Henri Kulju

LAADUNVARMISTUS KETTERISSÄ OHJELMISTOKEHITYSMENETELMISSÄ

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2014

TIIVISTELMÄ

Kulju, Henri
Laadunvarmistus ketterissä ohjelmistokehitysmenetelmissä
Jyväskylä: Jyväskylän yliopisto, 2014, 26 s.
Tietojärjestelmätiede, Kandidaatin tutkielma
Ohjaaja: Makkonen, Pekka

Tässä tutkielmassa tarkastellaan laadunvarmistusta ketterissä ohjelmistokehitysmenetelmissä. Ohjelmiston laadunvarmistus on tärkeässä roolissa koko ohjelmiston elinkaaren aikana ja sillä voi olla suuri vaikutus koko ohjelmistoprojektin onnistumiseen. Jos siihen ei kiinnitetä tarpeeksi huomiota, voi tuloksena olla aikataulun viivästyksiä, budjetin ylityksiä sekä huonolaatuista ohjelmistoa. Tutkielmassa esitellään ketteriä menetelmiä yleisellä tasolla, sekä tarkastellaan kahta menetelmää tarkemmin. Nämä menetelmät ovat Extreme Programming(XP) sekä Scrum. Ketterien menetelmien laadunvarmistusta tarkastellaan kahdeksan XP:stä ja Scrumista löytyvän käytänteen avulla. Tutkielmassa käydään läpi näiden käytänteiden hyötyjä, sekä vaikutusta ohjelmiston laatuun. Tutkielma tarjoaa loppupäätelmän siitä miten ketterien menetelmien käytänteet auttavat varmistumaan laadusta sekä siitä, onko ketterissä menetelmissä esiintyvät laadunvarmistuskäytänteet riittäviä. Tutkielman tutkimuskysymykset ovat seuraavat: Kuinka ketterissä menetelmissä voidaan varmistua ohjelmiston laadusta? Ovatko ketterien menetelmien laadunvarmistuskäytänteet riittäviä varmistumaan ohjelmiston laadusta? Tutkimusmenetelmänä käytetään kirjallisuuskatsausta ja tutkielman lähdeaineisto koostuu pääasiassa alaa käsittelevistä tieteellisistä artikkeleista.

Asiasanat: laatu, laadunvarmistus, ketterät menetelmät, extreme programming, Scrum

ABSTRACT

Kulju, Henri
Quality Assurance in Agile Methods
Jyväskylä: University of Jyväskylä, 2014, 26 p.
Information Systems Science, Bachelors thesis
Supervisor: Makkonen, Pekka

In this bachelors thesis will be described quality assurance in agile methods. Software quality assurance is in crucial role during the whole lifecycle of the software and it may have a big impact on the success of software project. If quality assurance is not on high enough level it can lead to delayed schedule, budget overruns and low quality software. This thesis will introduce agile methods in general, and it will also introduce two agile methods in specific. Those methods are Extreme Programming(XP) and Scrum. The quality assurance of agile methods will be introduced by describing eight practices that can be found in XP and Scrum. The benefits and the effects of these eight practices will be introduced in this thesis. This thesis will offer a conclusion about how the practices of agile methods will help to guarantee sufficient quality, and also a conclusion about the sufficiency of agile methods when thinking about quality assurance. The research questions in this thesis are: How can agile methods ensure quality? Are the quality assurance practices sufficient in agile methods? Research method used in this thesis is literature review and the source material of this thesis is mainly from scientific articles from this field of study.

Keywords: quality, quality assurance, agile methods, extreme programming, Scrum

SISÄLLYS

1 JOHDANTO

1990-luvun loppupuolella on alettu käyttämään yhä enemmän ketteriä menetelmiä perinteisten suunnittelupohjaisten menetelmien sijaan (Huo, Verner & Ali Babar, 2004). Ketterät menetelmät syntyivätkin vastareaktiona perinteisille suunnittelupohjaisille ja dokumentaatiokeskeisille menetelmille (Cohen, Lindvall & Costa, 2004). Toisin kuin suunnittelupohjaiset menetelmä, ketterät menetelmät ovat luotu vastaamaan epävakaisiin ja nopeasti muuttuviin asiakkaan vaatimuksiin (Huo ym., 2004). Ketterissä menetelmissä ei oleteta, että asiakas pystyy antamaan täydelliset ja oikeat vaatimukset ohjelmistolle projektin alussa. Sen sijaan projektin alussa kerätään pieni määrä vaatimuksia, jonka perusteella luodaan osa toimivaa ohjelmaa, jonka avulla asiakas pystyy tarkentamaan ja määrittämään lisää vaatimuksia ohjelmistolle. Käytetyimmät ketterät menetelmä ovat Extreme Programming(XP) ja Scrum.

Niin perinteisessä, kuin ketterässä ohjelmistokehityksessä on tärkeää, että valmistettu ohjelma on laadukasta. Tällöin voidaan varmistua asiakkaan tyytyväisyydestä. Tuotteen laadulle on monia määritelmiä, mutta kirjallisuudessa usein esiintyvän Juranin ja Grynan (1988) määritelmän mukaan laatu koostuu siitä, että tuote täyttää ominaisuuksiltaan ja piirteiltään asiakkaan tarpeet. Määritelmän mukaan tuotteella tulisi olla lisäksi mahdollisimman vähän puutteita ja heikkouksia. Myös ohjelmistojen laadulle löytyy monta määritelmää. Radatzin, Geracin ja Katkin (1990) mukaan ohjelmiston laatu on aste, jolla ohjelmisto, komponentti tai prosessi täyttää sille asetut vaatimukset sekä asiakkaan tarpeet ja odotukset. ISO 9126 laatustandardi(2001) taas määrittää kuusi kriteeriä ohjelmiston laadulle. Nämä kuusi kriteeriä ovat toiminnallisuus, luotettavuus, tehokkuus, käytettävyys, ylläpidettävyys ja siirrettävyys.

Laadunvarmistuksella tarkoitetaan varmistumista siitä, että ohjelmisto valmistuu ajallaan sekä budjetin määrittämissä rajoissa. Ohjelmiston on myös täytettävä sille asetetut toiminnalliset vaatimukset ja sen tulisi sisältää mahdollisimman vähän virheitä.(Hossain, Kashem & Sultana, 2013) Kirjallisuudessa painotetaan laadunvarmistuksen tärkeyttä ohjelmistokehityksessä. Esimerkiksi Owenin ja Khazanchin (2009) mukaan ohjelmiston laadunvarmistus on kriittisessä roolissa koko ohjelmiston elinkaaren aikana ja sillä voi olla suuri vaikutus koko ohjelmistoprojektin onnistumiseen. Jos ohjelmiston laadunvarmistukseen

ei kiinnitetä tarpeeksi huomiota, se voi johtaa budjetin ylitykseen, aikataulun viivästyksiin, projektin tavoitteiden saavuttamattomuuteen ja asiakastyytyväisyyden kärsimiseen (Chow, 1985). Jotta tuotetut ohjelmistot olisivat laadukkaita on valmistajan huolehdittava riittävästä ja oikeanlaisesta laadunvarmistuksesta. Owensin ja Khazancin (2009) mukaan laadunvarmistus on hyvin määritelty, toistettavissa oleva prosessi, joka on integroitu niin projektin johtamisen, kuin ohjelmiston elinkaaren aikana tapahtuviin toimiin. Näillä toimilla varmistutaan siitä, että ohjelmistolle asetettuja vaatimuksia noudatetaan, vähennetään ohjelmistoon kohdistuvia riskejä, parannetaan ohjelmiston laatua sekä pysytään aikataulussa ja budjetissa.

Tämän tutkielman tarkoituksena on muodostaa yleiskuva ketterän ohjelmistokehityksen laadunvarmistuksesta yleisellä tasolla, sekä tutustua tarkemmin kahdeksaan käytänteeseen, joilla ketterät menetelmät pyrkivät vastaamaan laadunvarmistuksen haasteisiin. Liitteeseen 1 on kerätty näiden kahdeksan käytänteet hyödyt, sekä Liitteessä 2 on tarkasteltu miten kukin käytänne auttaa ohjelmistokehittäjiä parantamaan tuotteen ominaisuuksia, jotta ne vastaisivat ISO 9126(2001) mukaisia laatukriteereitä. Ketteristä menetelmistä tutkielmaan on valittu tarkasteltaviksi XP ja Scrum.

Tutkimusmenetelmänä käytetään kirjallisuuskatsausta. Tutkielman lähdeaineisto koostuu pääosin alaa käsittelevistä tieteellisistä artikkeleista. Tutkielma on rajattu niin, että ketteristä menetelmissä tarkasteltaviksi on valittu kaksi suosituinta, eli XP ja Scrum. Laatua tarkastellaan tutkielmassa vain tuotteen laadun näkökulmasta. Prosessin laatuun liittyvät asiat on tutkielmasta jätetty pois. Tutkielman tutkimuskysymykset ovat:

- Kuinka ketterissä menetelmissä voidaan varmistua ohjelmiston laadusta?

- Ovatko ketterien menetelmien laadunvarmistuskäytänteet riittäviä ohjelmistonlaadun takaamiseksi?

Luvussa 2 määritellään mitä ketterät menetelmät ovat ja kuinka niiden avulla voidaan valmistaa ohjelmistoja. Luvussa tutustutaan tarkemmin kahteen ketterään menetelmään, jotka ovat XP ja Scrum. Luvussa 3 esitellään kirjallisuudesta löytyviä määritelmiä laadulle, ohjelmiston laadulle sekä laadunvarmistukselle. Luvussa 4 tarkastellaan laadunvarmistusta ketterissä menetelmissä. Luvussa esitellään kahdeksan ketteristä menetelmistä tuttua käytännettä, joilla ketterissä menetelmissä pyritään varmistumaan ohjelmiston laadusta. Luvussa 5 tarkastellaan tehtyä tutkimusta, sekä esitellään tutkimuksen tulokset.

2 KETTERÄT MENETELMÄT

Ennen kuin voidaan käsitellä tekniikoita ja menetelmiä, joilla ketterissä ohjelmistokehitysmenetelmissä voidaan varmistua ohjelmiston laadusta, on syytä tuntea ketterien menetelmien perusperiaatteita. Tässä luvussa käydään läpi ketterien menetelmien yleisiä periaatteita, sekä tutustutaan syvällisemmin kahteen suosituimpaan menetelmään, jotka ovat Extreme Programming(XP) ja Scrum. Tutkielmassa käydään myöhemmin läpi laadunvarmistusta ketterissä ohjelmistokehitysmenetelmissä. Näiden kahden menetelmän lisäksi on kehitetty monia muita ketteriä menetelmiä, mutta tässä tutkielmassa tarkastellaan vain näitä kahta niiden suosion takia.

Yritykset ovat alkaneet 90-luvun lopun jälkeen käyttämään toiminnassaan yhä enemmän ketteriä ohjelmistokehitysmenetelmiä perinteisten suunnittelupohjaisten menetelmien sijaan (Huo ym., 2004). Ketterät menetelmät syntyivätkin vastareaktiona perinteisille suunnittelupohjaisille ja dokumentaatiokeskeisille menetelmille (Cohen ym., 2004). Vuonna 2001 kutsuttiin koolle 17 ihmistä, jotka olivat kehittäneet omia kevyempiä ohjelmistokehitysmenetelmiä ja tämän kokoontumisen tuloksena oli Agile Alliance yhteisö, joka ajaa ketterien menetelmien kehitystä (Larman & Basili, 2003). Agile Alliance julkaisi vuonna 2001 Agile Manifeston, jonka pääperiaatteita kaikki ketterät ohjelmistokehitysmenetelmät noudattavat. (Abbas, 2009).

Agile Manifestossa (Fowler & Highsmith, 2001) määritetään neljä arvoa, jota ketterät menetelmät noudattavat ja joiden ympärille eri menetelmät rakentuvat:

- yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja,
- toimivaa ohjelmistoa enemmän kuin kattavaa dokumentointia,
- asiakasyhteistyötä enemmän kuin sopimusneuvotteluita ja
- vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa

Ketterät menetelmät ovat luotu vastaamaan epävakaisiin ja nopeasti muuttuviin asiakkaan vaatimuksiin (Huo ym., 2004). Ketterissä menetelmissä ei oleteta, että asiakas pystyy antamaan täydelliset ja oikeat vaatimukset ohjelmistolle

projektin alussa. Sen sijaan projektin alussa kerätään pieni määrä vaatimuksia, jonka perusteella luodaan osa toimivaa ohjelmaa, jonka avulla asiakas pystyy tarkentamaan ja määrittämään lisää vaatimuksia ohjelmistolle. Ketterissä menetelmissä korostetaankin asiakkaan jatkuvaa osallistumista ohjelmiston kehitykseen. (Abbas, 2009)

Toinen piirre, joka löytyy kaikista ketteristä menetelmistä on iteratiivinen kehitys. Iteratiivisella kehityksellä tarkoitetaan sitä, että ohjelmiston elinkaari koostuu useista iteraatioista. Jokainen iteraatio on pieni projekti, jossa käydään läpi kaikki ohjelmistokehityksen perusaskeleet, jotka ovat vaatimusmäärittely, suunnittelu, toteutus ja testaus. Jokaisessa iteraatiossa kehitetään pala toimivaa ohjelmistoa, jota parannetaan ja laajennetaan seuraavassa iteraatiossa uusien ja parannettujen vaatimusten perusteella. (Monash & Dwolatzky, 2007)

Kolmas yhteinen piirre ketterille menetelmille on inkrementaalisuus. Inkrementaalinen kehitys tarkoittaa sitä, että jokainen ohjelmiston moduuli kehitetään niin, että uusia ohjelmiston moduuleita on mahdollista luoda uusien vaatimusten perusteella. Näistä ohjelmiston moduuleista rakennetaan jokaisessa julkaistavassa ohjelmiston versiossa toimiva ja testattu ohjelma, jonka toiminnallisuudet ovat rajallisia, kunnes seuraavissa versioissa toiminnallisuuksia lisätään. (Monash & Dwolatzky, 2007)

Neljäs yhteinen piirre ketterissä menetelmissä on itseohjautuvat kehitystiimit. Itseohjautuvuus tarkoittaa sitä, että yksilöt tiimin sisällä hallitsevat itse omaa työkuormaansa ja jakavat työtehtäviä tiimin sisällä sen mukaan, mikä tehtävä yksittäiselle ryhmänjäsenelle parhaiten sopii (Highsmith, 2009). Tämän takia ketterissä ohjelmistoprojekteissa projektipäällikön rooli muuttuu perinteisestä hallitsevasta roolista fasilitaattorin ja kollegan rooliin (Hoda, Noble & Marshall, 2010).

2.1 Extreme Programming (XP)

Extreme Programming (XP) on laajimmin tunnettu ketterä ohjelmistokehitysmenetelmä (Boehm & Turner, 2003), jonka on kehittänyt yhdysvaltalainen Kent Beck 1990-luvun lopulla. Kuten muissakin ketterissä ohjelmistokehitysmenetelmissä, myös XP:ssä kehitystyö perustuu nopeisiin iteraatioihin, pieniin ja nopeisiin julkaisuihin sekä läheiseen yhteistyöhön asiakkaan kanssa. Beckin ja Andresin (2004) mukaan XP on kevyt menetelmä, joka keskittyy itse ohjelmiston kehittämiseen, eikä ota kantaa projektin johtamiseen tai taloudellisiin asioihin. XP vastaa hyvin asiakkaan epäselviin ja nopeasti muuttuviin vaatimuksiin ja toimii erikokoisissa kehitystiimeissä (Beck & Anders, 2004). Caon, Mohanin ja Rameshin (2004) mukaan XP:n etuja perinteisiin ohjelmistokehitysmenetelmiin verrattuna ovat pienemmät kiinteät kustannukset, tiimien korkeampi tuottavuus sekä lyhyemmät julkaisusyklit.

XP perustuu neljälle arvolle, jotka ovat yksinkertaisuus, kommunikaatio, palaute sekä rohkeus. XP koostuu myös käytänteistä, joista esimerkkejä ovat pariohjelmointi, jatkuva integraatio, refaktorointi, suunnittelupelit, testilähtöinen kehittäminen(TDD, Test-driven development) ja käyttötapaukset. (Beck,

2004) Myöhemmin tutkielmassa tarkastellaan tarkemmin XP:n käytänteitä osana ohjelmiston laadunvarmistusta.

Jokainen XP:n iteraatio alkaa valitsemalla asiakkaan kanssa tärkeimmät kehitettävät vaatimukset, jotka on kirjoitettu käyttötapauksiksi. Tämän jälkeen valituista käyttötapauksista kirjoitetaan testit, jotka ovat hyväksymisperuste iteraatiossa syntyneelle ohjelmistolle tai sen osalle. (Macias, Holcombe & Gheorge, 2003) Asiakas siis määrittää käyttötapauksista luotujen testien avulla sen, minkälainen ohjelmiston tulisi olla iteraation lopussa. Kun asiakas on itse suorittamassa kyseisiä testejä iteraation lopussa, voi hän helposti tarkastella ohjelmiston kehityksen vaihetta ja määrittää vaatimuksia uudestaan. (Beck, 1999) XP:ssä korostetussa roolissa onkin ohjelmiston tarkka testaus. (Cao ym., 2004)

Kuten kaikissa ketterissä ohjelmistokehitysmenetelmissä, myös XP:ssä kommunikaatiolla on suuri rooli asiakkaiden ja kehittäjien välillä. XP:ssä painotetaan kasvokkain tapahtuvaa kommunikaatiota esimerkiksi sähköisillä välineillä tapahtuvan kommunikaation sijaan. Kasvokkain tapahtuva kommunikaatio lisää ideoiden ja ajatusten vaihtoa projektissa toimivien ihmisten välillä (Macias ym., 2003) Yksi XP:n käytänteistä onkin se, että asiakkaan tulisi olla koko projektin ajan kehitystiimin käytettävissä samoissa tiloissa (Beck, 1999).

2.2 Scrum

Scrum on XP:n ohella käytetyimpiä ketteriä ohjelmistokehitysmenetelmiä (Rech, 2007) ja sen ovat kehittäneet Ken Schwaber ja Jeff Sutherland vuonna 1996. Toisin kuin XP, joka keskittyy itse ohjelmiston kehittämiseen, Scrum tarjoaa projektijohtamisen viitekehyksen (Abbas, 2009). Kuten muutkin ketterät menetelmät, myös Scrum jakaa ohjelmiston kehitysprojektin iteraatioihin, joita kutsutaan sprinteiksi. Ennen varsinaisten sprinttien alkua Scrumissa suoritetaan alustavat selvitystyöt toteutettavasta ohjelmasta. Tämän selvitystyön perusteella ohjelmistolle luodaan arkkitehtuuri, jonka pohjalta ohjelmistoa aletaan varsinaisten sprinttien aikana toteuttamaan. (Rising & Janoff, 2000) Sprintit pitävät sisällään seuraavia vaiheita (Cohen ym., 2004):

1. Sprintin suunnittelupalaveri, jossa valitaan kehitysjonosta tulevassa sprintissä toteutettavat ohjelman osat. Suunnittelupalaverissa luodaan sprintille myös päämäärä, joka auttaa kehittäjiä luomaan selkeämmän kuvan siitä, mitä kyseisessä sprintissä ollaan kehittämässä.
2. Sprint on vaihe, jossa suunnittelupalaverissa sovitut toteutettavat ohjelman osat toteutetaan. Suunnittelupalaverissa sovittuja toteutettavia ohjelman osia ei muuteta sprintin aikana, vaan kehittäjille annetaan työrauha näiden toteuttamiseen. Sprintin aikana pidetään päivittäin Scrum-palaveri, jossa seurataan sprintin kehittymistä.
3. Sprinttikatselmus pidetään jokaisen sprintin jälkeen. Siinä analysoidaan sprintin onnistumista, sekä esitellään ohjelman uusin ver-

> sio asiakkaalle. Tämän lisäksi tiimi pitää vielä sprintin retrospektiivin, jolla varmistetaan ohjelmistokehitysprosessin jatkuva kehitys (Mahnik & Zabkar, 2008).

Scrumissa on kolme ainoastaan Scrumissa esiintyvää roolia. Kaikki tiimiin kuuluvat ovat osa Scrum-tiimiä, jolla tarkoitetaan Scrumin periaatteiden mukaan toimivaa, itseohjautuvaa kehitystiimiä. Tiimiin kuuluu yleensä viidestä yhdeksään henkilöä. Perinteisen projektipäällikön rooli on Scrumissa nimeltään scrum-mestari (Scrum Master), jonka tehtävänä on ylläpitää prosessia Scrumin periaatteiden mukaisena, sekä estää ulkopuolelta tulevia häiriöitä haittaamasta Scrum-tiimin toimintaa. Näiden roolien lisäksi Scrumissa on tuoteomistaja (Product Owner), joka on vastuussa asiakkaan vaatimusten hallinnasta. (Heidenberg, 2011)

3 LAATU JA LAADUNVARMISTUS

Jotta laadunvarmistusta ohjelmistokehityksessä voidaan tarkastella, tulee ensin määrittää mitä laatu on ja mitä sillä tarkoitetaan ohjelmistokehityksen kontekstissa. Kirjallisuudesta löytyy monia määritelmiä laadulle niin yleisenä käsitteenä, kuin ohjelmiston laadulle. Laatua voidaan myös tarkastella eri näkökulmista, kuten asiakkaan ja kehittäjien näkökulmasta. Seuraavassa alaluvussa esitellään kolme kirjallisuudessa usein esiintyvää yleisen tuotteen laadun määritelmää.

Juran ja Grynan (1988) mukaan laatu tarkoittaa sitä miten sopiva tuote on sen tarkoitettuun käyttöön. Juranin ja Grynan (1988) laajennetussa määritelmässä huomioidaan myös, että tuote täyttää ominaisuuksiltaan ja piirteiltään asiakkaan tarpeet. Määritelmän mukaan tuotteella on myös oltava mahdollisimman vähän puutteita ja heikkouksia. Gilliesin (1992) mukaan laatu on näkymätöntä silloin, kun tuote on laadukas, mutta laadun puute on helposti havaittavissa. Hossainin ym. (2013) mukaan laatu rakentuu niistä tuotteen ominaisuuksista, jotka täyttävät asiakkaan tarpeet ja sitä myötä tuottavat tyydytystä.

3.1 Laadun määritelmä ohjelmistokehityksessä

Ohjelmiston laadun absoluuttinen määrittäminen on Gilliesin (1992) mukaan vaikeaa verrattuna muihin aloihin. Tämä johtuu siitä, että ohjelmisto ei ole fyysinen objekti, asiakkaat eivät tiedä tarkasti mitä he haluavat, asiakkaan tarpeet muuttuvat ajan kuluessa, tekniikka kehittyy hyvin nopeasti sekä asiakkailla on yleisesti suuret odotukset ohjelmistotuotteita kohtaan.

Radatz ym. (1990) taas määrittävät ohjelmiston laadun olevan aste, jolla ohjelmisto, komponentti tai prosessi täyttää sille asetut vaatimukset sekä asiakkaan tarpeet ja odotukset. Kanin (2002) mukaan ohjelmiston laatu koostuu kolmesta tekijästä. Nämä tekijät ovat ohjelmistotuotteen laatu, prosessin laatu, sekä asiakastyytyväisyys.

Ohjelmiston laatu, kuten laatu yleisestikin, tarkoittaa eri ihmisille eri asiaa. Esimerkiksi asiakkaalle laatu voi tarkoittaa sitä kuinka hyvin ohjelmisto täyttää

hänen odotuksensa, tai sitä, kuinka nopea tai halpa tuote on käyttää. Ohjelmiston suunnittelijalle laatu voi taas tarkoittaa sitä, minkälainen tekninen toteutus ohjelmistolla on tai, miten hyvin ohjelmisto on dokumentoitu. (Gillies, 1992) Esimerkiksi Heidenberg (2011) määrittää laadukkaan ohjelmiston olevan ohjelmisto, jossa on vähän virheitä ja että se tarjoaa mukavan ja luotettavan käyttäjäkokemuksen. Heidenberg (2011) laajentaa määritystään koskemaan myös kehittäjän näkökulmaa. Kehittäjän näkökulmasta laadukkaassa ohjelmistossa on otettava huomioon myös kirjoitetun koodin laadukkuus. Kirjoitetun koodin laatu ei välttämättä näy käyttäjälle, eikä vaikuta käyttökokemukseen, mutta se voi vaikuttaa esimerkiksi ohjelmiston ylläpidettävyyteen, sekä mahdolliseen uudelleenkäytettävyyteen (Rech, 2007).

ISO 9126 laatustandardi (International Organisation for Standardization, 2001) määrittää kuusi kriteeriä ohjelmiston laadulle. Nämä kuusi kriteeriä ovat toiminnallisuus, luotettavuus, tehokkuus, käytettävyys, ylläpidettävyys ja siirrettävyys. Gilliesin (1992) määrityksen mukaan ohjelmiston laatua voidaan tarkastella asiakkaan ja käyttäjän näkökulmista ja myös ISO 9126 laatustandardin asettamat kriteerit voidaan jakaa näiden näkökulmien mukaisesti. Asiakkaan näkökulmasta tärkeitä ohjelmiston laadullisia kriteerejä ovat toiminnallisuus, luotettavuus, tehokkuus sekä käytettävyys, kun taas kehittäjän näkökulmasta tärkeitä kriteerejä ovat ylläpidettävyys sekä siirrettävyys.

Kirjallisuudessa esitettyjen määritelmien mukaan ohjelmiston laatu on laaja käsite, joka kattaa niin itse ohjelmistotuotteeseen, kuin kehitysprosessiin liittyviä asioita. Ohjelmiston laatua voidaan myös käsitellä asiakkaan ja kehittäjän näkökulmasta, jolloin laatukriteerit voivat olla eriäviä. Tässä tutkielmassa ei käsitellä ohjelmistokehitysprosessin laatua, vaan keskitytään tekniikoihin ja menetelmiin, joilla voidaan varmistaa, että ISO 9126 laatustandardin mukaiset kriteerit voidaan täyttää ohjelmiston kehityksessä ja ylläpidossa.

3.2 Laadunvarmistus

Ennen kuin tarkastellaan miten ketterissä menetelmissä varmistutaan ohjelmiston laadusta ja minkälaisia tekniikoita ja menetelmiä ne tarjoavat laadunvarmistukseen, on syytä määritellä mitä laadunvarmistuksella ohjelmistokehityksessä tarkoitetaan. Tässä alaluvussa käydään läpi kirjallisuudesta löytyviä määritelmiä ohjelmiston laadunvarmistukselle, sekä tarkastellaan lyhyesti perinteisten ja ketterien ohjelmistokehitysmenetelmien eroja laadunvarmistusprosessin näkökulmasta.

Kirjallisuudessa esiintyy monia määritelmiä laadunvarmistukselle, joista Owens ja Khazanci (2009) ovat koostaneet oman määritelmänsä. Sen mukaan laadunvarmistus on hyvin määritelty, toistettavissa oleva prosessi, joka on integroitu niin projektin johtamisen, kuin ohjelmiston elinkaaren aikana tapahtuviin toimiin. Näillä toimilla varmistutaan siitä, että ohjelmistolle asetettuja vaatimuksia noudatetaan, vähennetään ohjelmistoon kohdistuvia riskejä, parannetaan ohjelmiston laatua sekä pysytään aikataulussa ja budjetissa.

Hossainin ym. (2013) määritelmän mukaan ohjelmiston laadunvarmistus on systemaattinen ja suunniteltu kokoelma toimintoja, joilla voidaan saavuttaa riittävä varmuus siitä, että ohjelmisto täyttää sille asetetut toiminnalliset vaatimukset, kuten myös johdon asettamat vaatimukset budjetin ja aikataulun suhteen. Termillä ohjelmiston laadunvarmistus tarkoitetaan monesti vain ohjelmiston testaukseen liittyviä toimia, mutta Owensin ja Khazachin(2009) mukaan ohjelmiston laadunvarmistus liittyy olennaisena osana kaikkiin projektin vaiheeseen ohjelmiston elinkaaren aikana.

Kirjallisuudessa painotetaan laadunvarmistuksen tärkeyttä ohjelmistokehityksessä. Esimerkiksi Owenin ja Khazanchin (2009) mukaan ohjelmiston laadunvarmistus on kriittisessä roolissa koko ohjelmiston elinkaaren aikana ja sillä voi olla suuri vaikutus koko ohjelmistoprojektin onnistumiseen. Jos ohjelmiston laadunvarmistukseen ei kiinnitetä tarpeeksi huomiota, se voi johtaa budjetin ylitykseen, aikataulun viivästyksiin, projektin tavoitteiden saavuttamattomuuteen ja asiakastyytyväisyyden kärsimiseen (Chow, 1985).

Laadunvarmistuksella pyritään siis varmistumaan siitä, että ohjelmisto valmistuu ajallaan sekä budjetin määrittämissä rajoissa. Ohjelmiston on myös täytettävä sille asetetut toiminnalliset vaatimukset ja sen tulisi sisältää mahdollisimman vähän virheitä. Hossainin ym. (2013) mukaan laadunvarmistuksen pääasiallinen tarkoitus ei ole löytää virheitä ohjelmistosta, vaan välttää virheiden syntymistä.

Ohjelmiston laadunvarmistustekniikat voidaan jakaa kahteen kategoriaan, jotka ovat staattiset ja dynaamiset tekniikat. Staattisilla tekniikoilla tarkoitetaan sitä, että ohjelmiston koodia ei suoriteta, vaan sitä tutkitaan manuaalisesti ja erilaisin työkaluin. Dynaamisissa tekniikoissa, kuten testaamisessa, ohjelmakoodin laatua tutkitaan suorittamalla ohjelmakoodi. (Hossain ym., 2013) Staattisia ja dynaamisia tekniikoita voidaan käyttää tukemaan toisiaan, mutta Huon (2007) mukaan staattisia tekniikoita käytetään enemmän perinteisissä ohjelmistokehitysmenetelmissä ja dynaamisia tekniikoita ketterissä ohjelmistokehitysmenetelmissä. Kuitenkin esimerkiksi XP:ssä laajasti käytetty tekniikka refaktorointi on staattinen tekniikka (Mkandla & Dwolatzki, 2007).

Galinin (2003) mukaan ohjelmiston laadunvarmistuksessa on mukana kolme sidosryhmää, jotka ovat yrityksen ja projektin johto, testaajat sekä erillisen laadunvarmistustiimin jäsenet. Huon ym. (2007) mukaan laadunvarmistusta on perinteisissä ohjelmistokehitysmenetelmissä pidetty erillisen tiimin suorittamana erillisenä toimintona. Myös Owens ja Khazanchi (2009) painottavat erillistä kehitystiimistä irrallaan olevaa laadunvarmistustiimiä.

Toisaalta Abbasin (2009) mukaan laadunvarmistuksen pitäisi olla integroituna koko ohjelmistokehitysprosessiin, ja laadunvarmistuksen pitäisi olla kaikkien projektiin osallistuvien vastuulla. Ketterissä menetelmissä laadunvarmistus onkin integroitu projektin eri toimintoihin koko kehitysprojektin ajalle ja siitä ovat vastuussa kaikki kehitystyöhön osallistuvat (Stamelos & Sfetsos, 2007). Laadunvarmistuksen integrointi projektin toimintoihin ei kuitenkaan tarkoita sitä, ettei yrityksellä voisi olla myös erillistä laadunvarmistustiimiä.

4 LAADUNVARMISTUS KETTERISSÄ OHJELMISTOKEHITYSMENETELMISSÄ

90-luvun lopun jälkeen yrityksissä on alettu käyttämään yhä enemmän ketteriä ohjelmistokehitysmenetelmiä perinteisten suunnittelupohjaisten menetelmien sijaan (Huo ym., 2004). Markin (2007) mukaan ketterien menetelmien yksi tärkeimmistä pääperiaatteista on se, että tuottava ohjelmisto on laadukasta. Onkin syytä tarkastella millä keinoin ketterissä menetelmissä laadukasta ohjelmistoa voidaan valmistaa. Luvussa ”Ketterien menetelmien laadunvarmistuskäytänteet” esitellään kahdeksan kirjallisuudessa esiintyvää käytännettä, joilla ketterät menetelmät vastaavat laadunvarmistuksen haasteisiin. Luvussa ”Ketterien menetelmien laadunvarmistuksen riittävyys” tarkastellaan mitä kirjallisuudessa sanotaan ketterien menetelmien laadunvarmistuksen riittävyydestä.

Perinteisissä ohjelmistokehitysmenetelmissä laadunvarmistus on ollut pääasiassa erillisen laadunvarmistustiimin vastuulla ja laadunvarmistustyö on tapahtunut enimmäkseen projektin loppuvaiheessa testauksen avulla (Huo ym., 2004). Ketterissä menetelmissä laadunvarmistus tapahtuu koko projektin elinkaaren aikana ja on pääasiassa integroitu ohjelmiston kehitystyöhön (Mark, 2007). Laadunvarmistuksen integrointi ohjelmiston kehitystyöhön tarkoittaa sitä, että vastuu ohjelmiston laadusta on koko kehitystiimillä, eikä vain erillisellä laadunvarmistustiimillä, kuten perinteisessä menetelmissä (Huo ym., 2004). Monissa ketterien menetelmien käytänteissä on ominaisuuksia, jotka edesauttavat laadukkaan ohjelmiston kehitystä (Hossain ym., 2013) Näitä käytänteitä esitellään alaluvussa ”Ketterien menetelmien laadunvarmistuskäytänteet”.

Ketterien menetelmien etuna on se, että laadunvarmistustyö alkaa aikaisemmin kuin perinteisissä ohjelmistokehitysmenetelmissä. Tämä johtuu siitä, että ketterissä menetelmissä valmista ohjelmistoa syntyy aikaisemmassa vaiheessa. (Huo ym., 2004) Tämä valmis ohjelmiston osa on jo kehityksen alusta asti tuotettu käytäntein, joissa laadunvarmistus on olennaisena osana mukana (Hossain ym.,2013).

Ketterissä menetelmissä asiakkaan kanssa tapahtuva tiivis yhteistyö on olennainen osa ohjelmiston kehitystä. Asiakas pääseekin valmiin ohjelmiston osan avulla näkemään onko ohjelmisto sellainen kun hän on ajatellut ja muokkaamaan vaatimuksiaan ohjelmistolle. Muuttuneiden vaatimusten pohjalta ke-

hitystiimi voi muokata seuraavassa iteraatiossa ohjelmistoa vastaamaan asiakkaan muuttuneita vaatimuksia. (Abbas, 2009) Perinteisissä menetelmissä asiakas näkee ohjelmiston vasta kehitysprojektin lopussa, jolloin muutosten tekeminen on kallista ja vaikeaa.

Ketterissä menetelmissä laadunvarmistustyötä tapahtuu myös huomattavasti useammin kuin perinteisissä menetelmissä. Tämä johtuu siitä, että ohjelmistoa kehitetään iteraatioissa ja jokaiseen iteraatioon kuuluu käytänteitä, joilla on laadunvarmistuksellisia ominaisuuksia. (Hossain ym.,2013)

Huon ym. (2004) mukaan ketterien menetelmien laadunvarmistuksessa on kolme pääpiirrettä, jotka ovat tärkeitä käytettäessä ketteriä menetelmiä ohjelmiston luomiseen. Ensimmäinen piirre on dynaamisten laadunvarmistusmenetelmien käyttö mahdollisimman aikaisin kehityksessä. Dynaamisten menetelmien käyttö aikaisessa vaiheessa on mahdollista, koska ohjelmiston implementaatiovaihe alkaa jo aikaisessa vaiheessa. Toinen piirre on se, että laadunvarmistusta siirretään enemmän kehittäjien vastuulle. Kolmas piirre on se, että asiakas vahvistaa tuotteen oikeellisuuden mahdollisimman aikaisessa vaiheessa, jotta kehittäjät voivat varmistua siitä, että he ovat kehittämässä oikeaa ohjelmaa.

4.1 Ketterien menetelmien laadunvarmistuskäytänteet

Tässä luvussa esitellään kahdeksan eri Scrumissa ja XP:ssä esiintyvää käytännettä, joilla on laadunvarmistuksellisia ominaisuuksia, eli jotka omalta osaltaan auttavat kehittämään laadukkaan ohjelman. Esiteltävät kahdeksan käytännettä on valittu, koska ne esiintyvät kirjallisuudessa eniten, kun tarkastellaan ketterien menetelmien laadunvarmistusta. Seuraavissa kappaleissa käsitellään valittuja käytänteitä, sekä Liitteeseen 1 on kerätty kirjallisuudesta näiden kahdeksan käytänteen vaikutukset ohjelmistokehitykseen. Liitteeseen 2 on taas koottu (Hossain ym., 2013; Cohen ym., 2004; Galin, 2004; Huo ym.,2004; Cao ym., 2004) taulukko siitä, mihin ISO 9126 laatukriteereihin mikäkin edellä mainituista käytänteistä vaikuttaa positiivisesti.

Testilähtöinen kehittäminen(TDD) on XP:stä lähtöisin oleva käytänne, joka on myös levinnyt muihin ketteriin menetelmiin. TDD:ssä perusperiaate on se, että asiakkaan ja kehittäjien laatimien käyttötapausten pohjalta luodaan testitapauksia ennen varsinaisen ohjelmointityön aloittamista. Vasta kun testitapaukset ovat luotu ohjelmoija voi alkaa kirjoittamaan ohjelmistokoodia. (Cohen, 2004) Puhtaimmassa muodossa TDD tarkoittaa sitä, että ohjelmoija kirjoittaa ensin ohjelmaosion testit, jotka epäonnistuvat, koska toteutusta kyseiselle ohjelmanosalle ei ole vielä tehty. Tämän jälkeen ohjelmoija kirjoittaa ohjelmakoodia kunnes kaikki testitapaukset menevät läpi. (Mark, 2007)

Metafora on XP:stä lähtöisin oleva tarina, jonka kehittäjät, asiakkaat ja projektipäälliköt voivat kertoa siitä kuinka ohjelma toimii (Hossain ym., 2013). Metaforan käyttö auttaa kyseisiä sidosryhmiä kommunikoimaan, sillä metaforaa ei kerrota teknisellä kielellä. Metaforaa käytetäänkin ketterissä menetelmissä formaalin teknisen arkkitehtuurin kuvauksen sijaan, sillä asiakkaan voi olla vaikea ymmärtää teknistä kuvausta ohjelmistosta. (Huo ym., 2004) Metafora on tärkeä

käytänne, koska sen käyttö auttaa kehittäjiä jalostamaan vaatimuksia projektin aikana (Cohen, 2007) ja Galininin(2004) mukaan sen käyttö ehkäisee myös kehittäjien tarkoituksellista poikkeamista asiakkaan vaatimuksista. Kun metafora on asiakkaan kanssa yhteistyössä luotu, kehittäjät käyttävät metaforan avulla saatua yleiskuvaa ohjelmiston alustavan arkkitehtuurin luomiseen (Huo ym.,2004).

Ketterissä menetelmissä painotetaan asiakkaan tiivistä osallistumista koko kehitysprojektiin. Myös XP:ssä ja Scrumissa löytyy useita käytänteitä, joissa painotetaan asiakkaan osallistumista. Näitä ovat esimerkiksi läsnä oleva asiakas(On-Site-Customer), iteraation suunnittelu, metafora, suunnittelupelit, sprinttikatselmus (Cohen, 2004). Osaa näistä käytänteistä käsitellään tarkemmin omina kappaleinaan, mutta koska kirjallisuudessa painotetaan asiakkaan osallistumista tärkeänä osana laadukkaan ohjelmiston kehittämistä, on syytä käsitellä asiakkaan osallistumista yhtenä ketterien menetelmien käytänteistä. Asiakkaan tiivis osallistuminen koko projektin aikana auttaa saamaan tärkeää tietoa kehittäjille jo hyvissä ajoin, jolloin voidaan alusta asti kehittää oikeanlaista ohjelmaa (Huo ym., 2004). Hossainin ym. (2013) mukaan ihannetilanteessa asiakas on osana kehitystiimiä, jolloin asiakas voi jalostaa vaatimuksia, vastata kehittäjien kysymyksiin, ratkaista erimielisyyksiä sekä priorisoida vaatimuksia. Kun asiakas on koko projektin ajan läsnä on hänen myös helppo seurata projektin kehittymistä (Cohen, 2004)

Yksi tärkeistä ohjelmiston laadullisista kriteereistä on ylläpidettävyys. Ketterissä menetelmissä ylläpidettävyyttä parantaa XP:stä lähtöisin oleva ohjelmakoodin refaktorointi. Refaktorointi on tekniikka, jolla muutetaan ohjelmakoodin sisäistä rakennetta muuttamatta kuitenkaan sen ulkoista toimintaa(Hossain ym., 2013). Jos kaikki testit eivät mene refaktoroinnin jälkeen läpi, on refaktoroitu ohjelmakoodi hylättävä, sillä tässä tapauksessa ohjelman ulkoinen toiminta on muuttunut (Cohen, 2004). Refaktoroinnin ideana on parantaa ohjelmakoodin sisäistä rakennetta, jotta siitä tulisi selkeämpää, yksinkertaisempaa ja paremmin suunniteltua (Hossain ym. 2013). Rechin (2007) mukaan refaktorointi tekee kehittämisestä ja ylläpitämisestä helpompaa ja refaktoroinnin avulla voidaan vähentää kuluja, työtä sekä kehitysaikaa.

Ohjelmakoodissa olevat suunnitteluvirheet vaikuttavat ohjelmiston toiminnalisuuteen, tehokkuuteen, ja luotettavuuteen. XP:ssä yritetään välttää virheitä pariohjelmoinnin avulla. Pariohjelmointi tarkoittaa sitä, että kaksi ohjelmoijaa ovat saman tietokoneen ääressä ja kirjoittavat yhdessä ohjelmakoodia (Cohen, 2004). Ohjelmoijista toinen kirjoittaa koodia ja toinen tarkkailee sekä kommentoi syntynyttä koodia. Tämän jälkeen ohjelmoijat vaihtavat paikkaa. (Hossain ym., 2013) Huon ym. (2004) mukaan pariohjelmoinnissa syntyy huomattavasti vähemmän virheitä, kuin normaalissa yksin tapahtuvassa ohjelmoinnissa. Hossain ym. (2013) toteavat myös, että pariohjelmoinnin avulla ohjelmistossa olevat virheet voidaan huomata tehokkaammin jo ohjelmoinnin aikana. Galinin (2004) mukaan pariohjelmoinnin avulla vältetään myös sitä, että ohjelmoijat jättävät noudattamatta yhteisesti sovittuja dokumentointi- ja ohjelmointiohjeita.

Ohjelmiston moduuleiden integraatio aiheuttaa ongelmia toiminnallisuudessa ja luotettavuudessa. Perinteisissä kehitysmenetelmissä integraatio on

yleensä toteutettu projektin loppuvaiheessa ja siitä on voinut seurata suuriakin ongelmia. Ketterissä menetelmissä tämä ongelma on ratkaistu käyttämällä jatkuvaa integraatiota, eli ohjelmisto pidetään täysin integroituna koko kehityksen ajan (Huo ym., 2004). Kehittäjät integroivat heidän työnsä ohjelmistoon vähintään kerran päivässä, jonka jälkeen jokaisen integraation jälkeen ohjelmisto testataan. Tämän seurauksena ihannetilanteessa käytössä on toimiva ohjelmisto koko kehityksen ajan. (Hossain ym., 2013)

Radatz ym. (1990) määrittävät ohjelmiston laadun olevan aste, jolla ohjelmisto, komponentti tai prosessi täyttää sille asetut vaatimukset sekä asiakkaan tarpeet ja odotukset. Jotta asiakkaan tarpeet täyttyvät on ohjelmiston vastattava asiakkaan asettamia vaatimuksia. Ketterissä menetelmissä asiakkaan tarpeiden täyttymisestä varmistutaan hyväksymistestauksella. Hyväksymistestaus on erityisesti XP:ssä käytettävä testauksen muoto, jossa testataan onko asiakkaan luomat käyttötapaukset toteutettu ohjelmistossa. Kun käyttötapaus on asiakkaan toimesta testattu ja hyväksytty, voidaan tietty ohjelmiston osa määritellä valmiiksi. (Hossain ym., 2013) Toisin kuin perinteisissä menetelmissä hyväksymistestausta ei tehdä ketterissä menetelmissä vain projektin lopussa, vaan hyväksymistestausta tehdään useasti koko projektin ajan (Huo ym., 2004). Koska asiakas luo käyttötapaukset, joiden mukaan hyväksymistestaus toteutetaan, on käyttötapaukset useasti vain ohjelmiston toiminnallisia vaatimuksia. Monochristoun ja Vlachopouloun (2007) mukaan ohjelmistosta on testattava näiden toiminnallisten vaatimusten lisäksi myös muita vaatimuksia.

Scrumissa ja XP:ssä laajasti käytetyt suunnittelupelit vähentävät virheellistä vaatimusmäärittelyä (Galin, 2004), joka voi johtaa siihen, että kehitetty ohjelmisto ei vastaa asiakkaan tarpeita. Suunnittelupelien tarkoituksena on tuottaa korkeatasoinen suunnitelma seuraavaan julkaisuun tai iteraatioon, jolla voidaan tuottaa mahdollisimman paljon arvoa asiakkaalle (Monochristou & Vlachopoulou, 2007). Suunnittelupelejä käytetään jokaisen iteraation ja julkaisun alussa. Niissä asiakas ja kehitystiimi arvioivat ja priorisoivat vaatimuksia.(Cohen, 2004) Suunnittelupelien avulla ohjelmistoon saadaan kussakin iteraatiossa tai julkaisussa ominaisuudet, jotka ovat asiakkaan kannalta tärkeimpiä. Tämän seurauksena ohjelmistosta saatu arvo asiakkaalle kasvaa ja asiakkaan kokema laatu sitä myötä paranee. (Monochristou & Vlachopoulou, 2007)

4.2 Ketterien menetelmien laadunvarmistuksen riittävyys

Ketterissä menetelmissä on alaluvussa 4.1 esiteltyjen käytänteiden perusteella paljon laadunvarmistusta sisältäviä toimintoja, joilla pyritään varmistamaan, että tuotetut ohjelmistot ovat laadukkaita. On kuitenkin syytä tarkastella, ovatko ketterien menetelmien laadunvarmistuskäytänteet riittäviä vai tarvitaanko perinteisistä menetelmistä tuttuja erillisiä laadunvarmistustiimejä ja toimintoja. Kirjallisuudesta löytyy paljon tutkimusta, joiden mukaan ketterien menetelmien tarjoamat käytänteet tarjoavat paremmat edellytykset laadukkaan ohjelman toteuttamiseksi. Tutkimusta, jonka mukaan ketterien menetelmien laadunvarmistus itsessään ei ole riittävää löytyy huomattavasti vähemmän.

Amblerin(2005) mukaan ketterien menetelmien käyttö johtaa huomattavasti korkeampaan laatuun kuin perinteisissä menetelmissä. Hänen mukaansa ketterien menetelmien painotus laadukkaan ohjelmiston luomiseen johtaa siihen, että erillisten laadunvarmistustiimien merkitys ohjelmistotuotannossa vähenee. Hossain ym. (2013) ja Huo ym. (2004) ovat samaa mieltä Amblerin (2005) esittämien väittämien kanssa. Heidän mukaansa ketterät menetelmät tuottavat laadukasta ohjelmistoa sen takia, että ketterissä menetelmissä laadunvarmistuksellisia ominaisuuksia sisältäviä käytänteitä esiintyy huomattavasti aikaisemmassa vaiheessa ja useammin, kuin perinteisissä menetelmissä. Myös Abbas (2009) on samaa mieltä ketterien menetelmien laadunvarmistuksen riittävyydestä, kun puhutaan tuotteen laadusta. Hänen mukaansa kuitenkin prosessin laatua ei ole vielä tutkittu tarpeeksi, jotta voitaisiin todeta, että ketterillä menetelmillä voidaan varmistua riittävästä prosessin laadusta.

Kirjallisuudesta löytyy melko vähän tutkimustuloksia, joiden mukaan ketterissä menetelmissä ei voida varmistua riittävästi ohjelmiston laadusta. Varsinkin kun verrataan perinteisiin menetelmiin, tutkijat ovat pääasiassa samaa mieltä siitä, että ketterillä menetelmillä voidaan tuottaa laadukkaampaa ohjelmistoa, kuin perinteisillä menetelmillä. Huon ym. (2004) ketterät menetelmät ovat riittäviä tuottamaan laadukasta ohjelmaa, mutta he kuitenkin toteavat, että kehittäjien on kuitenkin koko ajan oltava valmiina muokkaamaan käytössä olevia menetelmiä, jotta kehitetyn ohjelmiston laatu paranee. Jyothi ja Rao (2011) ovat myös sitä mieltä, että ketterien menetelmien laadunvarmistus on riittävää. Heidän mukaansa olisi kuitenkin hyvä käyttää erillistä laatufasilitaattoria, joka valvoo, että ohjelmistoa kehitetään määrättyjen sääntöjen mukaan. Tämän lisäksi laatufasilitaattorin tehtävänä on kehittää käytänteitä niin, että tuotettujen ohjelmistojen laatu paranee.

5 YHTEENVETO

Tässä tutkielmassa käsiteltiin laadunvarmistusta ketterissä ohjelmistokehitysmenetelmissä. Tutkielma ja se luvut rakentuivat niin, että Luvussa 2 määriteltiin mitä ketterät menetelmät ovat ja kuinka niiden avulla voidaan valmistaa ohjelmistoja. Luvussa tutustuttiin tarkemmin kahteen ketterään menetelmään, jotka ovat XP ja Scrum. Luvussa 3 esiteltiin kirjallisuudesta löytyviä määritelmiä laadulle, ohjelmiston laadulle sekä laadunvarmistukselle. Luvussa 4 tarkasteltiin laadunvarmistusta ketterissä menetelmissä. Luvussa esiteltiin kahdeksan ketteristä menetelmistä tuttua käytännettä, joilla ketterissä menetelmissä pyritään varmistumaan ohjelmiston laadusta.

Tutkielman tutkimuskysymykset olivat seuraavat: Kuinka ketterissä menetelmissä voidaan varmistua ohjelmiston laadusta? Ovatko ketterien menetelmien laadunvarmistuskäytänteet riittäviä ohjelmistonlaadun takaamiseksi? Tutkimusmenetelmänä käytettiin kirjallisuuskatsausta ja tutkielman lähdeaineisto koostuu pääasiassa alaa käsittelevistä tieteellisistä artikkeleista.

Tutkielmassa kerrottiin kuinka ketterät ohjelmistokehitysmenetelmät ovat yleistyneet vahvasti 90- luvun lopun jälkeen. Tämä murros on aiheuttanut tarpeen tutkia ketteriä menetelmiä tarkemmin. Tutkielmassa todettiin, että ketteriä menetelmiä edeltäneissä perinteisissä menetelmissä laadunvarmistus eroaa suuresti. Perinteisten menetelmien laadunvarmistusta on tutkittu pitkään ja tutkimustietoa on saatavilla paljon, kun taas ketterien menetelmien laadunvarmistuksesta ei ole saatavilla yhtä kattavaa tutkimustietoa.

Tutkielmassa esiteltiin ketterien menetelmien yhteisiä periaatteita ja arvoja, sekä ketterien menetelmien historiaa. Ketterien menetelmien keskeisimpiä arvoja ovat: yksilöitä ja kanssakäymistä enemmän kuin menetelmiä ja työkaluja, toimivaa ohjelmistoa enemmän kuin kattavaa dokumentointia, asiakasyhteistyötä enemmän kuin sopimusneuvotteluita ja vastaamista muutokseen enemmän kuin pitäytymistä suunnitelmassa. Tutkielmassa käytiin tarkemmin läpi kahta ketterää menetelmää, jotka ovat Xp ja Scrum. Nämä valittiin tutkielmaan niiden suosionsa perusteella.

Ketteristä menetelmistä muodostetun yleiskuvan jälkeen, tutkielmassa tarkasteltiin minkälaisia määritelmiä kirjallisuudesta löytyy laadulle, ohjelmiston laadulle sekä laadunvarmistukselle. Yleisien määritelmien esittelyn jälkeen

tutkielmassa tutustuttiin tarkemmin ohjelmiston laatuun ja laadunvarmistukseen. Ohjelmiston laatua pohdittiin eri sidosryhmien näkökulmasta, sekä esiteltiin ISO 9126 standardin määrittämät kriteerit ohjelmiston laadulle. Laadunvarmistusta tarkasteltiin jakamalla laadunvarmistusmenetelmät dynaamisiin ja staattisiin menetelmiin, sekä pohdittiin miten perinteisten ja ketterien menetelmien laadunvarmistus eroaa toisistaan.

Tämän jälkeen tutkielmassa esiteltiin ketterien menetelmien laadunvarmistusta yleisellä tasolla, sekä tutustuttiin kahdeksaan käytänteeseen, joilla ketterissä menetelmissä pyritään varmistumaan ohjelmiston laadusta. Näiden kahdeksan käytänteen hyötyjä ohjelmiston laadulle esitellään Liitteessä 1. Liitteessä 2 näitä kahdeksaa käytännettä tarkastellaan sen mukaan vaikuttaako kukin laadunvarmistuskäytänne kuuteen ISO 9126 mukaiseen laatukriteeriin. Tutkielmaan on myös kerätty kirjallisuudesta tutkimustuloksia siitä onko ketterien menetelmien laadunvarmistus riittävää, sekä saavutetaanko ohjelmistokehityksessä parempaa laatua käyttämällä ketteriä menetelmiä.

Tutkielman tarkoituksena oli selvittää millä keinoin ketterissä menetelmissä voidaan varmistua ohjelmiston laadusta. Tutkielmassa esitetyt kahdeksan laadunvarmistuskäytännettä, sekä tutkielman ulkopuolelle jätetyt laadunvarmistuksellisia ominaisuuksia sisältävät ketterien menetelmien käytänteet auttavat ohjelmistokehittäjiä luomaan laadukasta ohjelmistoa. Kun tarkastellaan tutkielmaan koostettua taulukkoa(Liite 1) laadunvarmistuskäytänteiden hyödyistä, voidaan todeta, että ketterien menetelmien laadunvarmistuskäytänteet antavat hyvän pohjan laadukkaan ohjelmiston luomiseen.

Tutkielman tarkoituksena oli myös tarkastella ketterien menetelmien laadunvarmistuksen riittävyyttä. Kirjallisuudessa esiintyvien tutkimustulosten perusteella voidaan todeta, että laadunvarmistus on ketterissä menetelmissä riittävää ja parempaa kuin perinteisissä menetelmissä. Kun tarkastelee tutkielmaan koostettua taulukkoa(Liite 2) voi todeta, että ainakin ohjelmiston toiminnallisuuteen liittyviin haasteisiin voidaan vastata ketterien menetelmien laadunvarmistuskäytänteillä, sillä jokainen kahdeksasta ketterien menetelmien laadunvarmistuskäytänteestä auttaa ohjelman toiminnallisuuden kehittämisessä.

Tutkielman tuloksia voidaan käyttää, kun yritys ottaa ketteriä menetelmiä käyttöön ja tarkastelee kuinka laadusta voidaan varmistua uudessa kehitysmenetelmässä. Tässä tutkielmassa tarkasteltiin laadunvarmistusta ainoastaan tuotteen laadun näkökulmasta. Prosessin laadun näkökulma on tutkielmassa rajattu pois, joten jatkotutkimusaiheena voisikin olla se, kuinka ketterissä menetelmissä voidaan varmistua prosessin laadusta, sekä sen jatkuvasta kehittämisestä.

LÄHTEET

Abbas, N. (2009). *Software Quality and Governance in Agile Software Development,*

Ambler, S. (2005). Quality in an agile world. *Software Quality Professional, 7*(4), 34.

Beck, K. (1999). Embracing change with extreme programming. *Computer, 32*(10), 70-77.

Beck, K., & Andres, C. (2004). *Extreme programming explained: Embrace change* Addison-Wesley Professional.

Boehm, B., & Turner, R. (2003). *Balancing agility and discipline: A guide for the perplexed* Addison-Wesley Professional.

Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2004). How extreme does extreme programming have to be? adapting XP practices to large-scale projects. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on,* pp. 10 pp.

Chow, T. S. (1985). Software quality assurance; a practical approach. *IEEE Computer Society Press Tutorial, Silver Spring: IEEE Computer Society Press, 1985, 1*

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in Computers, 62*, 1-66.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development, 9*(8), 28-35.

Galin, D. (2004). *Software quality assurance: From theory to implementation* Pearson education.

Gillies, A. (2011). *Software quality: Theory and management* Lulu. com.

Heidenberg, J. (2011). *Towards increased productivity and quality in software development using agile, lean and collaborative approaches*. Turku: Turku Centre for Computer Science.

Highsmith, J. (2009). *Agile project management: Creating innovative products* Pearson Education.

Hoda, R., Noble, J., & Marshall, S. (2010). Organizing self-organizing teams. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1,* pp. 285-294.

Hossain, A., Kashem, M. A., & Sultana, S. Enhancing software quality using agile techniques.

Huo, M., Verner, J., Zhu, L., & Babar, M. A. (2004). Software quality and agile methods. *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International,* pp. 520-525.

International Standards Organization. (2001). *ISO 9126: Software engineering – product quality. technical report, international standards organization*

Juran, J., & Gryna, F. Juran's quality control handbook, 1988.

Kan, S. H. (2002). *Metrics and models in software quality engineering* Addison-Wesley Longman Publishing Co., Inc.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. a brief history. *Computer, 36*(6), 47-56.

Macias, F., Holcombe, M., & Gheorghe, M. (2003). A formal experiment comparing extreme programming with traditional software construction. *Computer Science, 2003. ENC 2003. Proceedings of the Fourth Mexican International Conference on,* pp. 73-80.

Mahnic, V., & Zabkar, N. (2008). Measurement repository for scrum-based software development process. *Proc. of the 2nd WSEAS International Conference on Computer Engineering and Applications (CEA'08),* pp. 23-28.

Mark, S. (2007). Test-driven development: An agile practice to ensure quality is built from the beginning. *Ioannis G.Stamelos & Panagiotis Sfetsos: Agile Software Quality Assurance,*

Mnkandla, E., & Dwolatzky, B. (2007). Agile software methods: State-of-the-art. *Agile Software Development Quality Assurance,* , 1.

Monochristou, V., & Vlachopoulou, M. (2007). Requirements specification using user stories. *Agile Software Development Quality Assurance,* , 71.

Owens, D. M., & Khazanchi, D. (2009). Software quality assurance. *Handbook of Research on Technology Project Management, Planning, and Operations.ISBN, 1965131010*

Radatz, J., Geraci, A., & Katki, F. (1990). IEEE standard glossary of software engineering terminology. *IEEE Std, 610121990*, 121990.

Rech, J. (2007). Handling of software quality defects in agile software development. *Agile Software Development Quality Assurance, 90*

Rising, L., & Janoff, N. S. (2000). The scrum software development process for small teams. *IEEE Software, 17*(4), 26-32.

Stamelos, I. G., & Sfetsos, P. (2007). *Agile software development quality assurance* Igi Global.

LIITE 1 KETTERIEN MENETELMIEN KÄYTÄNTEET JA VAIKUTUKSET OHJELMISTOKEHITYKSEEN

Käytänne:	**Hyöty:**
Suunnittelupelit	• Auttavat kehittäjien ja asiakkaan välisessä kommunikoinnissa (Cohen, 2004) • Auttavat projektijohtoa projektin sekä julkaisujen suunnittelussa (Cohen, 2004) • Vähentävät virheellistä vaatimusmäärittelyä (Galin, 2004)
Refaktorointi	• Vähentää loogisia suunnitteluvirheitä (Galin, 2004) • Vähentää ohjelmointivirheitä (Galin, 2004)(Huo ym.,2004) • Auttaa tekemään ohjelmakoodista selvempää, yksinkertaisempaa sekä hyvin suunniteltua (Hossain ym., 2013) • Tekee ohjelman kehityksestä tehokkaampaa ja ohjelmistosta luotettavamman sekä testattavamman (Hossain ym., 2013) • Tekee ohjelmistosta uudelleenkäytettävämmän (Cao ym., 2004) • Tekee ohjelmiston ylläpidosta helpompaa, koska ohjelmakoodi on selkeämpää (Cohen, 2004)
Testilähtöinen kehittäminen	• Vähentää testiprosessin vajaavaisuutta (Galin, 2004) • Auttaa ohjelmoijia ymmärtämään mitä ohjelman tulee tehdä ennen ohjelmoinnin alkua (Cohen, 2004) • Tekee ylläpidosta helpompaa, sillä ohjelmistossa tulisi olla testilähtöisen kehittämisen ansiosta vähemmän virheitä (Cohen, 2004) • Antaa projektipäällikölle mahdollisuuden esitellä ohjelmistoa asiakkaalle missä vaiheessa kehitystä tahansa, sillä ohjelmiston tulee olla koko ajan toimivaa ja testattua (Cohen, 2004) • Vähentää ohjelmistossa olevia virheitä (Cohen, 2004) • Kehittäjät voivat olla varmempia siitä, että heidän kehittämänsä ohjelmisto vastaa vaatimuksia, eikä vaadi uudelleen kehitystä (Mark, 2007)
Hyväksymistestaus	• Auttaa löytämään virheet, jotka on aiemmin jäänyt huomaamatta (Hossain ym., 2013) • Hyväksymistestauksessa voidaan varmistua, että iteraatiolle asetetut tavoitteet ovat täyttyneet (Hossain ym., 2013)

Jatkuva integraatio	• Dokumentointi- ja ohjelmointiohjeiden noudattamatta jättäminen vähenee (Galin, 2004) • Vähentää aikaa, joka kuluu virheiden etsimiseen (Huo ym.,2004) • Pakottaa kehittäjiä testaamaan ohjelmaa aikaisessa vaiheessa (Cohen, 2004) • Antaa projektijohdolle mahdollisuuden tarkistaa projektin tila ja eteneminen, missä vaiheessa projektia tahansa (Cohen, 2004)
Pariohjelmointi	• Dokumentointi- ja ohjelmointiohjeiden noudattamatta jättäminen vähenee (Galin, 2004) • Parantaa ohjelmistosuunnittelun ja koodin laatua (Huo ym.,2004),(Cao ym., 2004; Hossain ym., 2013) • Vähentää virheitä ja kehitykseen kuluvaa aikaa(Huo ym.,2004; Cao ym., 2004) • Vähentää ohjelmoijien koulutukseen tarvittavaa aikaa, sillä ohjelmoijat oppivat toisiltaan (Cao ym., 2004) • Ylläpito helpottuu, koska ohjelmakoodista tulee selkämpää (Cohen, 2004) • Kehittäjien välinen kommunikointi paranee (Cohen, 2004)
Asiakkaan tiivis osallistuminen koko projektin aikana	• Kehitystiimi saa arvokasta informaatiota jo kehityksen aikaisessa vaiheessa (Huo ym.,2004) • Asiakkaan jatkuva osallistuminen kehitystyöhön auttaa jalostamaan ohjelmiston vaatimuksia. (Hossain ym., 2013) • Ihannetilanteessa asiakas on osa kehitystiimiä, jolloin asiakas voi vastata kehittäjien kysymyksiin, ratkaista erimielisyyksiä ja priorisoida tehtäviä. (Hossain ym., 2013)
Metaforan käyttö	• Vähentää käyttöliittymän virheitä (Galin, 2004) • Auttaa kehittäjiä jalostamaan vaatimuksia projektin aikana (Hossain ym., 2013; Huo ym., 2004),(Cohen, 2004) • Projektin johdon on helpompi kommunikoida asiakkaan kanssa (Cohen, 2004; Galin, 2004) • Vähentää kehittäjien tahallista poikkeamista asiakkaan vaatimuksista (Galin, 2004) • Yksinkertaistaa keskustelua ohjelmistosta asiakkaan ja kehittäjän, koska kieli ei ole teknistä (Mkandla & Dwolatzki, 2007)

LIITE 2 LAADUNVARMISTUKÄYTÄNTEIDEN VAIKUTUS ISO 9126 MUKAISIIN LAATUKRITEEREIHIN

ISO 9126 Laatukriteerit

	Toiminnallisuus	Luotettavuus	Tehokkuus	Käytettävyys	Ylläpidettävyys	Siirrettävyys
Suunnittelupelit	x					
Iteraatiosuunnittelu	x					
Refaktorointi	x	x	x		x	
Testilähtöinen kehittäminen	x				x	
Hyväksymistestaus	x	x				x
Jatkuva integraatio	x	x	x	x	x	
Pariohjelmointi	x		x	x	x	x
Asiakkaan tiivis osallistuminen koko projektin aikana	x			x		
Metaforan käyttö	x	x	x	x	x	