Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto
Kandidaatintyö

Laadunhallinta ketterissä menetelmissä

Työn ohjaaja ja tarkastaja: Tekniikan tohtori Ossi Taipale

Lappeenranta, 26.08.2008

Vesa Kettunen
Kaivosuonkatu 3 G 20
53850 Lappeenranta
Puh: 040-7479597

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Tietotekniikan osasto

Vesa Kettunen

Laadunhallinta ketterissä menetelmissä

Kandidaatintyö

2008

37 sivua, 4 kuvaa, 1 taulukko

Tarkastaja: Tekniikan tohtori Ossi Taipale

Hakusanat: ohjelmistotestaus, ketterät menetelmät, laadunhallinta
Keywords: software testing, agile methods, quality assurance

Ketterät menetelmät ovat yleistymässä ohjelmistotuotannossa. Yleistyminen aiheuttaa paineita ohjelmistotestaukselle, koska ketteryyttä vaaditaan myös testaukselta. Ketterät menetelmät jakavat samat perusarvot ja periaatteet, mutta lähestymistavat niiden täyttämiseksi poikkeavat toisistaan. Tässä työssä tutkitaan tarkemmin Scrum-menetelmän vaikutuksia ohjelmistotestaukseen ja laadunhallintaan. Scrum-menetelmää verrataan myös eXtreme Programming-menetelmään.

ABSTRACT

Lappeenranta University of Technology
Department of Information Technology

Vesa Kettunen

Quality Management in Agile Methods

Bachelor's Thesis

2008

37 pages, 4 figures, 1 table

Supervisor: D.Sc. Ossi Taipale

Keywords: software testing, agile methods, quality assurance

Agile methods are becoming more popular in software engineering. This causes pressure for software testing because testing methods are required to be also more agile. Agile methods share the same basic values and principles, however, the approaches for fulfilling these differ from each other. My purpose is to study further the influence of the Scrum method on software testing and quality management. Scrum is also compared to the eXtreme Programming method.

SISÄLLYSLUETTELO

LYHENTEET

AM	Agile Modeling
ANTI	Application Strategies of New Technologies in Industrial Automation
ASD	Adaptive Software Development
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
MASTO	Adaptive Reference Model for Software Testing Organizations
OSS	Open Source Software Development
PP	Pragmatic Programming
QA	Quality Assurance
RUP	Rational Unified Process
TBRC	Technology Business Research Center
XP	eXtreme Programming

1 JOHDANTO

1.1 Taustaa

Tämä työ on osa MASTO (Adaptive Reference Model for Software Testing Organizations) – projektia. Projektista vastaava organisaatio on Lappeenrannan teknillisen yliopiston TBRC (Technology Business Research Center) – yksikkö. MASTO – projekti on jatkoa aikaisemmalle ANTI – projektille, jossa selvitettiin ohjelmistotestaukseen vaikuttavia tekijöitä ja niiden välisiä riippuvuuksia. ANTI – hankkeessa havaittuihin tekijöihin kuuluvat prosessit, tietämyksenhallinta sekä testausautomaatio. MASTO – projektin tarkoituksena on tutkia tarkemmin näitä tekijöitä, niiden välisiä riippuvuuksia sekä niiden vaikutuksia ohjelmistotestaukseen.

Ohjelmistokehityksessä ollaan monilla sovellusaloilla siirtymässä perinteisistä suunnitelmalähtöisistä menetelmistä ketterien menetelmien käyttöön. Tästä johtuen MASTO - tutkimuksen näkökulmina ovat ketterät menetelmät ja tietämyksenhallinta.

1.2 Työn tavoitteet ja rajaukset

Tämän kandidaatintyö kuuluu yllä esiteltyyn aihealueeseen ja työtä tullaan käyttää pohjatietona jatkotutkimuksia tehtäessä. Työlle asetettu tutkimuskysymys kuuluu: *Miten ketterät menetelmät vaikuttavat laadunhallintaan?*

Laadunhallinta on käsitteenä laaja, joten sitä tutkitaan tässä työssä ohjelmistotestauksen osalta. Ohjelmistotestauksessa keskitytään puolestaan prosesseihin, tietämyksenhallintaan sekä automaatioon. Ketteriksi menetelmiksi kirjallisuudessa luetellaan noin 10 parhaiten dokumentoitua menetelmää, joten kaikkia ei voida tämän työn puitteissa tutkia. Tässä työssä keskitytään pääasiassa Scrum-menetelmän tarkempaan selvittämiseen. Scrum-menetelmän eroja ja yhtäläisyyksiä peilataan XP-menetelmään (eXtreme Programming).

1.3 Työn rakenne

Työn alussa esitellään työn tutkimusalue (luku 2), joka koostuu ketteristä menetelmistä, laadunhallinnasta sekä ohjelmistotestauksesta. Luvussa esitellään ketterien menetelmien perusarvoja ja periaatteita sekä eroja suunnitelmalähtöisiin menetelmiin. Laadunhallinnan osiossa tarkastellaan laadun käsitettä, ja miten se sopeutuu tämän työn kontekstiin. Ohjelmistotestausta käsitellään yleisesti sekä tarkemmin testauksen prosessien, tietämyksenhallinnan ja automaation näkökulmista.

Kolmannessa luvussa esitellään lyhyesti tutkimusmenetelmä, jolla haetaan ratkaisua tutkimuskysymykseen. Neljännessä luvussa käydään Scrum-menetelmä yksityiskohtaisesti lävitse ja analysoidaan menetelmän vaikutuksia laadunhallintaan. Neljännen luvun lopuksi peilataan Scrum-menetelmän eroja ja yhtäläisyyksiä XP-menetelmään. Lopuksi luvussa viisi pohditaan havaittuja tuloksia tutkimuskysymykseen liittyen.

2 TUTKIMUSALUE

Tällä kandidaatin työllä valmistellaan diplomityötä Lappeenranta Innovation Oy:n Kasvuväylä-hankkeeseen. Sekä tämän kandidaatintyön että sitä seuraavan diplomityön tavoitteena on tukea alueen kasvuyrityksiä ketterien menetelmien käyttöönotossa erityisesti laadunhallinnan näkökulmasta. Tämä työ valmistelee ja tätä työtä seuraavassa diplomityössä osallistutaan ketterien menetelmien soveltamiseen ja käyttöönottoon ja selvitetään niiden vaikutusta lopputuotteen laatuun. Ohjelmistotestaus liittyy oleellisesti laadunhallintaan, joten tässä työssä selvitetään, kuinka ketterät menetelmät vaikuttavat laadunhallintaan, ja erityisesti kuinka ketterät menetelmät vaikuttavat testausprosesseihin, tietämyksenhallintaan ja testausautomaatioon. Työn tutkimusalue koostuu ketteristä menetelmistä, laadunhallinnasta sekä ohjelmistotestauksesta.

2.1 Ketterät menetelmät

Termi *ketterät menetelmät* syntyi helmikuussa 2001, kun 17 ihmistä tapasivat laskettelukeskuksessa Utahissa. Nämä ihmiset muodostivat Agile Alliance – liittouman ja määrittelivät Agile Manifestin (Fowler & Highsmith 2001). Manifestin tarkoitus oli esittää parempia menetelmiä ohjelmistokehitykseen. Manifestissa määriteltiin ketterille menetelmille yhteiset perusarvot:

- Yksilöt ja vuorovaikutus ovat tärkeämpiä kuin prosessit ja työkalut.
- Toimiva ohjelmisto on tärkeämpää kuin kattava dokumentointi.
- Yhteistyö asiakkaan kanssa on tärkeämpää kuin sopimusneuvottelut.
- Muutoksiin sopeutuminen on tärkeämpää kuin suunnitelman noudattaminen.

Jokaisen väittämän oikealla puolelle jäävä arvo on myös tärkeä, mutta ketterissä menetelmissä kyse on siitä, mikä on vielä tärkeämpää. Ensimmäinen väittämä nostaa esiin ohjelmistokehittäjät ja heidän väliset vuorovaikutukset. Kehittäjien välinen yhteistyö ja kommunikointi ovat tärkeitä tekijöitä, ja niihin kannustetaan esimerkiksi tiiviillä työympäristöllä. Ketterille menetelmille on ominaista uusien ohjelmistoversioiden tiheä julkaisutahti. Jokaisen version tarkka dokumentointi tuottaisi ohjelmistonkehittäjille kohtuuttoman määrän työtä. Toisen väittämän tarkoitus on helpottaa dokumentointia siten, että projektien työryhmät saavat itse päättää tarvittavan dokumentaation määrän. Näin saadaan resurssit pidettyä varsinaisessa kehitystyössä. Kolmas väittämä yhdistää

asiakkaan ja kehittäjät. Vain jatkuvalla ja luotettavalla yhteistyöllä asiakkaan kanssa, kehittäjät voivat vastata asiakkaan toiveisiin ja vaatimuksiin. Nerurin (2005) mukaan asiakkaan tulisi toimia työryhmän aktiivisena jäsenenä edustaen loppukäyttäjää. Neljäs väittämä perustuu Fowlerin (2001) mukaan siihen, että jos projekti on onnistunut, se on onnistunut vain koska kehittäjät ovat reagoineet ulkoisista syistä johtuneisiin muutoksiin. Sen lisäksi, että työryhmällä pitää olla valtaa muutoksien aiheuttamien korjausten tekemiseen, pitää työryhmän myös olla valmis tekemään tarvittavat korjaukset kehitysprosessin aikana (Abrahamsson et al. 2003, Fowler & Highsmith 2001).

Agile Manifestissa (Fowler & Highsmith 2001) on määritelty myös periaatteet, joita ketterien menetelmien tulee noudattaa. Nämä periaatteet ovat:

1. Korkein prioriteetti on taata asiakastyytyväisyys jatkuvilla ja aikaisilla ohjelmistotoimituksilla.
2. Muuttuvat vaatimukset hyväksytään ja ne otetaan vastaan myös kehityksen lopussa. Ketterä prosessi mukautuu muutoksiin, jotta asiakas voi saavuttaa kilpailuedun..
3. Toimivia ohjelmistoversioita toimitetaan säännöllisesti, kahden viikon tai kahden kuukauden välein.
4. Liiketoimintahenkilöstö työskentelee ohjelmistokehittäjien kanssa päivittäin projektin ajan.
5. Ketterissä menetelmissä luotetaan motivoituneisiin yksilöihin, rakennetaan projektit heidän ympärilleen ja annetaan heille heidän tarvitsema työympäristö ja tuki.
6. Tehokkain tiedonsiirto kehitystiimin kanssa sekä kehitystiimin sisällä tapahtuu kasvokkain keskustelemalla.
7. Toimiva ohjelmisto on tärkein edistymisen mitta.
8. Ketterät prosessit edistävät kestävää kehitystä. Rahoittajien, kehittäjien ja käyttäjien pitäisi pystyä ylläpitämään tasainen työtahti.
9. Jatkuva huomio tekniseen erinomaisuuteen ja hyvään suunnitteluun lisää ketteryyttä.
10. Yksinkertaisuus on elintärkeää.
11. Itseorganisoituvat tiimit saavat aikaan kehittyneimmät arkkitehtuurit, vaatimukset ja suunnitelmat.
12. Säännöllisin väliajoin tiimi arvioi, miten se voisi toimia tehokkaammin. Arvioinnin perusteella tiimi kehittää toimintaansa.

Abrahamssonin et al. (2003) mukaan ketterät menetelmät ovat yksinkertaisia ja nopeita. Ohjelmistokehityksessä tämä tarkoittaa sitä, että kehittäjät keskittyvät ohjelmiston kannalta tärkeimpiin toimintoihin, niiden nopeaan implementointiin, implementoinnista saatuun palautteeseen ja palautteesta saadun tiedon käyttämiseen seuraavassa ohjelmistoversiossa. Näiden ominaisuuksien avulla ketterät menetelmät pyrkivät vastaamaan liike-elämän haasteisiin, joissa vaaditaan nopeampia ja kevyempiä kehitysprosesseja jatkuvasti kasvavilla sovellusalueilla.

Abrahamsson et al. (2003) luonnehtii ketteriä menetelmiä seuraavasti:

- Inkrementaalinen, kasvava – Ohjelmistoa kasvatetaan pienin lisäyksin, nopeiden kehityskierrosten jälkeen.
- Yhteistyöhaluinen – Asiakas ja kehittäjät työskentelevät jatkuvasti keskenään, yhteydenpito on tiheää.
- Suoraviivainen, vaivaton – Menetelmä on helposti opittavissa ja muokattavissa, lisäksi se on dokumentoitu hyvin.
- Sopeutuvainen – Menetelmä mahdollistaa viime hetken muutoksien teon.

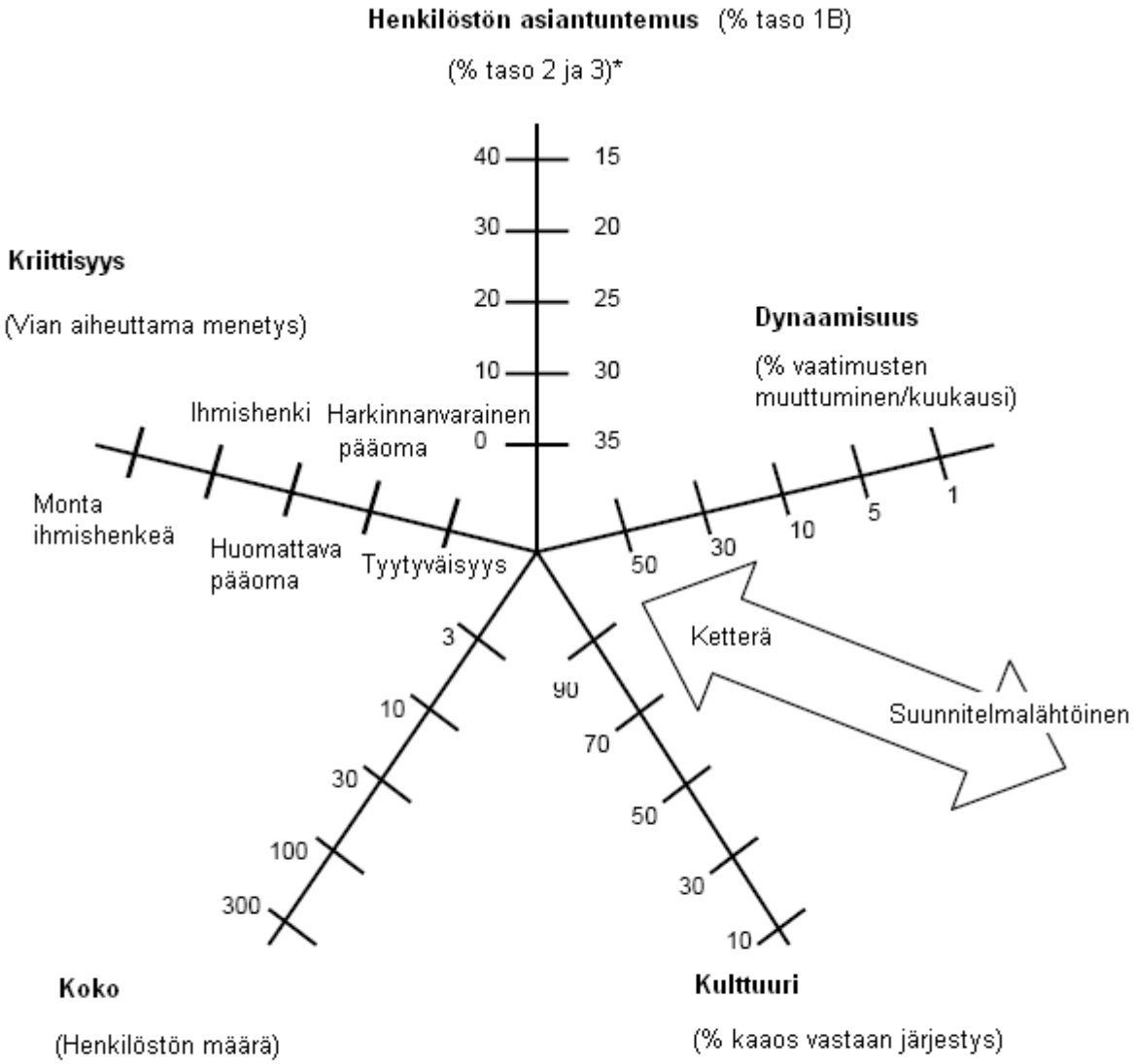
Abrahamssonin määritelmiä vastaavia ketteriä menetelmiä ovat Adaptive Software Development (ASD) (Highsmith 2000), Agile Modeling (AM) (Ambler 2002), The Crystal Family (Cockburn 2000), Dynamic Systems Development Method (DSDM) (Stapleton 1997), eXtreme Programming (XP) (Beck 2000), Feature Driven Development (FDD) (Palmer & Felsing 2002), Open Source Software Development (OSS) (O'Reilly 1999), Pragmatic Programming (PP) (Hunt & Thomas 2000), Rational Unified Process (RUP) (Kruchten 2004) sekä Scrum (Schwaber & Beedle 2002).

Ketterät menetelmät eroavat perinteisistä suunnitelmalähtöisistä (eng. plan driven) menetelmistä huomattavasti. Ketterien menetelmien yleistyminen on jakanut ohjelmistokehittäjien joukon ketterien menetelmien kannattajiin ja perinteisten menetelmien kannattajiin. Puolueettomat ovat esittäneet, että molemmilla menetelmillä on vahvuutensa ja heikkoutensa, mutta niiden sovellusalueet poikkeavat toisistaan (Nerur et al. 2005). Boehmin (2006) mukaan ketterät menetelmät soveltuvat pieniin ja riskittömämpiin projekteihin paremmin kuin perinteiset suunnitelmalähtöiset menetelmät. Projektit, joissa riskit ovat suuret ja ohjelmistolta vaaditaan vakautta, soveltuvat paremmin suunnitelmalähtöisille menetelmille. Menetelmien eroja on esitelty Nerurin mukaisesti taulukossa 1 ja Boehmin ja Turnerin (2003) mukaisesti kuvassa 1. Nykyisin

kuitenkin ketterien menetelmien käyttö laajenee sekä suurempien että kriittisempien järjestelmien toteutukseen.

Taulukko 1. Perinteisen ja ketterän menetelmän eroja (Nerur et al. 2005).

	Perinteinen	**Ketterä**
Perusolettamukset:	Järjestelmät ovat spesifioitavissa ja ne voidaan rakentaa erittäin tarkalla ja laajalla suunnittelulla.	Pienillä työryhmillä voidaan kehittää korkealaatuinen ja räätälöity ohjelmisto, hyödyntäen periaatteita: jatkuva suunnittelun ja testauksen parantaminen nopean palautteen ja muutosten ehdoilla.
Kontrolli	Prosessikeskeinen	Ihmiskeskeinen
Johtamistyyli	Komenna ja kontrolloi	Johtajuus ja yhteistyö
Tietämyksenhallinta	Eksplisiittinen, täsmällinen	Hiljainen tietämys
Roolien määrääminen	Yksilöllinen - suosii erikoistumista	Itseorganisoituvat ryhmät – kannustaa roolien vaihtoihin
Kommunikaatio	Muodollinen	Epämuodollinen
Asiakkaan rooli	Tärkeä	Kriittinen
Projektin sykli	Tehtävien tai toimintojen ohjaama	Tuotteen ominaisuuksien ohjaama
Kehitysmalli	Elinkaarimalli (vesiputous, spiraali tai variaatio)	Evoluutiomalli
Toivottu organisaatiorakenne	Mekaaninen (byrokraattinen ja muodollinen)	Orgaaninen (joustava ja osallistuva, kannustaen yhteistyöhön ja sosiaaliseen toimintaan)
Teknologia	Ei rajoituksia	Suosii olio-orientoitunutta teknologiaa



* Tasot 1B, 2 ja 3 kuvaavat Cockburnin ohjelmiston ymmärtämisen kolmea tasoa. Korkeammat tasot 2 ja 3 kuvaavat asiantuntijoita. Cockburnin asteikko on verrannollinen sovelluksen kompleksisuuteen. Kehittäjä voi olla tasolla 2 organisaatiossa, jossa kehitetään yksinkertaisia sovelluksia, mutta organisaatiossa, jossa kehitetään monimutkaisia sovelluksia sama kehittäjä voi olla tasolla 1A.

Kuva 1. Ketterien ja suunnitelmalähtöisten menetelmien eroja (Boehm & Turner 2003).

2.2 Laadunhallinta

Käsitteenä laatu on todella monimuotoinen. Laadulla tarkoitetaan usein jonkin tavaran tai palvelun hyvyyttä, mutta samalla se voi tarkoittaa myös asiakastyytyväisyyttä, virheettömyyttä, kestävyyttä, kannattavuutta, virheiden ennalta ehkäisyä tai muita vastaavia ominaisuuksia. Tervosen (2001) mukaan Suomessa käsite laatu on jatkuvasti laajentunut positiivisilla asioilla. Tämän vuoksi käsitteen selkeä määrittely on hankalaa. Käsitteen monimuotoisuus asettaa haasteita myös laadun mittaamiselle. Miten mitataan laatua, joka käsitteenä voi kattaa koko yrityksen toiminnan?

Tervonen (2001) yhtyy aiemmissa tutkimuksissa todettuun väittämään, jossa laatu pitäisi määritellä tapauskohtaisesti. Esimerkiksi Haikala ja Märijärvi (2004) kuvailee laatuajattelua ohjelmiston kannalta seuraavasti: ”Laadukkuutena pidetään mm. virheettömiä ohjelmia, ajan tasalla olevia käyttökelpoisia dokumentteja, mutta myös yksittäisen ohjelman suoritustehokkuutta ja luotettavuutta.” Asiakkaan näkökulman Tervonen (2001) määrittelee näin: ”Tuote tai palvelu katsotaan hyvälaatuiseksi, jos se täyttää asiakkaan tai käyttäjän tarpeet, odotukset ja vaatimukset. Virheettömyys- ja yhdenmukaisuusnäkökulmien voidaan katsoa sisältyvän myös asiakkaiden vaatimuksiin.” Tervosen mukaan mikäli yritys haluaa tuottaa laadukkaita tuotteita tai palveluita, pitää yrityksen koko henkilöstön sekä toimintojen toimia laadukkaasti. Tällöin asiakas saadaan tyytyväiseksi ja yrityksen toiminta pysyy kannattavana.

Ketterien menetelmien käyttö vaikuttaa valmistettavan tuotteen laatuun. Laadun mittareina toimivat asiakastyytyväisyys ja tuotteen virheettömyys. Asiakastyytyväisyys saavutetaan täyttämällä asiakkaan tarpeet ja tuotteen virheettömyys pyritään varmistamaan testaamalla tuotetta. Tässä työssä käsitelty tuotteen testaus on ohjelmistotestausta. Ohjelmistotestaukseen eniten vaikuttavia tekijöitä ovat prosessit, tietämyksenhallinta sekä testausautomaatio (Taipale 2007). Laadunhallinta ketterissä menetelmissä ulottuu näihin tekijöihin asti.

2.3 Ohjelmistotestaus

Kirjallisuudesta testaukselle löytyy useita määritelmiä. Tässä työssä käytetään määritelmää: Testaus muodostuu verifioinnista ja validoinnista (Kit 1995). Verifiointi ja validointi ovat prosesseja, joilla varmistetaan, että tuote täyttää spesifikaatiot ja asiakkaan odotukset (Sommerville 2001). Verifioinnilla tarkastetaan, tehdäänkö tuotetta oikein. Validoinnilla tarkistetaan, tehdäänkö oikeaa tuotetta. Verifiointia ja validointia suoritetaan koko ohjelmistokehityksen ajan. Kitin (1995) mukaan vaatimusmäärittelyvaiheessa syntyy yli 50 % ohjelmiston virheistä. Vaatimusten verifiointi mahdollistaa virheiden löytämisen aikaisessa vaiheessa, jolloin niiden korjaaminen on helpompaa ja halvempaa. Testaus tulee aloittaa ajoissa, jotta säästytään myöhemmin löytyvien virheiden aiheuttamilta kuluilta.

Ohjelmistotestauksen tavoitteena on tarkistaa, että ohjelmisto toimii mahdollisimman hyvin vaatimusten mukaisesti ja että ohjelmistoon jää mahdollisimman vähän virheitä ja heikkouksia

(Last et al. 2003, Kit 1995). Ohjelmistoa voidaan testata eri lähestymistavoilla; Watkinsin (2001) mukaan on olemassa positiivista ja negatiivista testausta. Positiivinen testaus pyrkii tarkastamaan, täyttääkö ohjelmisto sille asetetut vaatimukset. Negatiivisen testauksen lähtökohtana on, että ohjelmistossa on virheitä ja niitä on tarkoitus löytää. Yleensä testauksessa käytetään näitä elementtejä yhdessä. Kit (1995) lähestyy testausta negatiivisesta näkökulmasta kutsuen sitä luovaksi tuhoamiseksi.

Monissa lähteissä kerrotaan, että ohjelmistotestauksen osuus ohjelmistokehityksen kuluista voi olla jopa 50 % (Sommerville 2001, Bertolino 2007, Kit 1995). Vaikka tämä tosiasia monissa ohjelmistoyrityksissä tiedostetaan, tuotekehityksen venyessä otetaan tuotekehityksen tarvitsema lisäaika testauksesta (Taipale 2007, Kit 1995). Taipaleen mukaan tämä tarkoittaa sitä, että ohjelmistotestaukseen tarvittavia resursseja aliarvioidaan.

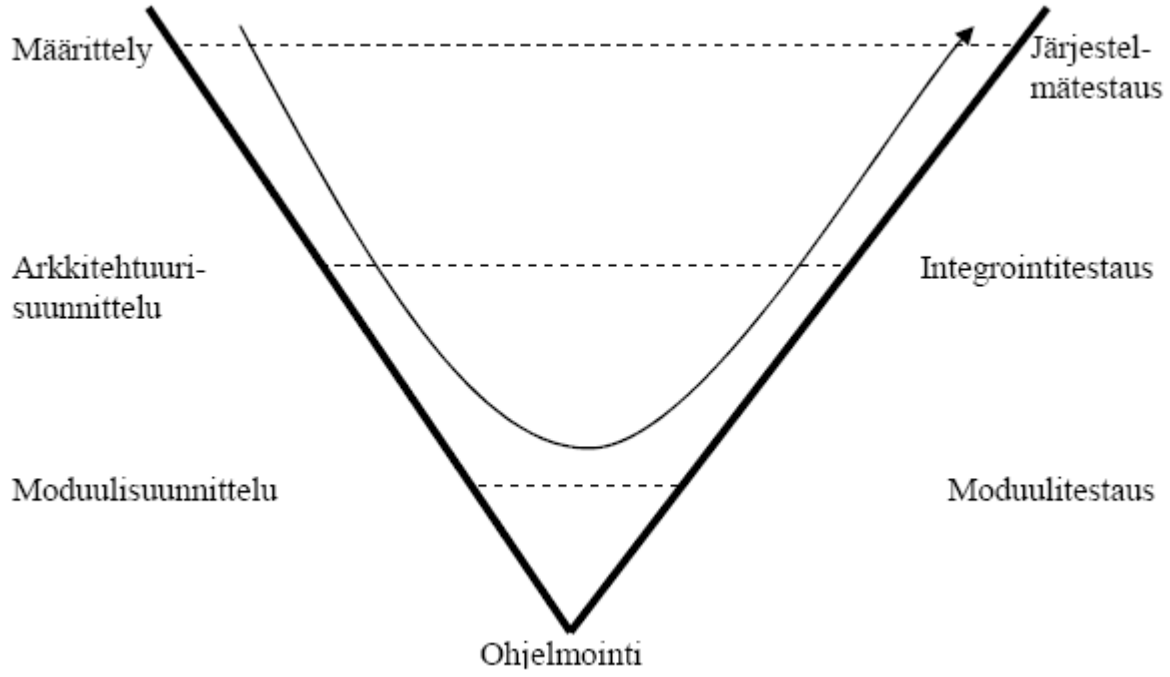
Testaustehtävät voidaan jakaa kahteen lähestymistapaan: Kirjoitettujen testien suorittamiseen (test scripts) ja tutkivaan testaukseen. Kirjoitetut testit koostuvat testitapauksien suorittamisesta ja niiden dokumentoinnista. Testitapaukset sisältävät testausohjeet, joten ne ovat automatisoitavissa. Tutkivassa testauksessa testaussuunnitelma muokkautuu suoritettujen testien myötä. Suoritettujen testien tuloksia hyödynnetään suunnittelemalla parempia testejä testauksen edetessä (Taipale 2007). Eri testauksen lähestymistavat korostavat erilaista tietämystä: käsikirjoitetut testit korostavat eksplisiittistä (dokumentoitua) tietämystä ja tutkiva testaus korostaa hiljaista tietämystä.

Kataran ja Kervisen (2006) mukaan ketterien menetelmien yleistyminen on kehitysaskel myös ohjelmistotestauksen näkökulmasta. Ohjelmistotestaus ei enää ole projektin viimeinen ja aliarvostetuin vaihe, vaan se kulkee osana koko kehitysprosessia. Testauksen arvostus lisääntyy, koska ketterissä menetelmissä arvostetaan testausautomaatiota ja hyväksymistestejä.

2.3.1 Prosessit

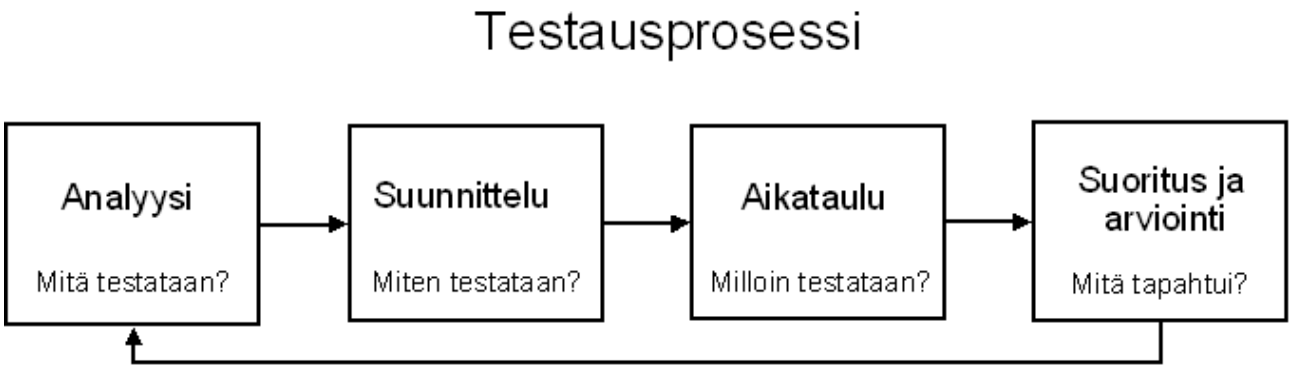
Kitin (1995) mukaan testauksen laatu on suoraan verrannollinen testauksessa käytettyjen prosessien laatuun. Laadukas tuote syntyy käyttämällä laadukasta kehitysprosessia ja laadukasta testausprosessia. Lisäksi tuote on laadukas vain, jos tuotteen loppukäyttäjä, asiakas, on siihen tyytyväinen.

Testausprosessien tarkoituksena on kytkeä testauksen suunnittelu jo ohjelmiston suunnitteluvaiheeseen. Bertolinon (2007) mukaan tähän mennessä esitellyistä testausprosesseista V-malli on ollut suosituin. V-malli käsittää moduuli-, integrointi- ja järjestelmätestauksen sekä antaa ohjeet, milloin kukin testausvaihe tulisi suunnitella. V-malli on esitelty kuvassa 2, vaakasuorat nuolet kertovat missä vaiheessa testaus suunnitellaan.



Kuva 2. Ohjelmistotestauksen V-malli (Haikala & Märijärvi 2004).

Davis (2000) on määritellyt yleisesti testausprosessin, kuvassa 3 on yleisen testausprosessin vaiheet.



Kuva 3. Testausprosessin vaiheet (Davis 2000).

Analyysivaihe kattaa lähdedokumentaation läpikäynnin, toimintojen arvojärjestyksen määrittelyn lähdedokumenttien avulla sekä testausehtojen määrittelyn toiminnoille. Suunnittelussa määritellään testausehtoja vastaavat testitapaukset. Testitapaukset kertovat testaajalle, mitä hänen tulee testata ja miten. Testitapaus sisältää myös tiedot testaukseen käytettävästä syötteestä sekä odotetusta tuloksesta. Aikataulu kertoo, missä järjestyksessä testitapaukset tulee suorittaa. Järjestys määräytyy testitapausten keskinäisten suhteiden ja prioriteettien perusteella. Suoritus ja arviointivaiheessa suoritetaan testitapaukset järjestyksessä. Jokainen tapaus kirjataan lokiin, josta ilmenee testitapauksen suoritusajankohta, onnistuiko testaus, pitääkö sama testi ajaa uudestaan sekä mahdolliset havaitut ongelmat testitapaukseen liittyen.

Ketterissä menetelmissä myös testausprosessilta vaaditaan ketteryyttä. Testauksesta ei vastaa erillinen testausryhmä; testaus on kehitystyöhön osallistuvien henkilöiden vastuulla. Ketterissä menetelmissä testaus kuuluu henkilöstön päivittäisiin työtehtäviin. Testaus tehdään kehitystyön rinnalla, ja testaustehtävät ovat samanarvoisessa asemassa kuin uusien toiminnallisuuksien kehittämistehtävät. Testauksesta saadaan palaute nopeasti ja löydetyt viat pyritään korjaamaan mahdollisimman pian. Ketterä testaus on haastavaa, koska kehittäjät on saatava testaamaan tuotetta (Talby et al. 2006).

2.3.2 Tietämyksenhallinta

Duffyn (1999) mukaan tietämyksenhallinta on organisaation kriittisen tietämyksen tunnistamista, kasvattamista ja tehokasta hyödyntämistä. Tietämyksen välittäminen on osa tietämyksenhallintaa (Taipale 2007). Nonakan (1994) mukaan tietämyksenhallintaan on kaksi strategiaa: kodifiointistrategia ja personalisointistrategia. Kodifiointistrategiassa tieto siirretään eksplisiittisessä muodossa eli esimerkiksi dokumentteina. Personalisointistrategiassa tietämys on hiljaista, ja se siirtyy henkilöiden välisissä vuorovaikutuksissa.

Agile manifestissa määriteltyjen periaatteiden mukaan, ketterät menetelmät kannustavat ihmisiä keskustelemaan toistensa kanssa. Tietämyksen välittämiseen käytetään personalisointistrategiaa, jolloin tietämys välittyy hiljaisessa muodossa. Fowlerin ja Highsmithin (2001) mukaan tietämyksen välittämisessä kyse ei ole dokumentaatiosta vaan ymmärtämisestä. He uskovat, että hiljaisen tietämyksen siirto paperille ei onnistu, joten tietämyksen siirto onnistuu vain viemällä ihmiset työskentelemään kasvokkain.

Karhu (2007) on todennut diplomityössään, että kodifiointistrategiasta on hyötyä tuotesuuntautuneessa ohjelmistokehityksessä, ja personalisointistrategiasta on hyötyä räätälöityjen ohjelmistojen kehityksessä. Kokonaan ilman kodifiointia ei kuitenkaan räätälöidyssäkään ohjelmistokehityksessä tulisi toimeen; sen vuoksi sopivan tasapainon löytäminen kodifioinnin ja pesonalisoinnin välillä olisi suotavaa.

2.3.3 Testausautomaatio

Testausautomaatio ei tarkoita testauksen täyttä automatisointia vaan testausautomaatio on tietokoneavusteista testausta (Kaner et al. 2004). Zhunin (2007) mukaan testausautomaatio on tulevaisuuden trendi ohjelmistotestauksessa. Vaikka testausautomaatiota on tutkittu vuosikymmeniä, on siitä nykyäänkin merkittävä osa manuaalisten testitapausten nauhoittamista, joita toistetaan myöhemmin esimerkiksi regressiotestauksessa. Regressiotestauksella varmistetaan mm., että ohjelmisto ei ole muuttunut esimerkiksi virheenkorjauksen jälkeen (Meszaros 2003).

Dustin et al. (1999) määritelmän mukaan testausautomaatio on sellaisen testaustoiminnan hallinnoimista, jossa vaatimusten verifiointiin kehitetään ja jossa suoritetaan testitapauksia käyttäen automatisoituja testaustyökaluja. Testauksen automatisoinnista saavutetaan huomattavimmat edut silloin, kun samoja testitapauksia joudutaan suorittamaan useamman kerran; esimerkiksi regressiotestauksessa samat testitapaukset toistuvat.

Inkrementaalisessa ohjelmistokehityksessä integrointitestauksen automatisointi on suotavaa. Uusi lisäys ohjelmistoon vaatii uusien testien luomisen, mutta integrointitestauksessa käytetään myös aiempien lisäysten jälkeen kehitettyjä testejä. Testausautomaation avulla voidaan lisätä ohjelmiston tarkkuutta ja luotettavuutta jokaisessa versiossa (Dustin et al. 1999).

Ketterät menetelmät hyödyntävät testausautomaatiota, koska menetelmille tyypilliset nopeat kehityskierrokset ja eri versioiden julkistaminen vaativat nopeaa ja täsmällistä testausta (Dustin 1999). Monet ketterät menetelmät suosivat testilähtöistä kehitystä, jossa ennen ohjelmointia koodille tehdään valmiiksi automatisoidut moduulitestit. Tällöin sovelluksella on testausautomaatio valmiina regressiotestausta varten, kun sovellusta kehitetään, laajennetaan ja ylläpidetään (Meszaros 2003).

3 TUTKIMUSMENETELMÄ

Ratkaisua tutkimuskysymykseen haettiin kirjallisuustutkimuksen menetelmillä. Kirjallisuustutkimus on vaihtoehto empiiriselle tutkimukselle, koska aina ei tarvita empiirisesti kerättävää aineistoa tutkimuksen tekemiseksi. Tällöin oletetaan, että tutkimuksessa tarvittava tieto on löydettävissä aiemmasta kirjallisuudesta. Usein kirjallisuustutkimus toteutetaan ennen empiiriseen tutkimukseen ryhtymistä (Routio 2007) kuten myös tässä.

Kirjallisuustutkimuksessa tutkija perehtyy ensin aiheeseen liittyvään kirjallisuuteen, jotta tutkijalle muodostuu käsitys aiheen tutkimustaustasta. Tutkimustaustan tunteminen antaa tutkijalle paremman käsityksen siitä, miksi hänen tutkimuksensa on ajankohtainen (Hirsjärvi et al. 1997).

Tämän työn tutkimustausta koostui ohjelmistotestauksesta, sen historiasta sekä siihen vaikuttavista tekijöistä. Ohjelmistotestauksen historian tuntemus auttoi ymmärtämään, miten ohjelmistotestaus on kehittynyt ja miten ketterät menetelmät vaikuttavat siihen.

Seuraava vaihe on konkreettisen aineiston kerääminen. Konkreettinen aineisto koostuu tutkimusongelmaan olennaisesti liittyvästä kirjallisuudesta (Hirsjärvi et al. 1997). Kirjallisuustutkimuksessa kirjallisuus ei ole vain tutkimuksen teoreettinen tausta, se on myös tutkimuksen aineisto sekä kohde (Jyväskylän yliopisto).

Tässä työssä konkreettisena aineistona toimivat Schwaberin ja Beckin teokset sekä niitä arvioivat tieteelliset julkaisut . Schwaber on yhdessä Sutherlandin kanssa kehittänyt Scrum-menetelmän. Beck on luonut XP-menetelmän. Schwaber ja Beck kuuluvat myös Agile Alliance – liittouman perustajiin. Schwaberin ja Beckin teoksia voidaan pitää puolueellisina, koska he uskovat luomiinsa menetelmiin ja eivät välttämättä halua nähdä menetelmien negatiivisia puolia. Molemmat kuitenkin myöntävät teoksissaan, että menetelmää ei välttämättä voi soveltaa kaikissa projekteissa. Puolueellista näkökulmaa on tässä työssä pyritty välttämään käyttämällä lähteinä Schwaberin ja Beckin teoksien lisäksi tieteellisiä julkaisuja, joissa menetelmiä on tutkittu empiirisesti.

Kirjallisuustutkimuksessa aineiston keräämisen jälkeen aineisto käsitellään ja tehdään muistiinpanoja. Muistiinpanot sisältävät tutkimusongelman kannalta oleellisia analyysejä, tuloksia

ja päätelmiä. Muistiinpanot auttavat kirjallisuustutkimuksen kirjoittamisessa. Aineiston käsittely kartuttaa myös tutkijan tutkimusmenetelmien ja – tekniikoiden tuntemusta (Hirsjärvi et al. 1997).

Kirjallisuustutkimuksen viimeinen vaihe on sen kirjoittaminen. Kirjallisuustutkimus kirjoitetaan muistiinpanojen pohjalta. Kirjallisuustutkimus yhdistää aiemmat oleellisesti aiheeseen liittyvät tutkimukset ja ne käsitellään tutkimuskysymykseen keskeisesti liittyvien asioiden näkökulmasta (Hirsjärvi et al. 1997). Tässä työssä tutkimuskysymykseen liittyvät keskeiset näkökulmat juontuvat ohjelmistotestauksen tekijöistä: prosessit, tietämyksenhallinta ja automaatio.

4 KETTERÄT MENETELMÄT

Työ on rajattu niin, että työssä käsitellään lähinnä Scrum menetelmää ja verrataan sitä XP menetelmään. Tässä luvussa esitellään Scrum menetelmä, verrataan sen yhtäläisyyksiä ja eroja XP menetelmään sekä esitetään havaintoja menetelmien vaikutuksista laadunhallintaan. Ketteristä menetelmistä tutkittavaksi on valittu Scrum, koska Scrum on hyvin dokumentoitu, se mukautuu erikokoisiin projekteihin ja sitä voidaan soveltaa myös muiden kuin tietojärjestelmien kehitysprosesseihin.

4.1 Scrum

Scrum on ketterä menetelmä, jossa ohjelmistoa tai tuotetta kehitetään iteraatio kerrallaan. Jokainen iteraatio kasvattaa ohjelmistoa yhdellä inkrementillä / laajennuksella. Schwaberin ja Beedlen (2002) mukaan Scrum on suoraviivainen, se poistaa tarpeettomat ja kömpelöt johtamismenetelmät, jolloin keskeisin osa on itse työ. Scrum antaa työryhmälle vapauden ja auktoriteetin määrätä itse työnteostaan. Itseorganisoituva työryhmä valmistaa ohjelmiston parhaan osaamisensa mukaan, parhaalla mahdollisella tavalla.

Scrum pohjautuu empiiriseen prosessin hallintaan. Empiirisen prosessin hallinnan lähtökohtia ovat avoimuus, tarkastelu ja sopeutuminen. Avoimuus tarkoittaa sitä, että käynnissä oleva prosessi on jokaisella hetkellä esillä eli mitään ei pidetä piilossa. Kun prosessi on avoin, jokainen näkee mihin suuntaan se on edistymässä, olkoon suunta sitten hyvä tai huono. Johdon on helpompi tehdä päätöksiä työn edistämiseksi, koska he näkevät missä vaiheessa prosessi on kullakin hetkellä. Tarkastelulla valvotaan, että prosessi toimii oikein. Jos tarkastuksessa havaitaan poikkeavuuksia prosessissa, sopeudutaan tilanteeseen korjaamalla prosessia (Schwaber 2007).

Sutherlandin (2001) kokemusten mukaan Scrumia voi käyttää niin pienissä kuin suurissakin projekteissa. Sutherland vakuuttaa, että Scrum toimii missä tahansa ympäristössä mukautuen erikokoisiin projekteihin. Samalla Scrum edistää ihmisten välistä kommunikointia sekä toimivan koodin tuottamista. Scrumin mukautumista erikokoisiin projekteihin tukee myös se, että Schwaber on kirjoittanut kirjan, jossa on ohjeistettu Scrumin toimintatapojen käyttöönottoa koko yrityksen toiminnassa. Tämän tutkimushankkeen seuraavissa vaiheissa on tarkoitus tutkia ketterien

menetelmien ja erityisesti Scrum-menetelmän mukautumista erikokoisiin projekteihin soveltamalla Scrum-menetelmää laajasti yrityksen eri prosesseihin.

4.1.1 Scrum roolit ja termit

Schwaber ja Beedle (2002) määrittelevät Scrumissa käytettävät roolit ja roolien vastuualueet. Scrum rooleja ovat Scrum-mestari (Scrum Master), tuotteen omistaja (Product Owner) ja Scrum-tiimi (Scrum Team). Roolien jälkeen on esitelty Scrum-prosessissa esiintyvät termit ja niihin liittyvät tapahtumat. Scrum-menetelmässä tuote kehitetään pyrähdyksissä (Sprint). Tässä työssä pyrähdyksestä käytetään termiä sprintti. Käytettävät termit ovat tuotteen työlista (Product Backlog), sprintti (Sprint), sprintin suunnittelupalaveri (Sprint Planning Meeting), sprintin tehtävälista (Sprint Backlog), Scrum-palaveri (Daily Scrum) ja sprintin arviointipalaveri (Sprint Review Meeting). Roolit ja termit on esitelty Schwaberin ja Beedlen (2002) kirjan mukaisesti.

Scrum-mestari (Scrum Master)

Scrum-mestari huolehtii siitä, että Scrumin sääntöjä noudatetaan projektissa. Hänen vastuullaan on opettaa ja informoida Scrum-prosessi muille asianomaisille (Scrum-tiimi, asiakas, tuotteen omistaja, johto, ym.). Scrum-mestari valitsee Scrum-tiimin jäsenet sekä tuotteen omistajan. Scrum-mestarin tärkein tehtävä on ylläpitää tiimin tuottavuutta kaikin keinoin. Hän ohjaa tiimin etenemistä ja poistaa häiriötekijöitä, joita tiimi tuo esille tai hän itse huomaa. Häiriöiden poisto on tärkeää, sillä tiimille on annettava työrauha jokaisen sprintin ajaksi. Tuottavuus laskee, jos tiimi altistuu häiriöille. Scrum-mestarilla tulee olla valtaa häiriötekijöiden poistamiseksi. Scrum-mestari ei saa puuttua liikaa Scrum-tiimin ongelmiin. Hänen on ohjattava tiimin toimintaa siten, että tiimi ottaa ohjat omiin käsiinsä. Näin tiimistä muodostuu itseorganisoituva, ja he oppivat tekemään ongelmien ratkaisuun vaadittavat päätökset itsenäisesti.

Tuotteen omistaja (Product Owner)

Tuotteen omistaja on projektista vastuussa oleva henkilö. Scrum-mestarin lisäksi, tuotteen omistajan valinnassa on mukana asiakas sekä yrityksen johto. Tuotteen omistajan vastuulla on tuotteen työlista (Product Backlog). Tuotteen omistaja on yksi henkilö, joka tekee päätökset työlistaan liittyen. Muiden on tuettava ja kunnioitettava hänen päätöksiään. Muut voivat vaikuttaa työlistaan vakuuttamalla tuotteen omistajaan. Tuotteen omistajan tehtävä on ylläpitää työlistaa ja pitää se julkisesti esillä.

Scrum-tiimi (Scrum Team)

Scrum-tiimi on itseorganisoituva. Scrum-mestari takaa tiimille työrauhan sprintin aikana, ja tiimi saa itse tehdä päätökset sprintin aikana esiintyvien ongelmien ratkaisemiseksi. Tiimi päättää itse miten se saavuttaa sprintille asetetut tavoitteet. Tiimi itse valitsee tuotteen työlistasta sprintin aikana toteutettavat toiminnallisuudet. Näistä toiminnallisuuksista muodostuu sprintin tehtävälista (Sprint Backlog).

Scrum-tiimin koko on 5-9 henkilöä; Schwaber (2002) on kuitenkin todennut, että yli kahdeksan henkilöä ei tuo tiimille lisäarvoa. Tiimin kokoonpanon tulisi olla sekoitus ammattilaisia kuten suunnittelijoita, laadun hallinnan henkilöitä sekä toteuttajia. Jokainen tiimin jäsen tuo oman ammattitaitonsa tiimin käyttöön, jolloin ongelmien ratkaisuun saadaan näkökulmia eri ammattilaisilta. Schwaberin (2002) mukaan tällaisen kokonaisuuden synnyttämä synergia parantaa koodin laatua sekä lisää tuottavuutta. Synergiaetuna on myös se, että sprinttien aikana tiimin yhteistyö ja keskustelut lisäävät yksilöiden tietämystä, kun he ovat vuorovaikutuksessa eri asiantuntijoiden kanssa. Jatkossa tämä edistää Scrum-prosessin toimintaa (tuottavuutta), koska tiimin jäsenet kehittyvät jatkuvasti ja heistä muodostuu monipuolisia osaajia.

Tuotteen työlista (Product Backlog)

Työlista koostuu asioista, jotka halutaan tuotteeseen tai jotka vaikuttavat tuotteeseen. Listassa luetellaan tuotteen ominaisuudet, toiminnot, teknologiat, parannukset sekä tuotteesta löydetyt virheet, joiden korjaus vaatii muutoksia tuotteeseen. Toisin sanoen työlistassa on listattuna kaikki työt, jotka liittyvät tuotteen valmistukseen. Listan kukin kohta on priorisoitu ja sille on arvioitu tarvittava työmäärä. Työmäärät arvioi Scrum-tiimi. Priorisoitu lista auttaa tunnistamaan tärkeimmät toiminnallisuudet, koska ne implementoidaan ensimmäisinä. Listaan tehdään muutoksia projektin edetessä, koska siten voidaan reagoida muuttuviin vaatimuksiin, teknologioihin tai muihin ennalta arvaamattomiin muutoksiin. Tuotteen omistaja vastaa listan päivittämisestä.

Sprintin tehtävälista (Sprint Backlog)

Tehtävälista koostuu sprintin aikana toteutettavista tuotteen työlistasta valituista toiminnallisuuksista. Jokainen tehtävä on kuvailtu niin yksityiskohtaisesti, että sille varattu toteutusaika on 4–16 tunnin välillä. Sprintin tehtävälista ei muutu, vaan se on pysyvä. Kun kaikki tehtävät on suoritettu, laajennus on valmis. Tehtävälista on samalla tavoin julkinen kuin tuotteen työlista, näin johdon ja muiden kiinnostuneiden on mahdollista seurata tiimin edistymistä. Tiimin on päivitettävä tehtävälistaa säännöllisesti.

Scrum-palaveri (Daily Scrum)

Sprintin aikana järjestetään päivittäin Scrum-palaveri, johon osallistuu Scrum-tiimi ja Scrum-mestari; myös muut saavat osallistua palaveriin mutta vain kuuntelijoina. Tapaaminen kestää noin 15 minuuttia. Näiden tapaamisten avulla Scrum-mestari tietää, miten Scrum-tiimi edistyy. Tapaamisen tarkoitus on selvittää osallistujien kesken, mitä tiimi on saavuttanut edellisen tapaamisen jälkeen, mitä on tarkoitus tehdä ennen seuraavaa tapaamista, ja mitä esteitä tai häiriöitä on ilmaantunut. Scrum-mestari on vastuussa palaverin järjestämisestä.

Sprintin suunnittelupalaveri (Sprint Planning Meeting)

Suunnittelupalaveri koostuu kahdesta osasta. Ensin asiakkaat, käyttäjät, johto, tuotteen omistaja, Scrum-tiimi ja Scrum-mestari kokoontuvat suunnittelemaan, mitä sprintin aikana toteutetaan. Suunnittelussa käytetään apuna tuotteen omistajan esittämää priorisoitua tuotteen työlistaa. Toisessa vaiheessa tiimi kokoontuu suunnittelemaan miten tavoitteeseen päästään. Samalla he muodostavat sprintille tehtävälistan. Tiimin kokoontuessa mukana voi olla Scrum-mestari ja tuotteen omistaja.

Sprintti

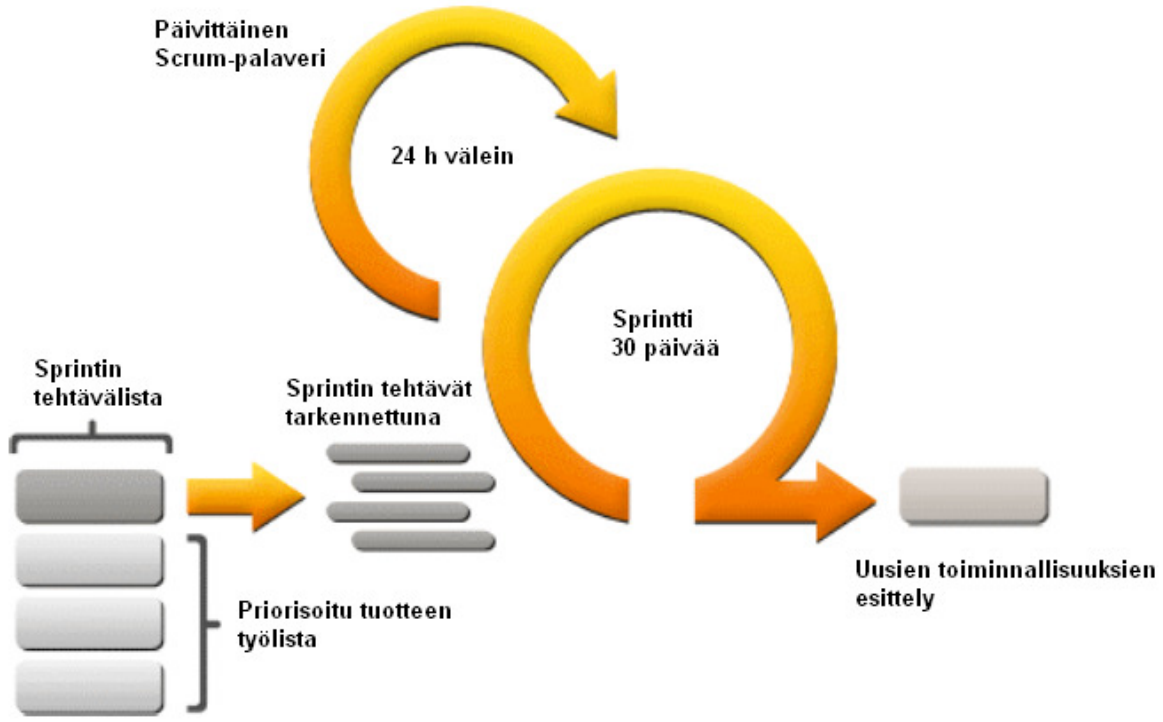
Sprintti on 30 päivän mittainen pyrähdys, jonka aikana tiimin odotetaan tekevän toimiva laajennus tuotteeseen. Sprintin aikana kukaan ei saa häiritä tiimiä, koska tiimi on itseorganisoituva ja pyrkii saavuttamaan sprintin tavoitteen itsenäisesti. Kun häiriöitä ei esiinny, työntekijä voi keskittyä laajennuksen tekemiseen, jolloin laajennuksesta tulee niin hyvä kuin työntekijän taidot mahdollistavat. Tiimi voi muuttaa sprintissä tehtävän työn määrää, mutta sen on kuitenkin pysyttävä tavoitteessa. Tiimiltä vaaditaan tuotelaajennus sprintin päätteeksi. Sprintin aikana tiimi analysoi, suunnittelee, valmistaa ja testaa laajennuksen. Testaus suoritetaan jokaisessa sprintissä, ja se on osa koko tiimin työtä.

Scrum arviointipalaveri (Scrum Review Meeting)

Arviointipalaverissa kaikki pääsevät näkemään tiimin tuloksen eli tuotelaajennuksen. Usein asiakkaat ja muut asianomaiset saavat uusia ajatuksia ja käyttökohteita tuotteelle laajennuksen nähtyään. Lisäksi osallistujat saavat käsityksen tiimin kokemuksista sprintin aikana, sekä saavat tietoa tuotelaajennuksen mahdollisista heikkouksista ja vahvuuksista. Kaikki edellä mainitut tiedot ovat tarpeen seuraavassa sprintin suunnittelupalaverissa.

4.1.2 Scrum-prosessi tiivistettynä

Scrum-prosessin iteratiivinen luonne on havainnollistettu kuvassa 4. Prosessi on niin yksinkertainen, että se voidaan kuvata yhdellä kaaviolla.



Kuva 4. Scrum-prosessi (Koskela 2007).

Tuotteen valmistus alkaa työlistan määrittämisellä. Työlista priorisoidaan, koska ensimmäiseksi tehdään tuotteen tärkeimmät osat. Varsinainen työ tehdään sprinteissä, jotka kestävät 30 päivää. Sprintin alussa tiimi valitsee yhdessä tuotteen omistajan kanssa sprintille tehtävälistan sprintin suunnittelupalaverissa. Sprintti on se työväline, jolla reagoidaan projektissa eteen tuleviin muutoksiin. Projektin etenemiseen voidaan vaikuttaa 30 päivän välein, jolloin valmistettava tuote pysyy kilpailukykyisenä (Schwaber 2007).

Sprintin aikana tehtävälista on lukittu eli siihen ei voi lisätä toiminnallisuuksia 30 päivään. Tiimi toimii sprintin ajan itsenäisesti tehtävänään saavuttaa sprintille asetettu tavoite parhaimmaksi katsomallaan tavalla. Scrum-tiimi kokoontuu päivittäin Scrum-palaveriin, jonka avulla he synkronoivat toimintansa ja tietävät miten projekti edistyy. Kunkin sprintin päätteeksi tiimi on valmistanut uuden laajennuksen tuotteeseen, joka esitellään sprintin arviointipalaverissa (Schwaber 2007).

Sprintin tavoitteesta huolimatta, suositellaan Scrum-tiimin jäsenten työajan pitämistä noin 40 tunnissa viikkoa kohden. Schwaber ja Beedle (2002) ovat havainneet, että ylityön aiheuttama kuormitus vähentää valmistettavan tuotteen laatua pitkällä aikavälillä. Lyhyellä aikavälillä ylityöllä voidaan saavuttaa enemmän tuloksia, mutta pitkään ylikuormitettu työntekijä väsyy, ja väsynyt työntekijä tekee enemmän virheitä (Robinson 2005).

4.1.3 Laadunhallinta

Scrum-menetelmän yhtenä lähtökohtana on korkea laatu. Miten korkea laatu saavutetaan, on jätetty itseorganisoituvan Scrum-tiimin päätettäväksi. Scrum tarjoaa tiimille täyden työrauhan jokaisen 30 päivän sprintin ajaksi, ja odottaa tiimin valmistavan laadukkaan tuotelaajennuksen sprintin päätteeksi. Schwaber ja Beedle (2002) mainitsevat, että laatuvaatimukset ovat yleensä organisaation standardin mukaiset. Scrum-tiimi voi kuitenkin määrittää laatuvaatimukset sprinteille erikseen, jos standardeja ei käytetä.

Scrum ei ohjeista jonkin tietyn tekniikan käyttöön tuotteen kehittämiseksi tai testaamiseksi. Usein tiimi aloittaa Scrum-menetelmän soveltamalla yrityksessä aiemmin käytettyjä tekniikoita. Prosessin edetessä Scrum puuttuu käytettyihin tekniikoihin, jos niistä koituu häiriöitä tiimille tai jos ne vähentävät tiimin tuottavuutta (Schwaber & Beedle 2002).

Scrumin sääntöjen mukaan jokainen tuotelaajennus tulee testata sprintin aikana, joten testaus on olennainen osa prosessia. Testaus on koko tiimin vastuulla. Kirjallisuudessa ei esiinny tiettyä testausprosessia, jota Scrumissa tulisi käyttää. Scrum-menetelmän käyttöönotossa on esiintynyt ongelmia juuri testauksen osalta.

Puleion (2006) kohtaamat ongelmat johtuivat testauksen kommunikoinnista, työmäärän arvioinnista ja automatisoinnista. Puleionin tiimi kuitenkin ratkaisi ongelmat projektin edetessä. Kommunikointiongelmat johtuivat siitä, että tiimin muut jäsenet eivät tienneet testauksesta mitään tai tiesivät siitä hyvin vähän. Testaaja oli uudessa tilanteessa, jossa hänen piti opettaa testausta muille tiimin jäsenille. Testauksen työmäärän arviointi oli testaajalle vaikeaa, koska hänellä ei ollut kokemusta toimenpiteiden pilkkomisesta Scrumin käytännön mukaisesti 4–16 tuntia vaativiin osiin. Testausautomaatioon kohdistuneet ongelmat johtuivat tiimin kehittäjäjäsenten ja tuotteen omistajan asenteista. Oli tärkeämpää saada uusia toiminnallisuuksia, kuin testata vanhat huolellisesti.

Sumrellin (2007) projektissa testaus ei pysynyt kehitystyön vauhdissa. Sprinttien aikana ei ehditty testaamaan tuotetta, joten regressiotestaukselle varattiin viikko aikaa sprintin jälkeen. Ongelmiin vaikutti vesiputousmallin mukainen ajattelu, jossa vaaditaan valmis koodi testauksen suorittamiseksi ja jossa kehittäjillä ja testaajilla on selvä roolijako. Lisäksi testauksella ei ollut suunnitelmaa sprintin alkaessa.

Testausprosessissa tapahtuu muutoksia väistämättä, jotta laatuvaatimukset täytetään. Testaustehtävät on osa sprintin tehtävälistaa, joten tehtävät suunnitellaan sprintin suunnittelupalaverin aikana ja testaus alkaa sprintin käynnistyessä. Sumrellin (2007) kokemuksen perusteella hyväksymistestien määrittely ennen sprintin alkua on tärkeää, jotta testauksella on jotkin kriteerit joiden mukaan toimia. Roosmalen (2007) on ehdottanut Scrum-projekteille testaussuunnitelmaa, joka perustuisi ylemmän tason yrityskohtaisesti määriteltyyn testauspolitiikkaan ja testausstrategiaan (ISO/IEC 29119).

Testaajan rooli muuttuu Scrumin myötä aktiivisemmaksi. Testaaja on osa Scrum-tiimiä, jolloin hän on aina ajan tasalla, ja kommunikointi tiimin jäsenten kesken on helppoa. Testaamisen lisäksi testaajan tehtävänä on esittää kysymyksiä ohjelmistosta ja varmistaa, että tiimin jäsenet pyrkivät tuottaa laadukkaan ohjelmiston (Roosmalen 2007). Testaajan on myös välitettävä tietämystään, jotta muut tiimin jäsenet ymmärtävät testauksen vaativuuden ja tarpeellisuuden. Jos tietämys ei siirry, muiden on vaikea ymmärtää testauksen vaatimaa työmäärää. Testaajan on myös ohjattava muut jäsenet testaamaan ohjelmistoa, koska testaaminen ja laatu ovat koko tiimin vastuulla. Schwaberin ja Beedlen (2002) mukaan kaikki osallistuvat kaikkeen työhön kykyjensä sallimissa rajoissa, joten roolijakoa ei tiimin sisällä ole. Tällöin kaikki tietävät valmistettavasta ohjelmistosta lähes yhtä paljon, joten heitä voidaan käyttää eri tehtävissä tehokkaasti, myös testaustehtävissä. Esimerkiksi Sumrell (2007) havaitsi, että kehittäjien täytyy tehdä yksikkötestaus, jotta ketterän menetelmän käyttö onnistuu.

Testausautomaation käyttäminen Scrum-menetelmässä on ensisijaisen tärkeää (Puleio 2006, Roosmalen 2007, Sumrell 2007). Sumrellin (2007) mukaan kevyttä automaattista testaustyökalua tulisi käyttää heti projektin alusta lähtien. Puleion (2006) toteaa testauksen automatisoinnin olevan ketterän testauksen perusta. Aikaisessa vaiheessa kehitetty helposti laajennettava testausautomaatiorakenne on hyödyllinen koko projektin ajan. Kun testaus on automatisoitu, testauksesta saatava palaute tulee nopeasti ja toiminnallisuuksien kehittämiseen jää enemmän aikaa (Puleio 2006).

Roosmalen (2007) perustelee testausautomaation käyttöä Scrum-menetelmässä sillä, että manuaaliseen regressiotestaukseen ei ole aikaa, joten se on automatisoitava. Testausautomaatio mahdollistaa myös testien suorittamisen öisin, jolloin niistä saatava palaute on valmiina seuraavan työpäivän alkaessa. Lisäksi testausautomaatio on tarpeellinen inkrementaalisen kehityksen vuoksi, koska aiemmin suunniteltuja testejä tarvitaan seuraavan laajennuksen testaamisessa.

Scrum-menetelmän on todettu vaikuttavan asiakastyytyväisyyteen. Schwaberin (2002, 2007) esimerkeissä asiakas on lähes poikkeuksetta ollut innoissaan, kun Scrum on otettu käyttöön. Tätä havaintoa tukee Mannin ja Maurerin (2005) tutkimus. Mannin ja Maurerin (2005) tutkimuksen mukaan asiakkaat ovat olleet tyytyväisiä Scrumiin, koska he ovat päässeet osallistumaan paremmin tuotteen kehitysprosessiin. Erityisen hyödyllisinä asiakkaat pitivät sprintin suunnittelupalavereja ja päivittäisiä Scrum-palavereja. Suunnittelupalavereja pidettiin hyödyllisinä, koska ne ehkäisivät väärinymmärryksiä tuotteen vaatimuksissa ja rajoituksissa. Scrum-palaverien avulla asiakkaat saivat päivittäin ajankohtaista tietoa prosessin etenemisestä. Asiakkaat kokivat, että heidän oli helpompi tehdä heiltä vaadittuja päätöksiä, koska he saivat ajankohtaista tietoa tuotteesta, ongelmista tai muista tuotteeseen vaikuttavista tekijöistä. Lisäksi asiakkaat pitivät siitä, että sprintin jälkeen he näkivät tuotteen ja pystyivät sen nähtyään muokkaamaan tai lisäämään tuotteen toiminnallisuuksia.

4.2 Scrum- ja eXtreme Programming-menetelmien vertailua

Scrum- ja XP-menetelmien (eXtreme Programming) yhtäläisyydet perustuvat Agile Manifestissa (Fowler & Highsmith 2001) ketterille menetelmille määriteltyihin perusarvoihin ja periaatteisiin. Molemmissa menetelmissä pyritään nopeisiin ohjelmistoversioiden julkaisuihin valmistamalla ohjelmisto iteratiivisesti, tuotelaajennus kerrallaan ja toimittamalla tärkeimmät toiminnallisuudet ensimmäisinä. Menetelmien huomattavin ero on, että XP-menetelmä keskittyy ohjelmiston kehitystapoihin, kun Scrum-menetelmä keskittyy kehitystyön hallitsemiseen (Vriens 2003).

Scrum- ja XP-prosesseissa on havaittavissa yhtäläisyyksiä ja eroja. XP-prosessi alkaa siten, että asiakas kirjoittaa haluamansa toiminnallisuudet. Seuraavassa vaiheessa kehittäjät arvioivat asiakkaan kirjoittamien toiminnallisuuksien implementointiin tarvittavan työmäärän ja arviointien avulla valitsevat asiakkaan kanssa tärkeimmät toiminnallisuudet, jotka toteutetaan ensimmäisessä julkaisussa. Toiminnallisuuksien valitsemisen jälkeen edetään yhdestä neljään viikkoa kestävään

iterointivaiheeseen, jossa toiminnallisuudet kehitetään toimivaksi kokonaisuudeksi. Ensimmäisessä iteraatiossa tulisi valita toiminnallisuudet, jotka muodostavat valmistettavalle ohjelmistolle karkean arkkitehtuurin. Jatkossa asiakas valitsee iteraatioiden aikana toteutettavat toiminnallisuudet.

XP-prosessissa ensimmäistä versiota ei välttämättä julkaista ensimmäisen iteraation päätteeksi, vaan menetelmässä sallitaan useampien iteraatioiden suorittaminen ennen ensimmäistä julkaisua. Julkaisun lähestyessä iteraatioiden pituus voi lyhentyä ja uusia testejä tulee kirjoittaa, jotta voidaan varmistua version olevan julkaisukuntoinen. Julkaisun jälkeen ylläpidetään ohjelmistoa ja tuotetaan siihen uusia toiminnallisuuksia samanaikaisesti. Kehitystyö päättyy, kun asiakas ei enää tuo esille uusia toiminnallisuuksia toteutettaviksi.

Scrum antaa tiimille valtuuden päättää siitä, millaista kehitystapaa se haluaa käyttää. Scrum puuttuu kehitystapaan, jos se aiheuttaa ongelmia tai esteitä kehitystyön etenemiselle. XP-menetelmässä annetaan tarkat ohjeet kehitystyölle. Kehitystyössä käytetään jatkuvaa koodin integroimista, koodin yhteisomistusta ja pariohjelmointia. Näiden tekijöiden yhteiskäytön uskotaan tuottavan korkealaatuista koodia (Beck 2000). Layman et al. (2004) tutkimustulokset tukevat tätä uskomusta. He vertasivat virheiden määrää ennen ja jälkeen XP-menetelmän käyttöönottoa. He havaitsivat, että ohjelmistojen esijulkaisuista (pre-release) löytyi 65 prosenttia vähemmän vikoja XP-menetelmän käyttöönoton jälkeen. Asiakkaille asti päässeet viat vähenivät myös 35 prosentilla.

Seuraavaksi on käsitelty tämän työn kannalta merkittävistä näkökulmista menetelmien yhtäläisyyksiä ja eroja. Näkökulmat ovat ohjelmistotestauksen prosessit, tietämyksenhallinta sekä automaatio.

4.2.1 Testausprosessi

Scrum-menetelmässä ei ohjeistettu tietyn testausprosessin käyttöön. Schwaber (2002) oli kuitenkin sitä mieltä, että tiimissä pitäisi olla testaajajäsen, joka ohjaa muut tiimin jäsenet testaamaan tuotetta. XP-menetelmässä testaus on asiakkaan ja pareittain ohjelmoivien kehittäjien vastuulla. Kehittäjät kirjoittavat kehittämilleen ominaisuuksille yksikkötestit, ja asiakkaat kirjoittavat sovellukselle käyttötapauksia sekä toiminnallisuustestejä. XP-menetelmässä testaajan rooli on auttaa asiakasta kirjoittamaan heiltä vaaditut testit (Beck 2000).

XP käyttää ohjelmiston kehityksessä testilähtöistä kehitystapaa, jossa testit tehdään ennen varsinaista ohjelmointia. Siten sovellus voidaan ohjelmoida läpäisemään testit. Beckin (2000) mukaan tämä lisää kehittäjien itseluottamusta ja varmuutta, koska kehittäjät saavat välittömästi palautteen sovelluksen läpäisemistä testeistä. Kehittäjien ei tällöin tarvitse arvuutella, toimiiko ohjelmoitu sovelluksen osa.

4.2.2 Tietämyksenhallinta

Sekä Scum- että XP-menetelmässä korostetaan kommunikoinnin ja henkilökohtaisten vuorovaikutusten merkitystä. Molemmissa menetelmissä arvostetaan tietämyksen välittämistä hiljaisessa muodossa, koska se on nopein ja tehokkain tiedonsiirtotapa (Schwaber 2002, Beck 2000).

Scrum-menetelmässä testaaja on tärkeässä asemassa tietämyksen välittäjänä, koska hänen pitää välittää testaustietämyksensä muille tiimin jäsenille. XP-menetelmässä testaaja ei ole samanarvoisessa asemassa. Testaus on kehittäjien vastuulla, joten kehittäjien välinen tietämyksen siirto korostuu. Beckin (2000) mukaan koodin yhteisomistus sekä pariohjelmointi edistävät tiedonsiirtymistä kehittäjien välillä. Tietämys välittyy myös kierrättämällä kehittäjiä siten, että parit vaihtuvat. Tämä vähentää projektin henkilöstöön liittyviä riskejä, koska jokainen sovelluksen osa on tuttu vähintään kahdelle kehittäjälle (Beck 2000).

4.2.3 Testausautomaatio

Tutkimuksissa on todettu, että testauksen automatisointi on lähes välttämätöntä käytettäessä Scrum-menetelmää (Puleio 2006, Roosmalen 2007, Sumrell 2007). Myös XP-menetelmässä testausautomaatio on otettava huomioon. Beckin (2000) kanta testausautomaation käyttöön XP-menetelmässä on selkeä: sovelluksen ominaisuutta, jota ei voi todentaa automatisoiduilla testeillä, ei ole olemassa.

XP-menetelmässä käytettävä koodin jatkuva integroiminen vaatii testausautomaation käyttöä, koska siten integroiminen voidaan tehdä nopeasti. Beck (2000) painottaa, että tarvitaan testausympäristö, jolla testit voidaan suorittaa muutamissa minuuteissa; muussa tapauksessa XP-menetelmän

mukainen kehitystapa ei ole mahdollinen. Beckin (2000) mukaan testit myös pidentävät sovelluksen elinikää, ja niiden avulla on helpompi tehdä muutoksia sovellukseen. Tämä tietenkin edellyttää, että myös testejä ylläpidetään ja suoritetaan muutoksia tehtäessä.

5 JOHTOPÄÄTÖKSET

Tässä luvussa esitetään havainnot tutkimuskysymykseen valituista näkökulmista. Tutkimusongelmana olivat ketterien menetelmien vaikutukset laadunhallintaan. Laadunhallinnan alueesta tutkimus kohdistui ohjelmistotestaukseen, ja tarkemmin ohjelmistotestauksen tekijöihin; prosessit, tietämyksenhallinta ja testausautomaatio. Tuote on laadukas vain, jos asiakas on tyytyväinen tuotteeseen. Tämän vuoksi asiakastyytyväisyys on huomioitu myös havaintoja tehtäessä.

Tämä työ rajattiin kahden ketterän menetelmän selvittämiseen. Tämä kandidaatintyö toimii lähtökohtana jatkossa tehtävälle diplomityölle, jossa sovelletaan tässä työssä lähemmin tarkasteltua Scrum-menetelmää. Niin tässä työ kuin tulevassa diplomityössä laadunhallinta on tärkeänä näkökulmana.

Seuraavassa on esitelty kirjallisuudesta tehdyt havainnot ketterien menetelmien vaikutuksista ohjelmistotestaukseen ja siten myös laadunhallintaan:

Ketterät menetelmät vaikuttavat testausprosessiin muuttamalla testauksen ajankohtaa, toimeenpanijoita ja arvostusta.

Ketterissä menetelmissä testausta ei tehdä vesiputousmallin mukaisesti projektin viimeisenä vaiheena, vaan testaus kulkee koko ajan kehitystyön rinnalla. Kehitettävällä tuotteella pitäisi olla testaussuunnitelma, koska sen avulla testausta voidaan kehittää tuotteen kehityksen rinnalla. Testausta on kehitettävä, koska muuten testaus jää jälkeen projektin edetessä.

Erillisen testausryhmän käyttö on ilmeisesti liian raskasta, joten vastuu tuotteen testauksesta on siirretty kehitystiimille. Testaaja voi olla tiimin jäsen ja häneltä vaaditaan sopeutumiskykyä uuden roolin omaksumiseksi.

Testauksen arvostus kasvaa, koska ketterille menetelmille tyypillinen testauksen lähtökohta on: tuotelaajennus, jota ei ole testattu, ei ole valmis. Tuotekehityksen venyessä, lisäaikaa ei voi ottaa testauksesta, koska tuotelaajennus on testattava ennen julkaisua. Tämä asetelma voi johtaa siihen, että testaukseen tarvittava aika otetaan tuotekehitykseen varatusta ajasta.

Ketterissä menetelmissä tietämyksenhallintaan on pääasiallisesti käytettävä personalisointistrategiaa.

Ketterät menetelmät kannustavat tiimin jäseniä keskustelemaan toistensa kanssa. Hiljaista tietämyksensiirtoa kannustetaan, koska nopein ja tehokkain tiedonsiirto tapahtuu henkilöiden välisissä vuorovaikutuksissa.

Tiedon laaja kodifiointi ei ole perusteltua, koska se on ketterien menetelmien muodostamassa työympäristössä hidasta, ja se vie aikaa varsinaiselta kehitystyöltä. Yrityksen testauspolitiikka ja -strategia voivat kuitenkin olla dokumentoituna, koska ne helpottavat projektien testaussuunnitelman tekoa. Dokumentoinnin tarve riippuu kuitenkin sovellusympäristöstä; esimerkiksi laillisuusvaatimukset voivat edellyttää laajempaa dokumentointia.

Menetelmien käyttämät nopeat kehityskierrokset vaativat, että testaustietämys siirtyy nopeasti tiimin jäsenten kesken. Testaajan roolimuutos edellyttää, että hän välittää testaustietämystään tiimin jäsenille ja asiakkaille tarpeen mukaan.

Ketterissä menetelmissä testausautomaation käyttö on välttämätöntä.

Nopeissa kehityskierroksissa manuaaliseen testaukseen ei ole aikaa, joten testaus on automatisoitava automatisoitavissa olevilta osiltaan. Ilman testausautomaatiota, testausta ei voida suorittaa riittävällä tarkkuudella. Automatisoinnin etuna on, että automatisoidut testit antavat palautteen nopeasti, jolloin tuotteen kehitystyölle jää enemmän aikaa. Lisäksi automatisoitu testausympäristö lisää kehittäjien itseluottamusta ja varmuutta, koska he saavat palautteen ohjelmoidun osan toimivuudesta nopeasti.

Ketterän menetelmän käyttö voi lisätä asiakastyytyväisyyttä.

Viitteitä asiakastyytyväisyyden kasvamisesta ketterän menetelmän käyttöönoton jälkeen on olemassa. Mannin ja Maurerin (2005) tutkimuksessa havaittiin, että asiakas oli tyytyväisempi kehitysprosessiin Scrum-menetelmän käyttöönoton jälkeen. Layman et al. (2004) tutkimuksessa ei virallisesti mitattu asiakastyytyväisyyttä. He kuitenkin havaitsivat, että asiakas oli erittäin tyytyväinen XP-menetelmällä valmistettuun tuotteeseen.

LÄHDELUETTELO

Abrahamsson, P., Warsta, J., Siponen, M. and Ronkainen, J, New Directions on Agile Methods: A Comparative Analysis, paper presented to 25th International Conference on Software Engineering, 2003.

Ambler, S. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process,* John Wiley & Sons, New York, 2002.

Beck, K., *Extreme Programming Explained: Embrace Change,* Addison-Wesley, 2000.

Bertolino, A., Software Testing Research: Achievements, Challenges, Dreams, *Future of Software Engineering FOSE '07,* pp. 85-103, 2007.

Boehm, B., A View of 20th and 21st Century Software Engineering, 2006.

Boehm, B. and Turner, R., Using Risk to Balance Agile and Plan-Driven Methods, *Computer,* vol. June, pp. 57-66, 2003.

Cockburn, A., *Writing Effective Use Cases, The Crystal Collection for Software Professionals,* Addison Wesley, 2000.

Davis, G., Managing the Test Process, *Proceedings of the International Conference on Software Methdods and Tools 2000,* Computer Society Press, Wollongong, NSW, Australia, pp. 119-126, 2000.

Duffy, N., Benchmarking Knowledge Strategy, 1999.

Dustin, E., Rashka, J. and Paul, J., *Automated Software Testing: Introduction, Management, and Performance,* Addison-Wesley, Boston, 1999.

Fowler, M. and Highsmith, J. The Agile Manifesto, 2001.

Haikala, I., ja Märijärvi, J., *Ohjelmistotuotanto,* Talentum, Helsinki, 2004.

Highsmith, J.A., *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems,* Dorset House Publishing, New York, 2000.

Hirsjärvi, S., Remes, P., ja Sajavaara, P. *Tutki ja kirjoita.* Dark Oy, Vantaa, 1997.

Hunt, A. and Thomas, D., *The Pragmatic Programmer,* Addison Wesley, 2000.

ISO, International Software Testing Standard, ISO/IEC 29119.

Jyväskylän yliopisto, Tutkimuksen prosessi, Saatavissa: http://www.avoin.jyu.fi/oppiaineet/kirjoittajakoulutus/palvelusivut/lahiopetus/tieteellinen_kirjoittaminen/johdantoluento/prosessi [Viitattu 14.8.2008]

Kaner, C., Bond, W. and McGee, P., High Volume Test Automation, International Conference on Software Testing Analysis & Review, Orlando, Florida, 2004.

Karhu, K., Tietämyksen välittäminen ohjelmistotestausorganisaatioissa, 2007.

Katara, M., and Kervinen, A., Making Model-Based Testing More Agile: a Use Case Driven Approach, To appear in Proc. Haifa Verfication Conference, LNCS, Springer, 2006.

Kit, E., *Software Testing in the Real World: Improving the Process,* Addison-Wesley, Reading, 1995.

Koskela, L., Scrum: Ketterien menetelmien markkinajohtaja, 2007. Saatavissa: http://ttlry-fi-bin.directo.fi/@Bin/c56cfdb4abc1b86289c4941114714319/1217922841/application/pdf/11062393/04_ScrumMarketLeaderOfAgileMethods_handout_LasseKoskela.pdf [Viitattu 5.8.2008]

Kruchten, P., *The Rational Unified Process An Introduction Third Edition,* Addison Wesley, Upper Saddle River, NJ, 2004.

Last, M., Friedman, M., and Kandel, A., The Data Mining Approach to Automated Software Testing, 2003.

Layman, L., Williams, L. and Cunningham, L., Exploring Extreme Programming in Context: An Industrial Case Study, *Proceedings of the Agile Development Conference (ADC'04),* 2004.

Mann, C. and Maurer, F., A Case Study on the Impact of Scrum, *Proceedings of the Agile Development Conference (ADC'05),* 2005.

Meszaros, G., Agile Regression Testing Using Record & Playback, *OOPSLA 2003*, Anaheim, California, 2003.

Nerur, S., Mahapatra, R., and Mangalaraj, G. Challenges of Migrating to Agile Methodologies, *Communications of the ACM*, vol. 48 no. 5, pp. 72-78, 2005.

Nonaka, I., A Dynamic Theory of Organizational Knowledge Creation, *Organization Science,* 5, pp. 14-37, 1994.

O'Reilly, T., Lessons from Open Source Software Development, *Communications of the ACM*, Vol. 42 (No. 4), pp 32-37, 1999.

Palmer, S.R. and Felsing, J. M., *A Practical Guide to Feature-Driven Development,* 2002.

Puleio, M., How Not to do Agile Testing, *Proceedings of AGILE 2006 Conference (AGILE'06)*, 2006.

Robinson, E., Why Crunch Mode Doesn't Work: 6 Lessons, International Game Development Association, 2005, Saatavissa: http://www.igda.org/articles/erobinson_crunch.php [Viitattu 12.8.2008]

Roosmalen, R., Testing and Scrum, Fall 2007 Scrum Gathering, 2007. Saatavissa: http://www.scrumalliance.org/resources/270 [Viitattu 5.8.2008]

Routio, P., Kirjallisuusselvitys, 2007, Saatavissa: http://www.uiah.fi/virtu/materiaalit/tuotetiede/html_files/120_kirjallisuus.html [Viitattu 14.8.2008]

Schwaber, K. and Beedle, M. *Agile Software Development with Scrum,* Prentice Hall, Upper Saddle River, NJ, 2002.

Schwaber, K., *Agile Project Management with Scrum,* Redmond, Washington, 2004.

Schwaber, K., *The Enterprise and Scrum,* Redmond, Washington, 2007.

Sommerville, I., *Software Engineering 6th Edition,* Pearson Education, Harlow, England, 2001.

Stapleton, J., *Dynamic systems development method – The method in practice,* Addison Wesley, 1997.

Sumrell, M., From Waterfall to Agile – How does a QA Team Transition?, *Agile 2007*, pp. 291 – 295, 2007.

Sutherland, J., Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies, Cutter IT Journal, Vol. 14, No. 12, 2001.

Taipale, O., *Observations on Software Testing Practice,* Lappeenranta, Acta Universitatis Lappeenrantaensis, 2007.

Talby, D., Keren, A., Hazzan, O. and Dubinsky, Y., Agile Software Testing in a Large-Scale Project, *IEEE Software,* pp. 30-37, 2006.

Tervonen, A., *Laadun kehittäminen suomalaisissa yrityksissä,* Lappeenranta, Acta Universitatis Lappeenrantaensis, 2001.

Vriens, C., Certifying for CMM Level 2 and ISO9001 with XP@Scrum, *Proceedings of the Agile Development Conference (ADC'03),* 2003.

Watkins, J., *Testing IT: An Off-the-Shelf Software Testing Process,* Cambridge University Press, United Kingdom, 2001.

Zhu, H., Wong, W. and Paradkar, A., Automation of Software Test – Report on the Second International Workshop AST 2007, *29th International Conference on Software Engineering ICSE'07,* 2007.