitettävyyden määritelmä	8
3.2.2 Jäljitettävyys etukäteen	9
3.2.3 Jäljitettävyys jälkikäteen	9
3.2.4 Muutosten hallinta	9
3.3 Motivaatio	9
3.4 Käytännöt	10
3.4.1 Vaatimusten pilkkominen	10
3.4.2 Hierarkia	10
3.4.3 Jäljitettävyysmatriisi	10
4 Jäljitettävyys ketterissä menetelmissä	12

4.1	Onko jäljitettävyys tärkeää?	12
4.2	Jäljitettävyyden soveltaminen	12
4.3	Käytäntöjen sopivuus ketteriin menetelmiin	13
4.3.1	Vaatumusten pilkkominen	13
4.3.2	Hierarkia	13
4.3.3	Jäljitettävyysmatriisi	14
4.3.4	Testivetoinen jäljitettävyys	14
4.4	Ketteryys jäljitettävyyttä vaativassa yrityksessä	14
5	Yhteenveto	16
	Viitteet	17

Sanasto ja lyhenteet

Käsitteet

Backlog	katso <i>työlista</i>
Käyttäjätarina	käyttäjätarina on Scrumissa ja XP:ssa asiakkaan vaatimusten dokumentointitapa
Scrum	yksi suosituimmista ketterän ohjelmistokehityksen prosessimalleista.
Sprint	Scrum-iteraatio
Tarina	katso <i>käyttäjätarina</i>
Tuoteomistaja	Scrumin asiakasta vastaava edustaja
Työlista	työlistalla tarkoitetaan Scrumissa listaa tuotteen tai iteraation vaatimuksista.
Vesiputous	perinteinen ohjelmistokehitysmalli

Lyhenteet

CMMI	Capability Maturity Model Integration
IEEE	Institute of Electrical and Electronics Engineers
XP	eXtreme Programming

1 Johdanto

Vaatimusten hallinta on yksi oleellisimmista ohjelmistotuotannon haasteista. Eräs vaatimusten hallinnan osa-alueista on jäljitettävyyys. Vaatimusten jäljitettävyydellä on monia määritelmiä, yksi niistä on IEEE:n määritelmä:

Vaatimusmäärittely on jäljitettävä, jos sen jokaisen vaatimuksen alkuperä on selvä, ja se helpottaa vaatimukseen viittaamisen tulevassa kehityksessä tai lisädokumentaatioissa [7]. (*suomentanut kirjoittaja*)

Jäljitettävyyys on perinteisissä menetelmissä usein hyvin oleellista. Scrumin ja muiden ketterien menetelmien maailmassa jäljitettävyyys on kuitenkin jäänyt taka-alalle tai hylätty kokonaan. Tässä työssä tutkitaan jäljitettävyyttä perinteisissä menetelmissä. Työssä otetaan kantaa perinteisten menetelmien jäljitettävyysskäytäntöjen sopevuuteen ketteriin menetelmiin. Lisäksi pohditaan testivetoisen kehityksen suomaa mahdollisuutta automaattisen jäljitettävyyden luontiin.

Scrum-prosessimalli on yksi suosituimmista ketteristä menetelmistä. Scrum on projektinhallintaan suunnattu iteratiivinen ja inkrementaalinen prosessiviitekehys. Se perustuu projektin jakamiseen pienempiin iteraatioihin eli *sprinteihin*, joiden aikana vaatimukset ovat lukittuja.

Vaatimusten jäljitettävyyys on perinteisesti suhteellisen raskas prosessi, joka vaatii alusta pitäen runsasta ja kattavaa dokumentaatiota. Tämä on ristiriidassa ketterien menetelmien kanssa, joissa pyritään minimoimaan dokumentaatio ja keskittymään toimivaan ohjelmistoon.

Perinteisiä ohjelmistokehitysmalleja suosivat väittävät usein, että ketterissä menetelmissä on unohdettu vuosikausien kokemus vaatimusten hallinnan alalta ja jätetty sen tuoma viisaus sikseen, kun vaatimusten selvittämiseen ja määrittelyyn ei käytetä alkuunsa runsaasti aikaa. Tämä johtuu menetelmien eri lähtökohdista: Scrumissa tuotteen vaatimukset elävät kehityksen myötä, kun taas perinteisissä menetelmissä kaikki asiakkaan vaatimukset – myös ne, joita hän ei itse tiedä tarvitsevansa – on tarkoitus saada dokumentoitua projektin alkuvaiheessa.

Perinteisissä menetelmissä vaatimusten jäljitettävyyttä pidetään hyvin olennaisena osana kehitysprosessia. Ketterien menetelmien suosijat väittävät, että jäljitettävyydestä voi olla jopa haittaa liiallisen dokumentaation vuoksi. Työssä pyritään vastaamaan, miksi jäljitettävyyys on tärkeää perinteisissä menetelmissä, ja onko se tärkeää myös Scrum-maailmassa.

Työssä pohditaan aluksi Scrumia sekä vaatimustenhallintaa ketterissä menetelmissä. Toiseksi tutkitaan perinteisten menetelmien vaatimusten jäljitettävyyttä. Näiden välille pyritään löytämään yhteys, joka selventää vaatimusten jäljitettävyyden asemaa Scrum-prosessin kannalta.

Toisessa luvussa kuvaillaan Scrum-prosessi yleisellä tasolla. Kolmannessa luvussa selvitetään jäljitettävyyttä perinteisissä ohjelmistokehitysmenetelmissä. Neljäs luku käsittelee vaatimusten jäljitettävyyttä ketterissä menetelmissä.

Tutkimus on tehty kirjallisuustutkimuksena käyttäen lähteenä lähinnä internetin kautta löytyneitä viitteitä sekä painettua kirjallisuutta. Tietoa on haettu käyttäen muun muassa Googlen scholar-palvelua, Teknillisen Korkeakoulun tarjoamaa Nelli-portaalia sekä Scopus-viitetietokantaa. Vaatimusten jäljitettävyys ketterissä menetelmissä on suhteellisen uusi tutkimuksen ala, joten teollisuudesta ei juurikaan vielä löydy tieteellistä tutkimusta aiheesta. Tutkimukseen sisältyy kuvaus ketterien menetelmien soveltamisesta eräässä belgialaisessa ohjelmistoyrityksessä, jolta vaaditaan jäljitettävyttä.

2 Scrum-prosessi

Tämä luku kertoo hyvin yleisesti käytössä olevasta ketterästä ohjelmistokehitysprosessimallista, Scrumista. Aluksi käsitellään Scrumin taustaa, Scrum-ryhmää sekä ryhmän tärkeimpiä sidosryhmiä. Myöhemmin käydään läpi Scrum-sykli päivittäis-
tasolta iteraatiotasolle. Lopuksi esitetään Scrumin vaatimustenhallintakäytäntöjä.

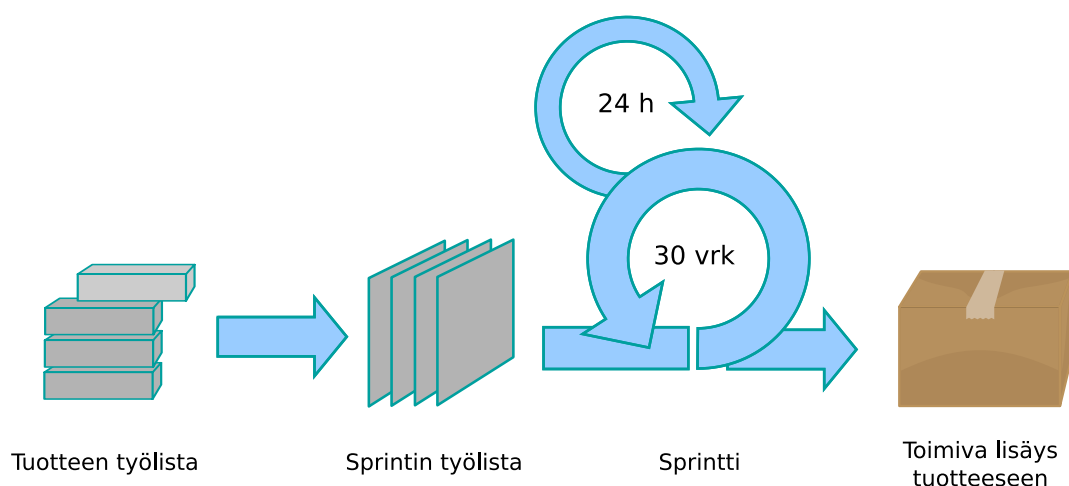
2.1 Yleistä Scrumista

Scrum on eräs vakiintuneimmista ketterän ohjelmistokehityksen prosessimalleista. Scrumin ovat kuvanneet Ken Schwaber ja Mike Beedle vuonna 2001 kirjassaan *Agile Software Development with Scrum*[19]. Tämän jälkeen prosessista ja sen soveltamisesta on julkaistu useita artikkeleita ja kirjoja.

Yksi Scrumin pääpiirteistä on sen iteratiivinen ja inkrementaalinen luonne. Ohjelmistotuote luodaan pala kerrallaan lyhyissä iteraatioissa, ja asiakkaalle pyritään tuottamaan arvoa niin aikaisin kuin vain suinkin mahdollista. Koska ohjelmisto rakennetaan pienissä pätkissä, tehdään myös kaikkia perinteisten menetelmien aktiviteetteja samanaikaisesti – testaus, suunnittelu ja toteutus tapahtuvat kaikki iteraation sisällä. [17]

2.2 Scrum-ryhmä

Perinteisistä ohjelmistokehitysmenetelmistä eroten Scrum-ryhmä koostuu usean eri alan osaajista. Suunnittelu-, testaus- ja kehitysryhmiä ei ole eroteltu, vaan ryhmä



Kuva 1: Scrum-prosessi. Lähde: *Wikimedia Commons*

tekee nämä kaikki aktiviteetit itse [10].

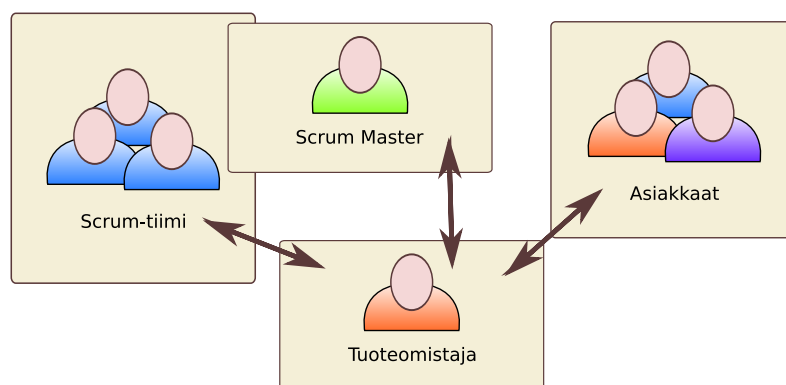
Scrum-ryhmän kanssa läheisessä yhteistyössä toimivat Scrum-mestari (*engl. Scrum master*) ja tuoteomistaja (*engl. product owner*). Scrum-mestarin työkuva voi verrata jossain määrin projektipäällikköön. Hänen tehtävänsä on mahdollistaa ryhmän työskentely ja poistaa työskentelyä haittaavia esteitä. Scrum-mestari on vastuussa prosessin noudattamisesta. Tuoteomistaja on taas eräänlainen asiakasrooli. [10]

2.3 Asiakkaan osuus Scrumissa

Scrumissa asiakassuhde poikkeaa perinteisistä menetelmistä. Perinteisen asiakassuhteen asemesta tuotteen vaatimuksista vastaa tuoteomistaja. Tuoteomistajana voi toimia joko asiakkaan edustaja tai tuottajaorganisaation jäsen. Tuoteomistajana toimiminen vaatii runsaasti aikaa ja resursseja, eikä tämä tapa siksi sovellu kaikille asiakkaille. Hänen täytyy lisäksi tuntea Scrum-prosessi hyvin sekä sopeutua noudattamaan sitä.

Kaikilla asiakkaan edustajilla ei ole mahdollisuutta osallistua Scrum-prosessin vaatimalla panoksella. Näissä tapauksissa voidaan valita edustaja tuottajaorganisaatiosta, joka vastaa sisäisesti tuotteen työstä ja kommunikoi asiakasorganisaation suuntaan.

On käytännössä mahdotonta löytää henkilöä, joka ymmärtää tuotteen kaikilta asiakailta tulevat vaatimukset sekä kaikkien sidosryhmien tarpeet. Scrumiinkin joudutaan lainaamaan vaatimusten hallinnan käytäntöjä perinteisistä menetelmistä [14]. Nämä käytännöt kulminoituvat tuoteomistajaan.



Kuva 2: Sidosryhmät Scrumissa

2.4 Scrum-sykli

Ohjelmistokehitysprosessi jaetaan usein neljään vaiheeseen: analyysiin, suunnitteluun, toteutukseen ja testaukseen. Perinteisessä ohjelmistokehityksessä nämä vaiheet tulevat peräkkäin, kun taas Scrumissa nämä on sijoitettu toistuvien iteraatioiden eli *sprinttien* sisään.

2.4.1 Sprintit

Scrumissa tuotteiden kehitys on jaettu lyhyisiin, yleensä yhdestä neljään viikkoa kestäviin iteraatioihin eli *sprintteihin* [17]. Ryhmä sitoutuu iteraation ajaksi kehittämään iteraatioon valittuja vaatimuksia. Jokaisen iteraation tavoitteena on mahdollisesti julkaistava tuotteen osa. Tämä ei kuitenkaan tarkoita, että jokaisen sprintin lopuksi tulisi tehdä julkinen julkaisu tuotteesta. Julkinen julkaisu voidaan tehdä esimerkiksi kolmen kuukauden välein – kahden viikon sprinteillä käytetään viisi sprinttiä kehitykseen ja yksi sprintti julkaisun edellyttämään laadunvarmistukseen ja korjauksiin [10].

Iteraatio alkaa iteraation suunnittelukokouksella, jossa valitaan iteraation työlista ja arvioidaan toteutettaviksi valittujen tarinoiden koko. Iteraation jälkeen pidetään katsaus (*engl. sprint review*), jossa esitellään asiakkaalle ja tuoteomistajalle työn tulokset [17]. Tätä katsausta kutsutaan myös *demoksi*.

2.4.2 Scrum-tapaaminen

Scrum-ryhmä kokoontuu kerran päivässä kertomaan työnsä edistymistä. Tämä niin sanottu Scrum-tapaaminen (*engl. daily scrum*) järjestetään joka päivä samassa paikassa samaan aikaan. Kokous pyritään pitämään lyhenä, korkeintaan 15 minuutin pituisena. Kokouksessa käydään jokaisen ryhmän jäsenen kohdalta läpi kolme asiaa:

1. mitä on tehnyt edellisen Scrum-tapaamisen jälkeen
2. mitä tulee tekemään ennen seuraavaa tapaamista
3. onko matkan varrella ilmennyt ongelmia. [20]

Kokouksen tavoitteena on lisätä kommunikaatiota ja ryhmän jäsenten tietoisuutta toistensa tekemisistä. Kehittäjien ongelmat myös havaitaan ajoissa – jos joku tekee samaa työtehtävää useamman päivän, on tyypillisesti jotain vialla. Scrumkokouksessa myös sovitaan tapaamiset, jotka ryhmä tarvitsee päästäkseen tavoitteeseensa [18].

2.4.3 Iteraation suunnittelu

Ennen jokaisen iteraation aloittamista pidetään iteraation suunnittelukokous. Kokoukseen osallistuvat ainakin tuoteomistaja, Scrum-mestari sekä ryhmä. Kokoukses-

sa arvioidaan tuotteen työlistan (*engl. product backlog*) vaatimuksia aloittaen korkeimman prioriteetin vaatimuksista. Vaatimuksille annetaan pistearvio niin sanoituissa *tarinapisteissä*, joilla kuvataan tietyn vaatimuksen suhteellista kokoa. Kun vaatimuksia on arvioitu riittävän pitkälle, Scrum-mestari kertoo, että ryhmä ei pysty tekemään enempää yhden iteraation aikana. Näistä vaatimuksista muodostuu iteraation työlista.

Vaatimuksia otetaan iteraatioon yhteenlasketuiden tarinapisteiden perusteella noin sama määrä kuin mitä edellisessä iteraatiossa saatiin tehtyä. Tätä metriikkaa kutsutaan ryhmän nopeudeksi (*engl. velocity*).

2.5 Vaatimustenhallinta Scrumissa

Vaatimuksia hallitaan Scrumissa pitämällä yllä niin sanottua *backlogia* eli työlistaa. Työlista on yksitasoinen, priorisoitu lista kaikista tuotteen vaatimuksista. Työlistaa päivitetään koko kehityksen ajan. Ei ole tarkoitus, että kaikki tuotteen toiminnalliset vaatimukset olisi kirjattuna työlistaan projektin alkaessa.

Scrumissa vaatimukset kirjataan niin sanottuina käyttäjätarinoina (*engl. user story*). Käyttäjätarina noudattaa muotoa “*Käyttäjänä haluan toiminnallisuuden, josta saavutettava hyöty*” [10]. Käyttäjätarinoille annetaan iteraation suunnittelussa niiden suhteellista kokoa kuvastava tarinapiste-arvio. Käyttäjätarinat ovat alunperin XP:sta lainattu käytäntö, mutta niitä koulutetaan nykyään Scrumin yhteydessä [10].

2.5.1 Tuotteen työlista

Jokaisella tuotteella on työlista (*engl. product backlog*), jota ylläpitää tuoteomistaja. Tuoteomistaja on vastuussa siitä, että lista on jatkuvasti ajan tasalla ja prioriteettijärjestyksessä. Työlistaa päivitetään koko kehitysprosessin ajan vastaamaan asiakkaan muuttuneita tarpeita.

Tuotteen työlista sisältää kaikki tuotteen tietyllä hetkellä tiedossa olevat käyttäjätarinat priorisoidussa järjestyksessä. Lista elää sitä mukaa, kun uusia vaatimuksia ilmaantuu, asiakkaan toiveet muuttuvat tai vanhat tarinat koetaan turhiksi. Listasta arvioidaan kärkipään tarinoiden työmäärä, koska ne ovat oleellimmat ymmärtää tuotteen kehityksen kannalta. Tuotteen työlista on tasainen, priorisoitu lista, jossa ei ole hierarkiaa.

2.5.2 Iteraation työlista

Iteraation työlista on yhden sprintin työlista, joka koostuu yleensä tuotteen työlistan korkeimman prioriteetin käyttäjätarinoista. Iteraation työlista muodostetaan iteraation suunnittelukokouksessa, jossa vaatimukset jäädytetään sprintin ajaksi. Tavoitteena on taata Scrum-ryhmälle työrauha estämällä sprintin aikaiset muutokset vaatimuksiin.

Suunnitellessa iteraatiota ryhmä jakaa tarinoina esitetyt vaatimukset ja käyttäjän tarpeet pienempiin, konkreettisiin työtehtäviin, joita yksittäinen ryhmän jäsen voi tehdä. Jokaisen tehtävän tulisi olla niin pieni, että sen voi saada valmiiksi kahden Scrum-kokouksen välissä.

3 Vaatimusten jäljitettävyys perinteisissä menetelmissä

Vaatimusten jäljitettävyys on eräs perinteisen vaatimusten hallinnan suurista osa-alueista. Tässä luvussa käsitellään jäljitettävyyttä perinteisten menetelmien, kuten vesiputouksen, kannalta. Aluksi kerrotaan vaatimusten hallinnasta yleisesti, minkä jälkeen käsitellään vaatimusten jäljitettävyyden eri osa-alueita. Lopuksi esitellään motivaatio ja käytäntöjä perinteiseen vaatimusten jäljitettävyyteen.

3.1 Vaatimusten hallinta

Vaatimusten hallinnan tavoitteena on luoda ymmärrys siitä, mitä ollaan tekemässä, ennen tuotteen kehittämisen aloittamista, jotta vältetään kalliilta muutostöiltä [11]. Perinteisissä ohjelmistokehitysmenetelmissä, kuten vesiputous-mallissa, vaatimusten hallinta perustuu kahteen oleelliseen olettamukseen:

- tuotteen vaatimukset on mahdollista määritellä etukäteen ennen järjestelmän toteutuksen aloitusta [11].
- mitä myöhemmin virheitä löydetään, sitä kalliimpaa niiden korjaaminen on [3]

Ensimmäinen ei kovinkaan usein pidä paikkaansa – asiakkaan tarpeet muuttuvat usein myöhempien vaiheiden aikana. Tämän vuoksi vaatimuksista pidetään kirjaa ja jäljitettävyyttä tarkemmista spesifikaatioista alkuperäiseen vaatimukseen on syytä pitää yllä. Vaatimusten muuttumisen varalta on perinteisissä menetelmissä varauduttu muutoksen hallinnan menetelmillä.

3.2 Jäljitettävyyden tyypit

3.2.1 Vaatimusten jäljitettävyyden määritelmä

IEEE:n (*Institute of Electrical and Electronics Engineers*) standardi määrittelee vaatimusten jäljitettävyyden seuraavasti:

Vaatimusmäärittely on jäljitettävä, jos sen jokaisen vaatimuksen alkuperä on selvä, ja se helpottaa vaatimukseen viittaamisen tulevassa kehityksessä tai lisädokumentaatioissa. [7, s. 8]. (*suomentanut kirjoittaja*)

Artikkelissaan Finkelstein ja Gotel ymmärtävät tämän seuraavasti:

Vaatimusten jäljitettävyys viittaa kykyyn kuvailla ja seurata vaatimuksen elinkaarta eteen- ja taaksepäin (eli alkuperästään lähtien, kehityksen ja määrittelyjen kautta, tämänjälkeiseen käyttöönottoon ja käyttöön, ja näiden kaikkien vaiheiden aikaiseen jalostukseen ja iterointiin). [5] (*suomentanut kirjoittaja*)

3.2.2 Jäljitettävyys etukäteen

Etukäteen jäljitettävyys tarkoittaa jäljitettävyyssuhdetta asiakkaan tarpeista kirjallisiin vaatimuksiin [5, 7]. Tämä on oleellinen ongelma ohjelmistokehityksessä, sillä monet vaatimusten hallinnan käytännöt tähtäävät nimenomaan asiakkaan vaatimusten ymmärtämiseen ja kirjaamiseen. Ketterissä menetelmissä pyritään ymmärtämään todellinen asiakkaan tarve, ei niinkään vaatimuksia. Tässä tutkimuksessa jäljitettävyyttä pohditaan lähinnä tältä kannalta.

3.2.3 Jäljitettävyys jälkikäteen

Jäljitettävyydellä jälkikäteen tarkoitetaan sellaista jäljitettävyyssuhdetta, jonka perusteella voidaan osoittaa toteutuksen yhteys tiettyyn vaatimukseen [5, 7]. Jälkikäteisjäljitettävyyden ongelmat voidaan korjata helposti [5].

3.2.4 Muutosten hallinta

Vaatimusten jäljitettävyys yhdistetään usein muutosten hallintaan. Muutosten hallinta käsittelee vaatimuksiin myöhemmin tulevia lisäyksiä, poistoja ja muutoksia sekä prosessia näiden tekemiseen. Jäljitettävyyttä tarvitaan muutosten hallinnassa, jotta pystytään varmistumaan, mihin vaatimukseen kyseinen muutos vaikuttaa [12].

Tässä tutkimuksessa ei käsitellä jäljitettävyyttä tältä näkökannalta, sillä ketterissä menetelmissä pyritään tekemään muutoksen tekeminen helpoksi, eikä sen tulisi näin ollen vaatia raskasta prosessia. Ketterissä menetelmissä vaatimusmäärittelyn toteutumisen täydellisyys ei ole onnistumisen mittari [1, 16].

3.3 Motivaatio

Vaatimusten jäljitettävyys on oleellinen osa perinteistä vaatimustenhallintaa. Perinteisissä menetelmissä esimerkiksi ilmailualan standardit ja laajasti käytössä oleva CMMI (*Capability Maturity Model Integration*) vaativat jäljitettävyyttä [15, 22].

Prosessin kannalta on oleellista pitää yllä jäljitettävyyttä, jotta voidaan olla varmoja tuotteen vaatimusten oikeellisuudesta. Jäljitettävyydellä voidaan varmistaa, että vaatimusmäärittelyyn ei tule vaatimuksia muualta kuin asiakkaalta. Tällainen tapaus voi syntyä esimerkiksi, kun projektissa on kehittäjä, joka on tehnyt vastaavanlaisia tuotteita aikaisemminkin, ja hän lainaa vaatimuksen edellisestä tuotteesta. [8]

3.4 Käytännöt

3.4.1 Vaatimusten pilkkominen

Monet asiakkaalta tulevat vaatimukset ovat hyvin laajoja ja epämääräisiä. Tällainen vaatimus voi olla esimerkiksi “Tuotteella pitää voida lähettää sähköpostia”. Vaatimuksena tämä on liian ylimalkainen, eikä sitä suoraan voi lähteä toteuttamaan. Vaatimuksia pilkkotaan pienemmiksi osasiksi, jotta on helpompi ymmärtää, mistä korkeamman tason vaatimus koostuu [8]. Monet alkuperäiset asiakkaan esittämät vaatimukset voivat sisältää myös useita vaatimuksia tuotteelle, esimerkiksi “Tuotteella pitää voida lähettää sähköpostia POP- ja IMAP-protokollien kautta”. Perinteisissä menetelmissä pyritään erottamaan jokaisen käyttäjän vaatimuksen omaksi vaatimukseksi.

Vaarana on liian tarkka pilkkominen toteutustason yksityiskohtiin asti, joilla ei vaatimusmäärittelyn kannalta ole merkitystä ja jotka voivat olla pikemminkin haitallisia. Liian tarkka vaatimusmäärittely johtaa tilanteeseen, jossa hienojakoiset vaatimukset rajoittavat muiden toteutusta. Toisaalta liian korkeatasoinen vaatimusten pilkkominen, esimerkiksi annettaessa tarjousta ohjelmistotuotteesta, vaikeuttaa työmäärän arviointia etenkin suurissa projekteissa. [8]

3.4.2 Hierarkia

Vaatimusten hierarkialla tarkoitetaan puumaista rakennetta ylimmän tason vaatimuksista aina tarkimpaan suunnittelun tasoon. Esimerkki vaatimushierarkiasta on esitetty kuvassa 3 sivulla 11. Mitä korkeammalla tasolla hierarkiassa ollaan, sitä abstraktimpia ovat vaatimukset. Konkreettisimmat vaatimukset ovat hierarkian alimmalla tasolla.

Hierarkia syntyy helposti vaatimusten pilkkomisen sivutuotteena. Hierarkian ongelmana on helppous pilkkoa vaatimuksia liian hienoiksi. Tällöin muodostuu helposti liian moniportainen hierarkia, joka on vaikeaselkoinen [8].

3.4.3 Jäljitettävyyismatriisi

Jäljitettävyyismatriisi on erillinen dokumentti, jossa pidetään kirjaa vaatimusten jäljitettävyyssuhteista. Taulukkoa voidaan käyttää paitsi jäljitettävyyden ylläpitämiseen myös testauksen pohjana: taulukolla voidaan katsoa, että kaikkia vaatimuksia testaa vähintään yksi testi. Taulukossa 1 on esitetty esimerkki perinteisestä jäljitettävyyismatriisista.

Taulukko 1: Esimerkki jäljitettävyyssmatriisista. Muokattu lähteestä: *Wikipedia*

Vaatimuksen tunnus	vaatimuksia testattu	REQ1 UC 1.1	REQ1 UC 1.2	REQ1 UC 2.1	REQ1 UC 2.2
Testitapauksia	10	3	3	2	2
1.1.1	1	X			
1.1.2	2		X	X	
1.1.3	2	X	X		
1.2.1	1				X
1.2.2	2		X		X
1.2.3	2	X		X	

Työajanseuranta

- ↳ Raporttien luominen
 - ↳ Summatiedon näyttäminen
 - ↳ Näyttäminen iteraatiosivulla
 - ↳ Käytetyn ajan kirjaaminen
 - ↳ Kirjaaminen projekteille
 - ↳ Kirjaaminen iteraatioille
 - ↳ Kirjaaminen tuotteille
- Portfolio-näkymä

Kuva 3: Esimerkki vaatimushierarkiasta

4 Jäljitettävyys ketterissä menetelmissä

Jäljitettävyys ei perinteisesti ole kuulunut ketteriin ohjelmistokehitysmenetelmiin. Monet ketterien menetelmien kehittäjät ovat suorastaan jäljitettävyyttä vastaan. Tässä luvussa pohditaan jäljitettävyuden soveltumista ketterien menetelmien maailmaan.

Ensimmäisessä alaluvussa pohditaan jäljitettävyuden tärkeyttä ketterissä menetelmissä. Toinen alaluku käsittelee jäljitettävyuden eri sovelluksia. Kolmannessa esitellään käytäntöjä ja niiden sopivuutta Scrum-ympäristöön. Lopuksi käydään läpi erään yrityksen sovellus jäljitettävyydestä ketterässä kontekstissa.

4.1 Onko jäljitettävyys tärkeää?

Monet ketterien menetelmien kannattajat väittävät, että vaatimusten jäljitettävyyyteen ei ole tarvetta [2, 9]. Perinteisten menetelmien puolella taas jäljitettävyuden on todettu olevan tärkeää [13].

Scrum-prosessin kehittäjä Ken Schwaber väittää, että jäljitettävyyttä ei kannata ylläpitää Scrumissa, koska ketterissä projekteissa vaatimuksia ilmestyy jatkuvasti lisää ja ne kehittyvät alati. Koska markkina-arvo (*engl. business value*) ohjaa kehitystä, aikaisempien vaatimusten puuttuminen nykyisestä toteutuksesta voi yksinkertaisesti heijastaa niiden vähentyntä markkina-arvoa. Projektin menestyksen mittarina toimii paremmin markkina-arvo kuin vaatimusten toteutumisen täydellisyys. [16]

Usein Scrum, kuten muutkin ketterät menetelmät, kuvataan ideaalitulanteessa, jossa tuote on riittävän pieni – yksi ryhmä kehittää yhtä tuotetta. Tässä tilanteessa tuoteomistaja voi vielä ymmärtää ja muistaa koko tuotteen vaatimukset. Kun kehitysorganisaatiota laajennetaan tästä asetelmasta, täytyy ryhmien koordinointiin käyttää perinteisiä menetelmiä [14]. Oleellista on ymmärtää ero jäljitettävyuden ja raskaan jäljitettävyysdokumentaation välillä.

4.2 Jäljitettävyuden soveltaminen

Kuten aikaisemmin luvussa 3.2.4 todettiin, ei Scrumissa tai muissa ketterissä menetelmissä kannata soveltaa jäljitettävyyttä muutosten hallinnan helpottamiseksi. Jäljitettävyuden itseisarvoa ei ole ketterissä menetelmissä perusteltu. Kirjassaan Henrik Kniberg kyseenalaistaa jäljitettävyuden [9], mutta ehdottaa kuitenkin yksinkertaista ratkaisua asiaan, joka ei kuitenkaan palvele tarkoitusta riittävällä tasolla: otetaan valokuva joka päivä ryhmän työlistana toimivasta taulusta.

Perinteisissä menetelmissä jäljitettävyyttä pidetään usein käsin yllä – tosin työkaluja tähänkin tarkoitukseen on ollut jo pitkään. Ketterät ohjelmistokehitysmenetelmät tuovat tähän mielenkiintoisia vaihtoehtoja: esimerkiksi testivetoisen ohjelmistokehityksen myötä voi syntyä kehityksen sivutuotteena jäljitettävyys vaatimuksille [6].

Ketterissä menetelmissä edistymisen seuraaminen on oleellisessa roolissa. *Burndown-*

kaaviolla voidaan seurata sprintin edistymistä [18]. Mitä korkeammalle tasolle mennään, sitä monimutkaisempaa on edistymisen seuranta. Jäljitettävyydellä voidaan mahdollistaa korkean tason vaatimusten edistymistä sitä mukaa, kun siitä pilkotut tarinat ja tehtävät etenevät tai tulevat valmiiksi.

Työkalujen käyttäminen ei ole välttämättä ketterien menetelmien fanaattisimpien kannattajien mielestä hyväksi. Heidän näkökantansa onkin usein rajoittunut aiemmin kuvailtuun yhden ryhmän ideaalitapaukseen. Työkalujen käytöstä voidaan kuitenkin hyötyä esimerkiksi automaattisen jäljitettävyyden generoimisen tukena. Esimerkiksi vaatimushierarkiaa ei ole järkevää pitää yllä pelkillä muistilapuilla. Lisäksi työkalut helpottavat tuotekehityksen läpinäkyvyyttä: asiakkaan tai tuotemistajan ei tarvitse käydä ryhmän työtilassa katsomassa työlistan etenemistä, vaan hän voi tehdä sen omalta päätteeltään.

4.3 Käytäntöjen sopivuus ketteriin menetelmiin

4.3.1 Vaatimusten pilkkominen

Scrumissa on oleellista vaatimusten pilkkominen riittävän pieniksi: iteraatioon voidaan ottaa ainoastaan sellaisia vaatimuksia, joiden työmäärä on arvioitavissa ja toteutettavissa ryhmän aikaisemman nopeuden sallimissa puitteissa. Vaatimuksia, joiden työmäärää ei voida arvioida, kutsutaan *epiikoiksi*. Mitä pienemmäksi toteutettava osa voidaan pilkkoa, sitä pienempi on myös riski, että työmäärän arviointi epäonnistuu.

Pilkkomisella pyritään jakamaan vaatimukset niin pieniksi, että ne on mahdollista toteuttaa yhden sprintin aikana. Näin ollen jokaisessa sprintissä tulisi saada valmiiksi vähintään yksi käyttäjätarina.

Pilkkomisen seurauksena voi syntyä joko hierarkia, jossa alkuperäiset pilkotut vaatimukset ovat mukana, tai tasainen lista, josta alkuperäiset vaatimukset poistetaan. Scrumissa periaatteena on tasainen lista ilman hierarkiasuhteita. Tämä mahdollistaa kaikkien käyttäjätarinoiden yksiselitteisen priorisoinnin, mutta hävittää helposti kaiken jäljitettävyyden alkuperäiseen vaatimukseen.

Liiallinen pilkkominen voi aiheuttaa päämäärän unohtamisen ja johtaa tuotteen arkkitehtuurin epämääräistymiseen [14]. Tämän vuoksi täytyy vaatimuksia pilkottaessa muistaa se asiakkaan tarve, mihin kyseinen vaatimus vastaa. [8]

4.3.2 Hierarkia

Perinteisesti on ajateltu, että ketterät menetelmät toimivat pienissä projekteissa, joissa vaatimusten määrä on rajoitettu [4]. Laajennettaessa projektiorganisaation kokoa täytyy löytää tasapaino ketteryyden ja perinteisten menetelmien välille.

Dean Leffingwell on esittänyt vaatimusten hierarkiaa rajatussa syvyydessä käytettäväksi suurissa ohjelmistoprojekteissa, joita tehdään ketterillä menetelmillä. Hie-

rarkian suunnittelutasot ovat teemat (*investment themes*), epiikat (*epics*), ominaisuudet (*features*), tarinat (*stories*) ja tehtävät (*tasks*). Tämän jaottelun mukaan ryhmäkohtaisten iteraatioiden työlistä koostuu tarinoista. Tarinat jaetaan edelleen iteraation sisällä tehtäviin, jotka ovat konkreettisia tehtäviä, joita ryhmän jäsenten pitää tehdä, jotta tarina valmistuisi. Hierarkia jatkuu vastaavasti tarinoihin, jotka edistävät ominaisuuksia, jotka ovat epiikoiden osasia. Korkeimman tason suunnitelu ja resurssijako tehdään teemoilla. [10]

Schwaber esittää toisenlaisen näkökannan hierarkiaan. Schwaberin mallissa pidetään useita työlistoja tasaisessa prioriteettijärjestyksessä. Näistä työlistoista luodaan hierarkia siten, että jokaisesta työlistasta vastaa yksi ryhmä. [20]

4.3.3 Jäljitettävyyssmatriisi

Jäljitettävyyssmatriisi on hyvin dokumenttipohjainen lähestymistapa jäljitettävyyteen. Tämä ei sovi kovin hyvin yhteen ketterien menetelmien kanssa, ellei jäljitettävyyssmatriisista ole saavutettavissa selkeää hyötyä tai arvoa asiakkaalle. Tämän vuoksi jäljitettävyyssmatriisia ei usein kannata soveltaa ketterissä projekteissa [2].

4.3.4 Testivetoinen jäljitettävyys

Testivetoisella kehityksellä tarkoitetaan ohjelmistokehityksen tapaa, jossa testeillä määritellään ohjelman toiminnallisuus. Testejä voidaan kirjoittaa etukäteen matlimalta, yksikkötestien, tasolta aina järjestelmän vaatimukseen ja hyväksymistestihin asti. Testivetoinen kehitys on XP:n käytäntö [3], mutta on levinnyt yleiseen käyttöön ketterissä menetelmissä.

Jäljitettävyyssmatriisia on käytetty yhdistämään vaatimuksia ja näiden testejä toisiinsa. Artikkelissaan Hayes (*et al.*) esittelee tavan ylläpitää jäljitettävyyttä automaattisesti testivetoisen kehityksen avulla. Kehittäjän työtaakkaa helpottamaan testivetoinen jäljitettävyys voisi mahdollistaa automaattisten regressiotestien luomisen. Eräs ajatuksista on luoda asiakkaan vaatimuksista automaattisia hyväksymistestejä, joilla voidaan varmistaa vaatimusten toteutuminen. [6]

Testivetoisella jäljitettävyydellä on mahdollista luoda osittainen jäljitettävyyssmatriisi kehityksen sivutuotteena [6]. Jäljitettävyyden automaattisessa generoinnissa on kuitenkin ongelmansa: testivetoiseen kehitykseen kuuluu oleellisena osana *refaktorointi*, joka saattaa sekoittaa automaattisen jäljitettävyyden pahoin. Refaktoroinnin aiheuttamat muutokset saattavat johtaa tilanteeseen, jossa jäljitettävyys on korjattava käsin. [6]

4.4 Ketteryys jäljitettävyyttä vaativassa yrityksessä

Tässä alaluvussa käsitellään jäljitettävyyttä eräässä belgialaisessa ilmailualan tuotetta toimittavassa organisaatiossa. Ilmailualalla standardit pakottavat jäljitettävyy-

den ylläpitoon [15], ja perinteisesti aloilla, joissa ihmishenkien menetys on mahdollista, on suositeltu käytettäväksi suunnitelmavetoisia prosessimalleja kuten vesiputousta [4]. Ilmailualan ohjelmistoteollisuudessa on noussut tarve asiakaslähtoisemmälle ja ketterämmälle ohjelmistokehitykselle: muutoksia vaatimuksiin tulee myöhään ja julkaisuväliä halutaan lyhyemmäksi [23].

“Ilmailualan ohjelmistokehitystä rajoittaa yksinkertainen, mutta joustamaton sääntö: estää ihmishenkien menetys” [23] (*suomentanut kirjoittaja*). Tästä johtuen alalla täytyy olla hyvin tarkka kaiken sellaisen kanssa, mikä saattaa vaarantaa matkustajien tai henkilökunnan hengen. DO-178B-standardi ei rajoita kehitystä seuraamaan tiettyä prosessia, vaan ainoastaan asettaa rajat kehitysprosessille. Käytännössä standardi pakottaa useamman dokumentin luomiseen jäljitettävyyden ja vaatimusten testauksen varmistamiseksi. Yrityksessä todettiin, että dokumentaatiosta mahdollisimman paljon kannattaa luoda automaattisesti, mikä sopii ketterien menetelmien periaatteisiin hyvin. [23]

Täysin ketteräksi ohjelmistokehitystä ei yrityksessä saatu. Prosessi ei mahdollista helppoja vaatimusmuutoksia tuotekehityksen loppuvaiheessa, mikä on taas yksi ketterien menetelmien pyrkimys. Loppuvaiheen muutokset vaatimuksiin aiheuttivat yrityksessä runsaasti lisätöitä verrattuna aiemmassa vaiheessa, joka tehtiin lähes täysin ketterästi, tehtyihin muutoksiin. Erityisesti sertifiointivaiheessa, jossa ohjelmistoa ei voida enää testata simuloidulla alustalla vaan aina lopullisella alustalla, voi automatisoitujenkin testien suorittamiseen kulua useampia päiviä. Jokaisesta tässä vaiheessa tehdystä muutoksesta täytyy lisäksi pitää yllä raskasta dokumentointia liittyen muun muassa jäljitettävyyteen. [23]

5 Yhteenveto

Jäljitettävyys on usein mielletty vain perinteisten menetelmien käytännöksi. On totta, että ketteryys ja jäljitettävyys eivät aina sovi hyvin yhteen. Pienissä projekteissa, joissa tuotteen vaatimukset ovat hyvin yksinkertaisia, ja niitä on suhteellisen vähän, ei jäljitettävyttä välttämättä kannata soveltaa. Kuitenkin jäljitettävyyden ylläpitämisestä voidaan hyötyä vaatimusten etenemisen seurannassa – hierarkiset vaatimukset muuttuvat läpinäkyviksi, kun kehittäjien tekemä konkreettinen työ näkyy edistymisenä jotain suurempaa päämäärää kohti.

Perinteisissä menetelmissä jäljitettävyttä käsitellään hyvin dokumenttilähtöisesti ja sitä pidetään yllä yleensä käsin. Ketterien menetelmien ajatusmalliin sopivat automatisoidut menetelmät luoda jäljitettävyttä, kuten testivetoinen jäljitettävyys. Perinteisten menetelmien jäljitettävyyskäytännöt voivat muokattuna sopia ketteriin menetelmiinkin. Työkalut voivat myös suoraan tukea jäljitettävyyden syntymistä luonnollisen suunnitteluprosessin sivutuotteena.

Ketteriä menetelmiä ei ole perinteisesti käytetty aloilla, joissa ihmishenkiä on vaarassa, kuten ilmailualalla. Näillä aloilla standardit tai vakiintuneet käytännöt pakottavat usein jäljitettävyyteen. On oleellista saada ketterä kehitys jäljitettäväksi, jotta sitä voidaan soveltaa myös tämänkaltaisilla aloilla.

Viitteet

- [1] Agile Alliance. Manifesto for agile software development. Viitattu 12.11.2009 <http://www.agilemanifesto.org>.
- [2] S. W. Ambler ja R. Jeffries. *Agile modeling: effective practices for extreme programming and the unified process*. Wiley New York, 2002.
- [3] K. Beck ja C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.
- [4] B. Boehm ja R. Turner. Using risk to balance agile and plan-driven methods. *Computer*, 36(6):57–66, 2003.
- [5] Orlena C. Z. Gotel ja Anthony C. W. Finkelstein. An analysis of the requirements traceability problem. Kirjassa *Requirements Engineering, 1994., Proceedings of the First International Conference on*, ss. 94–101, 1994.
- [6] J. H. Hayes, A. Dekhtyar, ja D. S. Janzen. Towards traceable test-driven development. Kirjassa *Proceedings of the 2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, ss. 26–30. IEEE Computer Society Washington, DC, USA, 2009.
- [7] IEEE. Ieee recommended practice for software requirements specifications. Tekninen raportti, IEEE, 1998. IEEE Std 830-1998.
- [8] David Kirkman. Requirement decomposition and traceability. *Requirements Engineering*, 3(2):107–114, 1998.
- [9] H. Kniberg. *Scrum and XP from the Trenches*.
- [10] D. Leffingwell. The big picture of enterprise agility.
- [11] F. Paetsch, A. Eberlein, ja F. Maurer. Requirements engineering and agile software development. Kirjassa *Proceedings of the Twelfth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, ss. 308–313. IEEE Computer Society Washington, DC, USA, 2003.
- [12] B. Ramesh ja M. Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.
- [13] B. Ramesh, T. Powers, C. Stubbs, ja M. Edwards. Implementing requirements traceability: a case study. Kirjassa *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, ss. 89–95. IEEE Computer Society Washington, DC, USA, 1995.
- [14] K. Rautiainen. Agile software development simple, but not easy. Luentokalvot 26.03.2009; Viitattu 15.11.2009.

- [15] R.D.O. RTCA. Do-178b. Tekninen raportti, RTCA, Incorporated, 1992.
- [16] K. Schwaber. The impact of agile processes on requirements engineering.
- [17] K. Schwaber. Scrum development process.
- [18] K. Schwaber. What is scrum? *Scrum Alliance*, 2007.
- [19] K. Schwaber ja M. Beedle. *Agile software development with Scrum*. Prentice Hall Upper Saddle River, NJ, 2001.
- [20] Ken Schwaber. *Agile Project Management With Scrum*. Microsoft Press, Redmond, WA, USA, 2004.
- [21] Ken Schwaber. *The enterprise and Scrum*. Microsoft Press, Redmond, WA, USA, 2007.
- [22] SEI. Cmmi for development, version 1.2. Tekninen raportti, Software Engineering Institute, 2006.
- [23] A. Wils, S. Van Baelen, T. Holvoet, ja K. De Vlaminc. Agility in the avionics software world. *Lecture Notes in Computer Science*, 4044:123, 2006.