TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietotekniikan tutkinto-ohjelma

HYVÄKSYMISTESTIVETOINEN OHJELMISTOKEHITYS

Hyödyt ja haasteet empiirisen tutkimuksen valossa

Kandidaatintyö

Ville Harvala

Tietotekniikan laitos
Espoo 2009

TEKNILLINEN KORKEAKOULU
Tietotekniikan osasto
Tietotekniikan tutkinto-ohjelma

KANDIDAATINTYÖN
TIIVISTELMÄ

Tekijä: Ville Harvala

Työn nimi:
Hyväksymistestivetoinen ohjelmistokehitys:
Hyödyt ja haasteet empiirisen tutkimuksen valossa

Päiväys: 3. joulukuuta 2009 **Sivumäärä:** 12 + 28
Pääaine: Ohjelmistotuotanto **Koodi:**
Vastuuopettaja: prof. Lauri Savioja
Työn ohjaaja: Di Juha Itkonen

Hyväksymistestivetoinen ohjelmistokehitys on suunnittelukäytäntö, jossa asiakas otetaan vahvemmin mukaan projektin suunnitteluun. Asiakas luo vaatimusten pohjalta hyväksymistestejä, joiden avulla voidaan testata järjestelmää suoraan vaatimuksia vastaan heti projektin alusta alkaen.

Tässä työssä tutkittiin kirjallisuudessa esitettyjä väitteitä hyväksymistestivetoisesta ohjelmistokehityksestä. Näiden väitteiden pohjalta lähdettiin tutkimaan empiirisiä tutkimuksia seuraavista näkökulmista: Vaatimusmäärittely, kommunikaatio, järjestelmän laatu, dokumentointi, työmäärä ja menetelmän opittavuus. Näiden näkökulmien valossa pyrittiin löytämään johtopäätöksiä menetelmän hyödyllisyydestä. Tutkimuksessa myös kerättiin asiakkaan, testaajien ja ohjelmistokehittäjien kokemuksia menetelmän käytöstä.

Tutkimustulokset osoittivat, että menetelmällä on positiivisia vaikutuksia vaatimusmäärittelyyn ja kommunikaatioon. Menetelmä käyttö on koettu mielekkääksi eikä sen oppiminen ole ollut hankalaa. Kaikkia väitteitä ei ole kuitenkaan tutkittu riittävän kattavasti esimerkiksi asiakkaan roolia menetelmässä. Todellisten yrityskokeiden puuttumisen vuoksi menetelmästä hyödyistä ei vielä voida tehdä kattavia johtopäätöksiä. Työn tuloksena syntyi luettelo kirjallisuudessa esitetyistä väitteistä sekä empiirisistä tutkimuksista havaittuja tuloksia näihin väittämiin.

Avainsanat: ATDD, Hyväksymistestivetoinen ohjelmistokehitys, Testaus, Empiiriset tutkimukset
Kieli: Suomi

HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Computer Science and Engineering
Degree Program of Computer Science and Engineering

ABSTRACT OF BACHELOR'S THESIS

Author: Ville Harvala

Title of thesis:
Acceptance Test Driven Development:
Benefits and challenges in empirical studies

Date: December 3 2009 **Pages:** 12 + 28
Professorship: Ohjelmistotuotanto **Code:**
Supervisor: Professor Lauri Savioja
Instructor: Juha Itkonen, M. Sc. (Tech)

Acceptance Test Driven Development (ATDD) is a software development technique where customer is more involved in the project planning. Customer creates acceptance tests based on requirements before the implementation of a feature. These tests are then used right from the beginning of the software development to see that the system meets the requirements.

This study represented the suggested benefits and challenges of ATDD from these point of views: Defining requirements, communication, quality of the system, documenting, work load and the learnability of the method. From these point of views, empirical studies were investigated and analyzed. The target was to find evidence of the positive effects of the method. Also customer, tester and software developer experiences were gathered from the studies.

Results show that ATDD has positive effects on defining requirements and communication. Different stakeholders have reacted in very positive way towards the method. Method was considered easy to learn. Results indicate also that all point of views haven't yet been studied enough like the customer point of view. Any big conclusions cannot be made, because there is no real case studies with real external customers. The results of this study show a list of suggested benefits and challenges, and also the results of the empirical studies found.

Keywords: ATDD, Acceptance Test Driven Development, Testing, Empirical Studies
Language: Finnish

Sisältö

Luku 1

Johdanto

Hyväksymistestivetoinen ohjelmistokehitys (engl. Acceptance Test Driven Development, ATDD) on ohjelmistokehityksen suunnittelukäytäntö, jossa pyritään sitomaan asiakas vahvemmin mukaan kehitysprosessiin. ATDD:n ajatuksena on, että järjestelmän toteutettavalle ominaisuudelle luodaan niin sanottuja hyväksymistestejä ennen kuin aloitetaan varsinainen ominaisuuden toteuttaminen. ATDD:n esitetyt edut tulevat siitä, että asiakas otetaan mukaan suunnittelemaan hyväksymistestejä ja asiakas on itse vastuussa suurimmilta osin hyväksymistestien luonnista. ATDD:tä on käytetty etenkin ketterien ohjelmistokehitysmenetelmien yhteydessä, kuten esimerkiksi eXtreme Programming (XP) -menetelmässä.

Testivetoisen ohjelmistokehityksen (engl. Test-Driven Development, TDD) kehittäjänä tunnettu Kent Beck esitti ensimmäisenä ajatuksen hyväksymistestivetoisesta ohjelmistokehityksestä. Hän näki, että TDD ei tuo ratkaisua erääseen suureen ongelmaan ohjelmistokehitysprosessissa: Kun kehittäjät luovat itse testinsä ja pyrkivät arvioimaan mitä käyttäjät haluavat, on olemassa suuri riski, että lopputulos ei vastaa käyttäjien tarpeita. (Beck, 2002, sivut 137-138)

Hyväksymistestivetoisen ohjelmistokehityksen ideologia pohjautuukin Beckin kehittämään TDD:hen. ATDD:n ero TDD:hen on se, että yksikkötestien (engl. Unit test) sijaan järjestelmälle toteutetaan testit hyväksymistestitasolla asiakkaan vaatimusten pohjalta. TDD:ssä yksikkötestejä luodaan järjestelmään testaamaan hyvin matalan tason toteutuksia, kun taas ATDD:n testit integroidaan korkeammalle tasolle järjestelmässä: korkeimman sovelluslogiikan (engl. Business Logic) ja käyttöliittymän väliin, tai suoraan käyttöliittymään (Haugset ja Hanssen, 2008). Tällä tavoin päästään suoraan testaamaan vaatimuksissa määriteltyjä toiminnallisuuksia.

Perusajatuksena ATDD:ssä on se, että asiakas luo testit yksin tai vahvassa yhteistyössä ohjelmistokehittäjien kanssa. On esitetty, että tämä toimintatapa tuo

useita hyötyjä esimerkiksi paremmin määritellyt vaatimukset (Mugridge, 2008), parantuneen ohjelmiston laadun (Melnik et al., 2006) sekä parantuneen kommunikaation asiakkaan ja ohjelmistokehittäjien välillä (Park ja Maurer, 2008). Esitettyjä haasteita ovat muun muassa menetelmän oppimisvaikeus ja lisääntynyt työmäärä perinteisiin menetelmiin verrattuna (Melnik et al., 2006).

Eräs hyväksymistestivetoisen ohjelmistokehityksen haasteista on se, että asiakas on vastuussa testien määrittelyistä. Asiakkaalla ei yleensä ole samalla tavalla (tai lainkaan) ohjelmointikokemusta kuin ohjelmistokehittäjillä, joten haasteena on löytää helppo tapa tai väline, jolla asiakas pystyy tehokkaasti testejä luomaan. Ratkaisuna ongelmaan on kehitetty erityyppisiä työkaluja, kuten FIT Framework, Selenium ja EasyAccept. Esimerkiksi Fit-työkalun avulla asiakas pystyy luomaan hyväksymistestejä HTML-taulukoiden muodossa, joihin asiakas määrittelee kyseisen testin syötearvot ja oikeat tulokset. Ohjelmistokehittäjien vastuulla on mahdollistaa se, että käytössä oleva työkalu pystyy keskustelemaan järjestelmän kanssa ja tällä tavoin testaamaan sitä.

Hyväksymistestivetoista ohjelmistokehitystä on kutsuttu kirjallisuudessa usealla eri nimityksellä esimerkkinä Story-Test Driven Development ja Executable Acceptance Test Driven Development. Poikkeavista nimityksistä huolimatta kyseisien suunnittelukäytäntöjen perusajatukset ovat pääosin samanlaisia. Toisessa luvussa kerron hieman hyväksymistestivetoisen ohjelmistokehityksen taustoista ja teoriasta sekä kerron näiden poikkeavien nimitysten mahdollisista erovaisuuksista. Samassa luvussa esittelen lyhyesti yleisimmät työkalut, joilla asiakkaat pystyvät määrittelemään hyväksymistestejä.

Tämän työn tarkoituksena on käydä läpi empiirisiä tutkimuksia ATDD:n käytöstä. Tutkimus keskittyy esitettyjen hyötyjen ja haasteiden käsittelyyn eikä niinkään ATDD:hen liittyvien työkalujen käsittelyyn. Tavoitteena on kerätä kokemuksia menetelmän hyödyistä ja haasteista, ja niiden pohjalta tehdä päätelmiä ATDD:n hyödyllisyydestä. Kirjallisuudessa on esitetty ATDD:n tuovan hyötyjä ja haasteita useasta eri näkökulmasta: Vaatimusmäärittely, kommunikaatio, järjestelmän laatu, dokumentaatio, työmäärä ja opittavuus (Mugridge, 2008; Melnik et al., 2006). Luvussa kolme tarkastelen empiirisiä tutkimuksia näiden näkökulmien valossa ja pyrin löytämään tuloksia esitetyille väitteille.

Empiirisiä tutkimuksia etsittiin tunnetuimmista alan tietokannoista: Google Scholar, ACM Digital Library, ScienceDirect ja Scopus. Hakusanoina käytettiin luvussa kaksi mainittuja ATDD:n eri synonyymejä. Lisäksi lähteitä etsittiin myös löydettyjen tutkimusten viittauksista.

Asiakas on hyvin keskeisessä roolissa ATDD:ssä, joten asiakkaan kokemusten kerääminen on myös tämän tutkimuksen tavoitteena. Luvussa kolme kerron asiakkaan kokemusten lisäksi myös testaajien ja ohjelmistokehittäjien kokemuksista

ATDD:n käytöstä. Neljännessä luvussa tehdään tuloksien perusteella johtopäätöksiä esitettyjen väitteiden paikkaansa pitävyydestä. Johtopäätösten yhteydessä käydään myös pohdiskelua ATTD:n osa-alueista, jotka vaativat lisätutkimusta. Kirjallisuudessa esitetyt väitteet ja empiirisistä tutkimuksista havaitut tulokset kootaan lopuksi samaan taulukkoon, joka löytyy liitteenä tutkimuksen lopussa.

Luku 2

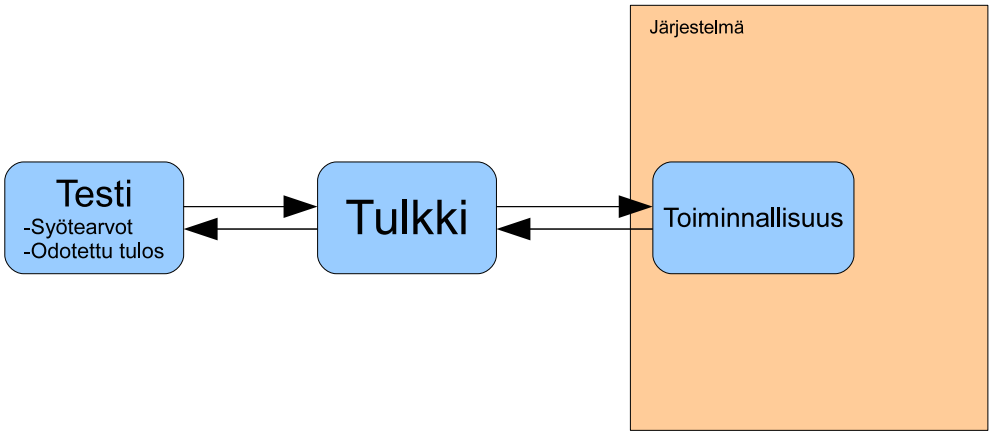
Hyväksymistestivetoinen ohjelmistokehitys

Hyväksymistestivetoinen ohjelmistokehitys on suunnittelukäytäntönä melko tuore. Se on kehitetty 2000-luvun alkupuolella ja menetelmää on kutsuttu usealla eri nimityksellä. Tässä luvussa esitellään hieman näitä eri nimityksiä ja niiden eroavaisuuksia. Menetelmän ydinprosessi esitellään vaihe vaiheelta ja eri roolien tehtävät näissä vaiheissa. Luvun lopussa esitellään lyhyesti erityyppisiä työkaluja, joita havaittiin käytettävän erilaisissa tutkimuksissa ja projekteissa.

2.1 Taustaa

Vaatimusmäärittely on eräs kriittisimmistä osa-alueista ohjelmistokehityksessä ja se muodostaa vahvan pohjan ohjelmistoprojektille. Vaatimusmäärittely vastaa kysymykseen: mitä lähdetään tekemään? Perinteisissä ohjelmistokehitysmenetelmissä järjestelmää testataan vaatimusten näkökulmasta vasta ohjelmistoprojektin viimeisessä vaiheessa. Projektin lopussa löydetyt ohjelmistovirheet ovat huomattavasti kalliimpia korjata, kuin projektin alkuvaiheessa korjatut. Vaikka ohjelmistovirheitä ei löytyisikään, lopullinen järjestelmä ei usein kuitenkaan vastaa asiakkaan toiveita. (Reppert, 2004)

Beck tunnisti tämän ongelman ja esitteli ajatuksen ATDD:stä nimellä Application Test Driven Development. Jos voisimme luoda testejä järjestelmään vaatimusten tasolla, silloin asiakas itse voisi luoda testit järjestelmälle ja määritellä millä tavalla järjestelmän tulisi toimia (Beck, 2002, sivut 137-138). ATDD:ssä hyväksymistestit toimivat sopimuksena asiakkaan ja ohjelmistokehittäjien välillä järjestelmän oikeanlaisesta käyttäytymisestä (Reppert, 2004). ATDD on kuiten-



Kuva 2.1: Hyväksymistestit keskustelevat järjestelmän kanssa tulkin avulla.

kin suunnittelukäytäntö, eikä niinkään testausmenetelmä. Menetelmä kannustaa tarkkaan ennalta suunnitteluun ja pakottaa käsittelemään ominaisuuksien käyttäytymistä hyvinkin tarkasti jo heti ohjelmistokehityksen elinkaaren alkuvaiheessa.

2.2 Vaiheet

ATDD:ssä suunnittelu alkaa siitä, että asiakas määrittelee järjestelmän haluttuja ominaisuuksia eli vaatimuksia. Vaatimusten muotoa ei ole määritelty millään tavalla, vaan ne voivat olla esimerkiksi käyttötapausdokumentteja (engl. Use Case) tai epäformaalia vapaata kuvausta tekstimuodossa. Usein asiakas esittää vaatimukset ohjelmistokehittäjille ja testaajille palavereissa, joissa kehittäjät ja testaajat voivat tarkentaa vaatimusten määrittelyitä esittämällä kysymyksiä (Reppert, 2004). Jokaista vaatimusta kohden asiakas, itse tai tiiviissä yhteistyössä testaajien kanssa, luo vähintään yhden hyväksymistestin. Käytännössä hyväksymistestejä luodaan useita selkeyttämään ominaisuuden toiminnallisuutta, sekä testaamaan myös erikoistilanteita.

Hyväksymistestit ovat kuvauksia, jotka toimivat esimerkkitapauksina millä tavalla ominaisuuden tulisi toimia testin määrittelemässä tilassa ja annetuilla syötteillä. Hyväksymistesti sisältää siis tiedon esimerkkitilanteen syötearvoista ja halutusta lopputuloksesta. Näiden testien tehtävänä on välittää tietoa järjestel-

män halutusta toiminnallisuudesta ohjelmistokehittäjille ja selkeyttää vaatimuksia (Mugridge, 2008). Näiden kuvauksien avulla ohjelmistokehittäjät näkevät konkreettisesti, millä tavalla järjestelmän tulisi toimia testin mukaisella syötearvoilla.

Ohjelmistokehittäjien tehtävänä on aluksi luoda tulkki järjestelmän ja asiakkaan luomien testien välille, jotta asiakkaan luomat hyväksymistestit pystyvät keskustelemaan järjestelmän kanssa ja testaamaan kehitettäviä toiminnallisuuksia (Kuva 2.1). Tulkki saa hyväksymistestiltä esimerkkitilanteen syötearvot ja kutsuu järjestelmän osia, jotka ohjelmistokehittäjä on tulkkiin määritellyt. Lopulta tulkki palauttaa lopputuloksen ja voidaan verrata sitä haluttuun lopputulokseen. Tulkin toteuttamisen jälkeen ollaan tilanteessa, jossa testit keskustelevat järjestelmän kanssa, mutta järjestelmä ei vielä läpäise testejä, koska itse toteutus on tekemättä. Nyt kehittäjä pystyy aloittamaan varsinaisen toteuttamisen ja voi jatkuvasti testata koodiaan hyväksymistestien avulla asiakkaan määrittelemiä vaatimuksia vastaan.

Kaikkien vaatimusten kaikkia hyväksymistestejä ei tehdä kerralla, vaan esimerkiksi XP:n yhteydessä luodaan vaan tulevan iteraation vaatimuksista hyväksymistestejä. Kehittäjän ei tarvitse myöskään odottaa kaikkien vaatimuksen hyväksymistestien toteutusta, vaan voi aloittaa alustavan toteutuksen jo parin hyväksymistestin pohjalta (Reppert, 2004). ATDD kannustaa ja mahdollistaa jatkuvan keskustelun asiakkaan ja testaajan välillä hyväksymistestien kautta (Melnik et al., 2006).

2.3 Ominaispiirteet

Kirjallisuudessa ATDD:tä on kutsuttu useilla eri nimityksillä ja menetelmää ei aina sovelleta täysin samalla tavalla. Seuraavanlaisiin termeihin törmäsin tutkimuksen aikana:

- Acceptance Test Driven Development, ATDD (Rantanen, 2007; Sauvé et al., 2006; Andersson et al., 2003)
- Acceptance Testing (Ricca et al., 2009; Crispin et al., 2001; Watt ja Leigh-Fellows, 2004)
- Application Test Driven Development, ATDD (Beck, 2002)
- Automated Acceptance Testing, AAT (Haugset ja Hanssen, 2008)
- Customer Test Driven Development, CTDD (Crispin, 2005)

- Executable Acceptance Test Driven Development, EATDD (Park ja Maurer, 2008; Melnik ja Maurer, 2007)
- Executable Acceptance Testing (Read et al., 2005; Melnik ja Maurer, 2005)
- Story-test Driven Development, STDD (Mugridge, 2008; Reppert, 2004)

Yllä mainittuja termeillä tarkoitetaan lähes aina samaa asiaa tässä kontekstissa, mutta poikkeuksia myös löytyy. Hyväksymistestivetoinen ohjelmistokehitys on vielä varsin tuore idealtaan, joten siihen kuuluvat käytännöt eivät ole vielä kiveen kirjoitettu.

Taulukko 2.1: Hyväksymistestivetoisen ohjelmistokehityksen ominaispiirteitä.

1. Asiakas määrittelee järjestelmän vaatimukset.	Asiakas määrittelee ja esittelee alustavat vaatimukset tiimille esimerkiksi palaverissa. Palaverissa heränneiden kysymysten avulla vaatimukset tarkentuvat.(Reppert, 2004)
2. Asiakas luo hyväksymistestejä yksin tai yhteistyössä tiimin kanssa.	Asiakas voi luoda testejä yksin, jos siihen kykenee, mutta usein testaaja tai QA (engl. Quality Assurance) vastaava auttaa asiakasta testien luomisessa.(Reppert, 2004)
3. Kehittäjä toteuttaa tulkin järjestelmän ja testin välille.	Ohjelmistokehittäjä määrittelee tulkkiin ne järjestelmän osat, joita hyväksymistestit kutsuvat.(Reppert, 2004)
4. Tulkin määrittelyn jälkeen kehittäjä aloittaa varsinaisen toteutuksen tekemisen.	ATDD:ssa ohjelmistokehittäjä aloittaa varsinaisen toiminnallisuuden toteuttamisen vasta, kun hyväksymistestit ovat määritelty ja yhteydessä järjestelmään. (Reppert, 2004)
5. Ajettavia hyväksymistestejä voidaan hyödyntää laadunvarmistuksessa.	Hyväksymistestejä voidaan ajaa manuaalisesti, mutta suuri hyöty saadaan testien automaattisesta ajamisesta. Ajettavien hyväksymistestien avulla voidaan varmistaa, että järjestlmä toimii asiakkaan vaatimusten mukaisesti. (Sauvé et al., 2006).
6. Hyväksymistestit toimivat kehitysprojektin edistymisen mittarina.	Asiakas voi seurata kehitysprojektin edistymistä reaaliaikaisesti. Sitten kun kaikki testit näyttävät vihreää, niin toteutetut toiminnallisuudet ovat valmiina. (Park ja Maurer, 2008)

Taulukossa 2.1 on ilmaistu hyväksymistestivetoisen ohjelmistokehityksen ominaispiirteitä. Usein poikkeuksia menetelmän työtavoissa tehdään taulukon neljännen kohdan kanssa: Hyväksymistestejä ei aina luoda ennen toiminnallisuuden toteuttamista. Acceptance testing ja Automated acceptance testing -termien yhteydessä testejä ei aina toteuteta Test-First -ideologialla. Story-test Driven Development, Customer Test Driven Development ja Acceptance Test Driven Development -termien yhteydessä menetelmä useimmin sisältää nämä kaikki piirteet (Mugridge, 2008; Crispin, 2005; Rantanen, 2007, sivut 22-27). Termeistä ei kuitenkaan suoraan voida päätellä käytettyjä toimintatapoja, koska niitä on sovellettu projektikohtaisesti.

Hyväksymistestit ovat siis ajettavia testejä, joita voidaan helposti automatisoida. Tällä tavoin asiakas voi seurata ohjelmistokehitysprojektin etenemistä lähes reaaliaikaisesti, koska hän näkee mitkä testit järjestelmä on läpäissyt. ATDD:n ideologia pohjautuu TDD:hen, joten myös ATDD:ssä tarinatestit luodaan järjestelmään ennen toiminnallisuuden toteuttamista. ATDD:tä käytetään yleensä ketterien menetelmien yhteydessä, joten kaikkien toiminnallisuuksien testejä ei luoda kerralla vaan usein luodaan testit kyseisen iteraation toiminnallisuuksille. Kirjallisuudessa on usein mainittu ATDD:n soveltuvan myös muihin kehitysmenetelmiin kuin ketteriin menetelmiin (Reppert, 2004), mutta tutkimuksia asiasta ei ole väittämien tukemiseksi.

2.4 Työkaluja

ATDD:n ajatus on nimenomaan se, että asiakas on itse mukana luomassa testejä järjestelmälle. Asiakkaalla ei kuitenkaan yleensä ole ohjelmistokehitykseen tarvittavaa taitoa luodakseen testejä itse. Asiakkaiden tulisi voida luoda tarinatestit helpolla tavalla ilman, että joutuu tekemisiin lähdekoodin kanssa (Ricca et al., 2009). Tätä varten on kehitetty useita erilaisia työkaluja. Eräs suosituimmista on ollut FIT Framework, jonka käyttöä on tutkittu useissa empiirisissä tutkimuksissa (Ricca et al., 2009; Melnik ja Maurer, 2007).

Seuraavaksi esittelen lyhyesti yleisemmin käytettyjä työkaluja ja niiden toimintaperiaatteita. Katsaus ei ole kattava tutkimus markkinoilla tarjolla olevista työkaluista, vaan tarkoitus on lähinnä esitellä pari erityyppisellä logiikalla toimivaa menetelmää. Tämä antaa paremman käsityksen millä tavalla hyväksymistestejä voidaan luoda käytännössä.

Työkalujen eroavaisuudet tulevat lähinnä tavasta, jolla testit kirjoitetaan. Useimmat työkaluista integroituvat järjestelmän sovelluslogiikkakerrokseen (engl. Business Logic) eli ylimpään toiminnallisen logiikan kerroksen, jolloin ne voivat tes-

tata järjestelmän yleisiä toiminnallisuuksia. Poikkeuksen tekee Selenium-työkalu, joka integroituu itse näkymään.

2.4.1 FIT Framework

Payroll.Fixtures.WeeklyCompensation			
StandardHours	HolidayHours	Wage	Pay()
40	0	20	$800
45	0	20	$950
48	8	20	$1360 expected $1040 actual

Kuva 2.2: Esimerkki Fit-taulukosta.

Eräs suosituimmista työkaluista hyväksymistestien luomiseen on FIT Framework. Lähes kaikissa tämän työn tutkimuksissa käytettiin kyseistä työkalua. FIT-työkalun kehitti Ward Cunningham vuonna 2002, jolloin hän toteutti työkalusta Java-version. Nykyisin työkalusta on toteutettu versiot myös muille suosituille ohjelmointikielille. FIT:ssä asiakas luo testit järjestelmää varten HTML-taulukoilla. Taulukossa on määritelty syötearvot ja odotettu tulos sarakkeissa. Esimerkkikuvassa 2.2 syötearvoina ovat StandardHours-, HolidayHours- ja Wage-sarakkeissa olevat luvut. Pay()-sarakkeessa on odotettu tulos ja vihreä väri kertoo, jos testi on läpäisty. (Cunningham)

FIT integroituu sovelluksen toiminnalliseen logiikkakerrokseen ja ohjelmistokehittäjät toteuttavat integroinnin tulkin avulla (FIT:ssä käytetään termiä Fixture). Fixture on luokka, jossa syötearvot ovat muuttujina ja yllä olevassa esimerkkikuvassa pay()-sarake on metodi, jota kutsutaan. Tämän metodin sisään asetetaan kutsut varsinaiseen järjestelmään ja lopulta palautetaan laskettu lopputulos.

FITin käyttöä on tutkittu useissa akateemisissa ja teollisissa tutkimuksissa. Akateemiset tutkimukset ovat osoittaneet, että työkalun oppiminen on helppoa ja FIT-taulukoiden avulla voidaan kommunikoida tehokkaasti vaatimuksia. (Melnik et al., 2006)

2.4.2 Selenium

Selenium on työkalu, jolla voidaan testata www-sivujen sisältöä. Työkalu testaa sivuja siten, että se avaa valitun selaimen (Firefox, Internet Explorer, Safari) ja tekee käyttäjän määrittelemät toiminnot sivustolla. Käyttäjä voi esimerkiksi määritellä työkalun kirjoittamaan www-sivun lomakkeeseen tietyt arvot ja tämän jälkeen tarkistaa tulossivulta, että halutut asiat näkyvät. Testien sisältämiä toimintoja voidaan myös nauhoittaa tekemällä itse vastaavat toiminnot selaimella. (Selenium)

Selenium eräs haittapuoli on se, että se integroituu sovelluksen näkymään. Usein käyttöliittymään tehdään pieniä viilauksia projektin edetessä. Tämä tarkoittaa sitä, että Seleniumin testejä joudutaan näissä tilanteissa muuttamaan, vaikka itse vaatimukset ovat ovat pysyneet samoina. (Haugset ja Hanssen, 2008)

2.4.3 EasyAccept

EasyAccept on tekstipohjainen työkalu hyväksymistestien luomiseen. Se integroituu FIT:in lailla sovelluslogiikkakerrokseen. EasyAcceptissa testit luodaan tekstitiedostoon, johon määritellään lähtöarvot, odotettu lopputulos ja toiminnon nimi mitä kutsutaan. Myös FIT:in lailla testien ja järjestelmän välille pitää toteuttaa tulkki (EasyAcceptissa kutsutaan nimellä Facade), joka yhdistää testeissä määritellyt toiminnon nimet varsinaisen järjestelmän metodeihin. (Sauvé et al., 2006; EasyAccept)

EasyAcceptin käyttöä ovat tutkineet muun muassa Sauvé et al. (2006). He ovat tutkineet itse työkalun käyttöä ATTD:n yhteydessä sekä työkalun käyttöä ATTD:tä opettaessa. Tulokset ovat molemmissa tapauksissa olleet positiivisia.

Luku 3

Hyödyt ja haasteet

Tässä luvussa käydään läpi kirjallisuudessa esitettyjä väitteitä ATDD:n hyödyistä ja haasteista sekä käsitellään empiirisissä tutkimuksissa havaittuja tuloksia. Luvun alussa kerrotaan ensin hieman itse tutkimuksesta: Millä tavalla tutkimuksia haettiin ja mistä näkökulmista niitä analysoitiin. Luvun lopussa käsitellään myös eri sidosryhmien kokemuksia ATDD:n käytöstä ja minkä tyyppisiä heidän tehtävät ovat olleet ATDD:n yhteydessä.

3.1 Tutkimuksen taustaa

ATDD ei ole vielä laajasti omaksuttu suunnittelukäytäntö eikä empiirisiä kokeita löydy kirjallisuudesta suuria määriä. Suurin osa empiirisistä kokeista on suoritettu yhteistyössä yliopistojen kanssa kurssien muodossa tai opiskelijoille suunnatuilla vapaaehtoisilla tutkimuksilla. Yliopistoissa tehdyt tutkimukset ovat yleensä olleet kahdenlaisia: vertailuryhmillä tehtyjä tutkimuksia tai kyselypohjaisia tutkimuksia. Ensimmäisenä mainitussa opiskelijat jaettiin eri ryhmiin ja verrattiin perinteisiä suunnittelukäytäntöjä ATTD:hen antamalla osalle ryhmistä hyväksymistestit käyttöön ja osalle ei. Kyselypohjaisissa tutkimuksissa opiskelijat olivat ensin kurssin aikana tutustuneet hyväksymistestivetoisen ohjelmistokehityksen teoriaan ja soveltaneet sitten sitä kurssin tehtävissä. Kurssin jälkeen opiskelijoita oli haastateltu menetelmän hyödyllisyydestä ja opittavuudesta.

Kokemuksia ATDD:n käytöstä yritysten ohjelmistokehitysprojekteista on olemassa, mutta nekin on toteutettu melko pienissä mittakaavoissa ja ideaalisen kokoisilla kehitysryhmillä (alle 15 henkilöä). Useimmissa näissä projekteissa ei ollut ulkoista asiakasta, vaan projektit toteutettiin yrityksen sisällä, jolloin asiakkaan roolin otti henkilö yrityksen sisältä. Koska ATDD on suunnittelukäytäntönä mel-

ko tuore, empiiristen kokeiden tulosten ja kokemuksien keräämisessä ei harrastettu kovin ankaraa seulontaa. Useat tutkimukset poikkesivat ATTD:n käytännöissä esimerkiksi Haugsetin tutkimuksessa testejä ei luotu aina ennalta ja asiakas ei ollut kovin vahvasti mukana tekemässä itse testejä (Haugset ja Hanssen, 2008). Halusin kuitenkin liittää työhöni kaikki olennaiset tutkimukset, koska kaikki esiin nousseet näkökulmat ja kokemukset ovat tervetulleita. Lopulliseen analyysiin päätyi kuusi opiskelijatutkimusta ja viisi yritysmaailman case-tutkimusta.

Yritysprojekteissa kaikki käyttivät ketteriä ohjelmistokehitysmenetelmiä ATDD:n yhteydessä paitsi yksi, jossa menetelmää ei mainittu. Useimmissa projekteissa kehitettiin webjärjestelmää, mutta poikkeuksia myös oli. Eräässä projektissa kirjoitettiin uudelleen vanha C-kielellä toteutettu järjestelmä. Uusi järjestelmä toteutettiin C++ kielellä käyttäen ATDD:tä suunnittelukäytäntönä (Andersson et al., 2003).

Mugridge (2008) esitti, että hyväksymistestien luominen on haastava tehtävä ja saattaa vaatia liikoja asiakkaalta. Hyväksymistestien luominen vaatii Mugridgen mukaan näkemystä muun muassa testauksen eri osa-alueista: raja-arvo- ja erikoistilannetestauksesta. Tämä ongelma on pyritty huomioimaan osassa yliopistotutkimuksissa huomioimalla hyväksymistestien tekijöiden taustat. Melnik et al. (2006) käyttivät tutkimuksessaan eri tiedoilla ja taidolla omaavia opiskelijoita. Tutkimukseen osallistui tietotekniikan jatko-opiskelijoita, liiketoiminnan jatko-opiskelijoita sekä tietotekniikan opiskelijoita, jotka eivät vielä olleet suorittaneet alempaa tutkintoaan. Kaikissa tutkimuksissa ei kuitenkaan asiakkaan taustoja huomioitu, joten tämä näkökulma on hyvä huomioida.

Tähän työhön otettiin mukaan kaikki kirjallisuudessa esitetyt relevantit positiiviset ja negatiiviset väitteet koskien ATTD:tä. Nämä väitteet koskivat seuraavia ohjelmistokehityksen osa-alueita:

- Vaatimusmäärittely ATDD:n avulla.
- Asiakkaan ja kehitystiimin kommunikaatio.
- Järjestelmän laatu.
- Hyväksymistestien dokumentaatio ja hallinta.
- ATDD:n vaikutukset työmäärään ja menetelmän oppimiseen.

Seuraavaksi esittelen kirjallisuudessa esitettyjä väitteitä näistä näkökulmista. Jokaisen näkökulman kohdalla esittelen esitetyt väitteet taulukon avulla, jossa plusmerkillä(+) on merkitty positiiviset väitteet ja miinusmerkillä(-) negatiiviset. Tämän jälkeen esittelen empiirisissä tutkimuksissa ilmenneitä tuloksia kyseisen nä-

kökulman kannalta. Lopulta kokoan esitetyt väitteet ja empiiristen kokeiden tulokset samaan taulukkoon, joka löytyy tämän työn lopusta liitteessä A.

ATDD:ssä asiakkaan rooli korostuu perinteisiin suunnittelukäytäntöihin verrattuna, mutta menetelmällä on myös vaikutuksia kehittäjien, testaajien ja laadunvalvonnasta vastaavien tehtäviin. Useat empiiriset tutkimukset ovatkin tutkineet ATDD:n kokemuksia ja käytön mielekkyyttä eri roolien näkökulmasta, joten kerron niistä hieman luvun lopussa.

3.2 Vaatimusmäärittely

Taulukko 3.1: Kirjallisuudessa esitettyjä väitteitä vaatimusmäärittelyn näkökulmasta.

Esitettyjä väitteitä	
Väite	
+ Hyväksymistestit helpottavat vaatimusten ymmärtämistä.	Hyväksymistestien avulla eri sidosryhmät ymmärtävät paremmin vaatimuksia proosallisiin vaatimusmäärittelyihin verratuna. Vapaalla tekstillä kuvatut vaatimukset sisältävät usein epätarkkuutta ja täten johtavat virheisiin (Ricca et al., 2009).
+ Hyväksymistestien avulla lopullinen järjestelmä vastaa paremmin asiakkaan vaatimuksia.	ATDD:ssä asiakas on itse määrittelemässä hyväksymistestejä, jolloin järjestelmään toteutetaan juuri asiakkaan haluama toiminnallisuus (Park ja Maurer, 2008). Menetelmä kannustaa jatkuvaan yhteistyöhön ja kommunikaatioon, jonka tuloksena usein asiakas saa mitä haluaa (Crispin, 2005; Mugridge, 2008).
+ Hyväksymistestit ovat riittäviä itsessään toimimaan vaatimusmäärittelydokumentteina.	Projektin aikana kehittyvät ja tarkentuvat hyväksymistestit toimivat hyvänä korvikkeena perinteisille vaatimusdokumenteille (Mugridge, 2008; Melnik et al., 2006).
- Hyväksymistestit eivät sovellu kaikentyyppiseen testaukseen.	Vaatimukset, esimerkiksi koskien tietyn tyyppisiä käytettävyyttä ja ylläpitoa, voivat olla hankalia testata hyväksymistestien avulla (Ricca et al., 2009).

ATDD:n on väitetty parantavan vaatimusmäärittelyä, koska se asettaa asiakkaat itse määrittelemään yksityiskohtaisella tasolla järjestelmän toiminnallisuut-

ta. Hyväksymistestien avulla ohjelmistokehittäjät ymmärtävät paremmin vaatimusten toimintaperiaatteita ja yksityiskohtia. Ketterien ohjelmistokehitysprosessien yhteydessä hyväksymistestit toimivat hyvänä vaihtoehtona perinteisille vaatimusdokumenteille (Mugridge, 2008).

Perinteisesti vaatimusdokumentit ovat olleet vapaassa muodossa ja epäformaalilla tekstillä kuvattuja. Näiden dokumenttien suurimpia ongelmia ovat tulkinnanvaraisuus ja yksityiskohtien puute. Ohjelmistokehittäjät ja -suunnittelijat joutuvat usein tulkitsemaan vapaassa muodossa kuvattuja vaatimuksia ja itse arvailemaan erikoistilanteiden käsittelyjä. Lopputulos on valitettavan usein se, että asiakas ei saa mitä haluaa. (Ricca et al., 2009)

Yliopistoissa tehdyissä tutkimuksissa on tutkittu vaatimusmäärittelyä koskevia väitteitä melko kattavasti. Ricca et al. (2009) tutkivat taulukon 3.1 ensimmäistä väitettä omassa vertailuryhmillä tehdyllä tutkimuksessaan. Tutkimuksessa järjestettiin koe kahdessa eri sijainnissa opiskelijoille (15 henkilöä per sijainti). Toiselle ryhmälle annettiin vain tekstipohjaiset vaatimusdokumentit ja toiselle lisäksi FIT-taulukoilla määritellyt hyväksymistestit. Tämän jälkeen molemmille ryhmille esitettiin samat kysymykset järjestelmän vaatimuksista ja tällä tavalla tutkittiin onko asiakkaan esittämät vaatimusmäärittelyt mahdollistaneet riittävän ymmärryksen halutusta järjestelmästä. Tuloksien mukaan Fit-taulukoiden avulla vaatimuksia ymmärrettiin 95 % oikein, keskimäärin noin neljä kertaa enemmän oikein kuin ilman taulukoita.

Calgaryn yliopistossa ja SAIT:ssä (Southern Alberta Institute of Technology) tehdyssä tutkimuksessa tutkittiin taulukon kolmatta väitettä eli voidaanko hyväksymistesteillä kommunikoida järjestelmän vaatimuksia. Kursseilla toteutettiin WWW-palvelu valmiiden hyväksymistestien perusteella, sekä opiskelijat loivat ryhmissä hyväksymistestejä toisille ryhmille. Tutkimuksen tulokset perustuvat kurssien jälkeiseen kyselyyn opiskelijoilta. Kolmasosa opiskelijoista vastasivat, että hyväksymistestit soveltuvat hyvin tai loistavasti vaatimusten määrittelemiseen. Kukaan ei vastannut, että testit eivät olisi soveltuneet lainkaan. Kun opiskelijoilta kysyttiin, että olisivatko he halunneet mieluummin proosallisia vaatimuskuvauksia hyväksymistestien sijaan. 16 opiskelijaa 20:stä vastasi "ei"(Read et al., 2005; Melnik ja Maurer, 2005). Melnik et al. (2004) tutkivat samoilla kursseilla kuinka hyvin lopullinen järjestelmä vastaa asiakkaan vaatimuksia, kun se toteutetaan hyväksymistestien avulla. 73 % ryhmistä toteuttivat 100 % asiakkaan vaatimuksista (Melnik et al., 2004).

Yritysmaailman projekteissa havaittiin myös myönteisiä kokemuksia hyväksymistesteistä. Norjalaisessa keskikokoisessa yrityksessä tehdyssä projektissa suurin osa kehittäjistä ilmaisi, että hyväksymistestit parantavat järjestelmän ymmärtävyyttä. He eivät kuitenkaan käyttäneet hyväksymistestejä vaatimusdokumentteina,

vaan lähinnä tukevina asiakirjoina (Haugset ja Hanssen, 2008). Eräässä toisessa case-tutkimuksessa käytettiin hyväksymistestejä varsinaisina vaatimusmäärittelydokumentteina ja tulosten perusteella projekti onnistui hyvin. Jokaisen XP:n iteraatioiden alkuvaiheessa pidettiin tapaamisia kaikkien sidosryhmien kanssa ja keskusteltiin toteutettavista ominaisuuksista. Tämän jälkeen asiakas loi hyväksymistestit yhdessä testaajan kanssa. Kehittäjät eivät aina tehneet ominaisuuden toteuttamista Test-First -ideologialla, mutta he saivat hyväksymistestit käyttöön toteuttamisen aikana (Melnik ja Maurer, 2007).

Hyväksymistesteihin pätee kuitenkin samat ongelmat, kuin perinteiseen vaatimusmäärittelyyn. Jos itse testit ovat määritelty laiskasti ja suppeasti, niin ne eivät palvele tarkoitustaan. Opiskelijoilla tehdyissä tutkimuksissa havaittiin, että opiskelijoiden ryhmien kesken vaihdettujen hyväksymistestien laadussa oli eroja ja olivat usein riittämättömiä (Read et al., 2005; Melnik ja Maurer, 2005). Tutkimuksissa nousi myös esiin, että kaikkia ominaisuuksia ei ole tarpeen määritellä hyväksymistesteillä, koska ne ovat helposti ymmärrettävissä ilman niitäkin (Ricca et al., 2009).

Yhteenvetona ATDD:n vaikutus vaatimusmäärittelyyn on empiiristen kokeiden perusteella positiivista. Hyväksymistestien parantavista vaikutuksista vaatimusten ymmärtämiseen on havaittu tuloksia sekä yliopistoissa, että yrityksissä tehdyissä tutkimuksissa. Viitteitä löytyi myös siitä, että hyväksymistestien avulla lopullinen järjestelmä vastaa paremmin vaatimuksia, mutta tätä väitettä ei ole tutkittu vertaisprojektien avulla muualla kuin opiskelijoilla. Yritysmaailman projekteissa voidaan kuitenkin paljon päätellä siitä, että useimmat onnistuivat hyvin ja asiakas oli tyytyväinen lopputulokseen. Asia vaatii kuitenkin lisätutkimusta, koska useimmissa yritysprojekteissa ei ollut aitoa ulkoista asiakasta.

Yliopistotutkimusten tuloksissa ja yritysten kokemuksista voidaan myös päätellä, että hyväksymistestit voivat myös toimia korvaavana dokumentaationa perinteisille vaatimusdokumenteille, varsinkin ketterien menetelmien yhteydessä. Kokemuksia ei kuitenkaan ole kuin ketteristä menetelmistä ja pienen skaalan yliopistollisista tutkimuksista, joten lisätutkimusta tarvitaan hyväksymistestien soveltuvuudesta vaatimusdokumenteiksi muunlaisissa kehitysmenetelmissä. Tutkimuksissa havaittu mitään tuloksia tai havaintoja siitä, millä tavalla hyväksymistestit soveltuvat erityyppisten vaatimusten testaamiseen.

3.3 Kommunikaatio

Taulukosta 3.2 hyvin nähdään, että ATDD:stä ei ole esitetty kuin positiivisia väitteitä, kun puhutaan menetelmän vaikutuksesta kommunikaatioon. ATDD mene-

Taulukko 3.2: Kirjallisuudessa esitettyjä väitteitä kommunikaation näkökulmasta.

Esitettyjä väitteitä	
Väite	
+ ATDD parantaa järjestelmän vaatimusten kommunikointia ohjelmistoprojektin sidosryhmien välillä	Kaikki sidosryhmät ovat mukana hyväksymistestien ylläpidossa, kehittämisessä ja ajamisessa jossain vaiheessa heidän työtään (Park ja Maurer, 2008).
+ Ohjelmistokehityksen tilanne on tiedossa kehitystiimille ja asiakkaalle eri projektin vaiheissa.	Hyväksymistestejä voidaan käyttää ohjelmistoprojektin edistymisen mittarina. Se kuinka monta testiä järjestelmä läpäisee tiettynä hetkenä, kertoo projektin tilanteen. (Park ja Maurer, 2008; Read et al., 2005)

telmänä pakottaa sidosryhmät keskustelemaan keskenään, koska hyväksymistestien luominen ja suunnittelu vaatii sitä. Projektin elinkaaren aikana hyväksymistestit paranevat ja tarkentuvat. Tilanne yleensä on se, että ohjelmistokehittäjä tai testaaja huomaa erikoistilanteen, jonka käyttäytymistä asiakas ei ole määritellyt. Tällaiset tilanteet johtavat jatkuvaan keskusteluun asiakkaan, testaajan ja kehittäjien välillä. (Mugridge, 2008)

Yliopistotutkimuksissa ei ole tarkasteltu ATDD:n vaikutuksisia sidosryhmien kommunikaatioon. Yritysten case-tutkimuksista voidaan kuitenkin havaita positiivisia merkkejä. Watt ja Leigh-Fellows (2004) tekemässä case-tutkimuksessa yritys havaitsi menetelmän kannustavan parempaan kommunikaatioon. Iteraatioiden lopussa pidetyt "Getting our Stories Straight" -tapaamiset asiakkaan ja laadunvarmistajan kanssa koettiin oikein toimiviksi. Näissä tapaamisissa tarkasteltiin projektin tilannetta ja valmisteltiin seuraavan iteraation hyväksymistestejä. Iteraation alussa esiteltiin nämä hyväksymistestit kehittäjille ja pienissä ryhmissä kehittäjät analysoivat näitä testejä. Tämän jälkeen kehittäjät aloittivat ominaisuuksien implementoinnin. Samanlaisia positiivisia tuloksia havaittiin myös muissa yritysprojekteissa (Melnik ja Maurer, 2007; Andersson et al., 2003; Haugset ja Hanssen, 2008).

Eräässä case-tutkimuksessa havaittiin, että ATDD kannustaa myös ohjelmistokehittäjien välistä kommunikaatiota. Usein oltiin tilanteessa, että kehittäjä joutui muuttamaan toisen kehittäjän luomaa koodia, joka johti hyväksymistestin rikkoutumiseen. Tämä kannusti kehittäjää keskustelemaan muutoksista toisen kehittäjän kanssa. (Haugset ja Hanssen, 2008)

ATTD:tä voidaan myös käyttää kommunikoimaan ohjelmistoprojektin etenemis-

tä, jos hyväksymistestejä luodaan oikealla tavalla eli Test-First -ideologialla. Usein kaikkia hyväksymistestejä ei luoda projektin alussa, vaan pelkästään esimerkiksi tulevassa lyhyessä iteraatiossa toteutettaville ominaisuuksille. Tällöin iteraation etenemistä voidaan seurata reaaliaikaisesti. Empiirisissä tutkimuksissa ei lainkaan mainittu, että hyväksymistestejä oltaisiin hyödynnetty tällä tavalla. Tämä johtuu varmasti osaltaan siitä, että harvoissa projekteissa oli aito ulkoinen asiakas, jolloin projektin seuraamisen merkitys korostuisi entisestään.

ATTD:n positiivisista vaikutuksista sidosryhmien väliseen kommunikaatioon ei voida kiistää. Lähes kaikissa tutkimuksissa on vähintään lyhyesti mainittu menetelmän positiivisesta vaikutuksesta kommunikaatioon ja missään tutkimuksessa tulokset eivät ole osoittaneet, että menetelmä vaikeuttaisi kommunikaatiota.

3.4 Järjestelmän laatu

Taulukko 3.3: Kirjallisuudessa esitettyjä väitteitä järjestelmän laadun näkökulmasta.

Esitettyjä väitteitä	
Väite	
+ ATDD parantaa koodin laatua.	ATDD kannustaa parempaan suunnitteluun ja se synnyttää yksinkertaisempaa koodia.(Reppert, 2004)
+ ATDD mahdollistaa turvallisen ympäristön koodimuutoksille.	Ajettavat hyväksymistestit toimivat hyvänä regressiotesteinä järjestelmälle ja tällä tavalla mahdollistaa järjestelmän korkean laadun ylläpitämisen (Park ja Maurer, 2008; Mugridge, 2008).

Hyväksymistestit integroituvat yleensä sovelluslogiikkakerrokseen ja ne säilyvät melko muuttumattomina. Ne ovat irrallisia järjestelmästä ja itsenäisiä. Hyväksymistestit pysyvät refaktoroinnin aikana muuttumattomina, mikä ei aina pidä paikkaansa esimerkiksi yksikkötestien tapauksessa. Tämän vuoksi ne toimivat hyvänä turvaverkkona, kun järjestelmän koodia refaktoroidaan. (Andersson et al., 2003)

Haugsetin yritysmaailman tutkimuksessa ohjelmistokehittäjät yhtyivät mielipiteeseen, että hyväksymistestit vaikuttavat positiivisesti ohjelmiston laatuun. Testien avulla katetaan suurempi määrä virheitä ja regressiotestaus vähentää työmäärää manuaalisen testaukseen verrattuna. Vapautunut työaika voitiin käyt-

tää muihin laadunvarmistusprosesseihin. ATDD pakottaa suunnittelemaan melko tarkasti toteutettavaa toiminnallisuutta ennalta ja tämä koettiin hyvänä asiana ohjelmistokehittäjien joukossa, koska erikoistapaukset huomioitiin paremmin suunnittelussa. (Haugset ja Hanssen, 2008)

Erään projektin laadunvalvoja kommentoi ATTD:n tuoneen mukanaan puhtaamman koodin. Jos järjestelmästä löytyi ohjelmistovirhe, niin heidän tarvitsi vaan päivittää hyväksymistestien FIT-taulukoita, jonka jälkeen kyseiseen virheeseen oltiin varauduttu. (Melnik ja Maurer, 2007)

Yliopistokurssilla tutkittiin laadukkaiden hyväksymistestien vaikutusta järjestelmän laatuun. Hyväksymistestien laatua arvosteltiin antamalla niille pisteitä siitä, kuinka hyvin ne määrittelevät toteutettavan järjestelmän vaatimuksia. Lähdekoodia laatua mitattiin katselmoimalla sitä kahden henkilön toimesta. Vaikka tutkimus osoitti pientä positiivista korrelaatiota laadukkaiden hyväksymistestien ja laadukkaan koodin välillä, väitettä ei voida tukea tulosten perusteella. (Melnik et al., 2006)

Tutkimusten vähäisestä määrästä huolimatta taulukossa 3.3 listattuihin väitteisiin löytyy positiivisia tuloksia. Viitteitä ATTD:n postiivisista vaikutuksesta koodin laatuun löytyy empiirisissä tutkimuksissa. Kattavien hyväksymistestien avulla virhetilanteet huomioidaan paremmin ja itse koodin suunnittelu paranee myös. Ajettavat hyväksymistestit mahdollistavat regressiotestauksen, jolloin koodin muutokset voidaan suorittaa turvallisemmin.

3.5 Dokumentointi

Taulukko 3.4: Kirjallisuudessa esitettyjä väitteitä dokumentoinnin näkökulmasta.

Esitettyjä väitteitä	
Väite	
- Suuren hyväksymistestimäärän hallinta ja ylläpito on haasteellista.	Suuren testimäärän hallinta ja organisoiminen on haastavaa. Usein on tärkeää löytää helposti suuren datamäärän joukosta relevantit testit (Mugridge, 2008).

Hyväksymistestien dokumentoinnin ja hallinnan haasteita ei ole kirjallisuudessa tutkittu lähes lainkaan, vaikkakin ongelma on huomioitu. Hyväksymistestejä luodaan yleensä suuri määrä per vaatimus, jolloin projektin elinkaareen aikana

lukumäärä kasvaa isoksi. Eräässä yritysprojektin aikana hyväksymistestejä syntyi 7000 kappaletta. Tutkimuksessa ei kuitenkaan ilmaistu, että testien valtava määrä olisi aiheuttanut ongelmia (Melnik ja Maurer, 2007).

Hyväksymistestien hallinta on tärkeää, koska kun järjestelmän vaatimukset muuttuvat, muuttuvat myös testit. Tämän vuoksi niiden selaaminen ja muuttaminen pitää olla helposti tehtävissä. Kyseinen näkökulma tarvitsee lisätutkimusta.

3.6 Työmäärä ja menetelmän opittavuus

Taulukko 3.5: Kirjallisuudessa esitettyjä väitteitä työmäärän ja opittavuuden näkökulmasta.

Esitettyjä väitteitä	
Väite	
- ATDD:n oppiminen on haasteellista.	Menetelmän oppiminen, jotta voidaan tuottaa laadukkaita testejä, on haastavaa. Henkilöiden koulutuksella ja taustoilla on merkitystä (Melnik et al., 2006).
- Hyväksymistestien luominen voi olla liian haastavaa asiakkaalle.	Hyväksymistestien luominen vaatii asiakkaalta myös tietoa testauksesta, kuten raja-arvojen ja erikoistapausten testauksesta. (Mugridge, 2008)
- ATDD lisää työmäärää perinteisiin ohjelmistokehitysmenetelmiin verrattuna.	ATDD lisää työmäärää esimerkiksi TDD:hen verrattuna. (Melnik ja Maurer, 2007)

Useimmat hyväksymistestivetoisen ohjelmistokehityksen haasteista on esitetty koskevan menetelmän opittavuutta ja sen vaatimaa työmäärää perinteisiin menetelmiin verrattuna. Taulukosta 3.5 näemme, että kaikki tämän näkökulman esitetyt väitteet ovat negatiivisia. Tutkimukset eivät kuitenkaan tue näitä kaikki väitteitä.

ATDD:n yhteydessä on ollut kiistelyä siitä, että tulisiko asiakkaan yksin , yhdessä IT ammattilaisen kanssa vai ei lainkaan luoda hyväksymistestejä (Melnik et al., 2006). Mugridge (2008) esittää, että testien luomiseen liittyy myös paljon tietämystä testauksesta yleisesti ja tätä tietoa ei voida vaatia asiakkaalta.

Useimmissa tutkimuksissa asiakas pystyi luomaan hyväksymistestejä myös yksin. Carmen Systemsin ohjelmistokehitysprojektissa asiakkaan roolissa oleva henkilö oli täysin vastuussa hyväksymistestien luomisesta. Projektin tutkimustuloksissa

ei ilmentynyt, että asiakkaalla olisi ollut ongelmia hyväksymistestien luomisessa (Andersson et al., 2003). Tuloksessa täytyy kuitenkin huomioida, että asiakkaan roolissa oli henkilö yrityksen sisältä, joten kyseinen asetelma ei vastaa todellisen ulkoisen asiakkaan tilannetta.

Melnik et al. (2006) pyrkivät omassa Calgaryn yliopistossa tehdyssä tutkimuksessa selvittämään, huomioiden asiakkaan roolissa olevat koulutukselliset taustat, että kykeneekö asiakas luomaan laadukkaita hyväksymistestejä yksin vai tarviiko asiakas rinnalle IT ammattilaisen. Asiakkaan muodosti pienet ryhmät, joihin kuului alemman tutkinnon suorittaneita liiketoiminnan ja tietotekniikan opiskelijoita. Osa ryhmistä sisälsivät vain liiketoiminnan opiskelijoita. Tutkimustuloksissa ei havaittu suuria eroja hyväksymistestien laaduissa ryhmien välillä. Tulokset kuitenkin osoittivat, että molemmat ryhmät pystyivät tuottamaan laadukkaita vaatimusmäärittelyitä hyväksymistestien avulla. Kaikki osanottajat osallistuivat kurssin alussa kolmen tunnin koulutukseen, jossa aiheena oli hyväksymistestit ja FIT-työkalu.

Samassa Calgaryn tutkimuksessa tutkittiin myös hyväksymistestien luomiseen käytettyä työmäärää. Kurssi esitti tuntiarvion ennen kurssin aloittamista, kuinka paljon aikaa ryhmät käyttävät aikaa testien luomiseen. Ainoastaan yksi ryhmä yhdeksästä ylitti tämän arvion muiden jäädessä alle (Melnik et al., 2006). Tästä tuloksesta ei voida kuitenkaan päätellä paljon mitään, koska kyseessä oli vain kurssihenkilökunnan asettama arvioitu tuntimäärä eikä se perustunut mihinkään tiettyyn tilastoon.

Ricca et al. (2009) tekemässä yliopistollisessa tutkimuksessa ei havaittu merkittävää eroa työmäärässä, kun vertaisryhmien avulla tarkasteltiin käytettyä työmäärä vaatimusten ymmärtämiseen. Puolet ryhmistä käyttivät pelkkiä tekstimuotoisia määrittelyitä vaatimusten ymmärtämiseen, kun toisella puolikkaalla oli lisäksi hyväksymistestit käytössä. Vaatimusten hyvä ymmärtäminen hyväksymistestien avulla ei vaatinut suurempaa työmäärää verrattuna tilanteeseen, jossa hyväksymistestejä ei ollut saatavilla.

ATDD-menetelmän oppimisessa ei ole pääosin havaittu ongelmia. Usein lyhyt koulutus projektin alkuvaiheessa on riittänyt menetelmän ideologian ja työkalun opettamiseen. Eräässä yritysprojektissa ATDD-menetelmää koulutettiin aluksi neljän tunnin koulutuksella sekä ulkoinen konsultti toimi neuvonantajana ensimmäisessä iteraatiossa. Heidän mielestään menetelmän ja käytettävään työkalun (FIT) opettelu oli helppoa. Asiakkaan roolissa ollut täysipäiväinen ja paikanpäällä työskentelevä projektipäällikkö loi hyväksymistestit yhdessä testaajan kanssa. Asiakkaan roolissa ollut henkilö kertoi, että itse testien tekeminen ei ollut hankalaa, vaan siihen liittyvä kuri oli vaikea ylläpitää. Testien luominen koettiin hieman epämiellyttäväksi, koska niiden tekeminen vaati tarkkaa suunnittelua. (Melnik ja

Maurer, 2007)

Yhteenvetona tutkimusten perusteella asiakas on kyennyt luomaan hyväksymistestejä yksin. Ongelmana näissä tutkimuksissa on kuitenkin sama aiemmin mainittu asia: Aitoa ulkoisen asiakkaan tilannetta ei ole tutkittu riittävästi. Vaikka osassa yliopistollisissa tutkimuksissa henkilöiden taustat on pyritty huomioimaan, ei tutkittavien henkilöiden taustoissa ole ollut merkittävää eroa.

ATDD-menetelmän oppiminen yleensä onnistui lyhyen koulutuksen avulla ja tämän jälkeen sidosryhmät olivatkin yleensä melko valmiita luomaan hyväksymistestejä. ATDD:n vaikutuksista työmäärään ei ole pystytty tutkimaan riittävän hyvin, ainoastaan kurssien muodossa tapahtuneissa tutkimuksissa. Ja ne eivät vastaa riittävän hyvin todellisia tilanteita.

3.7 Sidosryhmien tehtävät ja kokemukset

Asiakkaan tehtävänä ATDD:ssä on kommunikoida vaatimukset yleisellä tasolla eri sidosryhmille ja vaatimusten pohjalta tehdä hyväksymistestit yksin tai yhteistyössä testaajan kanssa. Myös ohjelmistokehittäjät voivat osallistua testien tekemiseen. Tärkeää kuitenkin on se, että asiakas on itse mukana testien tekemisessä, koska vain tällä tavalla saavutetaan selkeät vaatimusmäärittelyt (Beck, 2002). Vaikka tutkimukset osoittivat viitteitä siitä, että asiakas pystyisi luomaan hyväksymistestejä yksin, niin useissa yritysmaailman tutkimuksissa havaittiin hyviä tuloksia, kun testaaja tai laadunvarmistaja osallistui myös testien luomiseen (Melnik ja Maurer, 2007; Watt ja Leigh-Fellows, 2004). Tuloksissa havaittiinkin myös negatiivisia kokemuksia siitä, että asiakas on itse ollut vastuussa testien tekemisestä. Asiakas joutuu itse suunnittelemaan ja miettimään toteutettavien ominaisuuksien käyttäytymistä melko tarkkaa, joka on usein uuvuttavaa (Andersson et al., 2003).

Testaajien rooli hyväksymistestivetoisissa projekteissa on ollut toimia hyväksymistestien laadunvarmistajina. Testaajien tehtävänä on usein ollut toimia hyväksymistestien tarkastajina ja sidosryhmien välisenä kommunikoijana. Testaajan käy läpi hyväksymistestejä, etsii ongelmia ja ehdottaa tarvittaessa uusia testejä asiakkaalle (Melnik ja Maurer, 2007; Crispin, 2005). Perinteisissä suunnittelukäytännöissä testaajat koetaan usein rasitteena. Laadunvalvojan roolissa ollut henkilö kommentoi ATDD:n nostaneen hänen arvostustaan tiimissä, koska hänen roolinsa on hyvin keskeinen asiakkaan ja kehittäjien kanssa kommunikoidessa (Reppert, 2004).

Tutkimuksissa havaittiin positiivisia kokemuksia myös testaajien puolelta: Hyväksymistestien avulla testaajat havaitsivat puuttuvia ja virheellisiä vaatimuk-

sia (Melnik ja Maurer, 2007). ATDD:n testauspainoitteinen ideologia vähentää tyypillisiä virheitä, kuten esimerkiksi syötevirheitä, joten testaaja voi keskittyä muihin ongelmiin (Crispin, 2005). Negatiivisia kokemuksia ei noussut esiin empiirisissä tutkimuksissa testaajan näkökulmasta.

Ohjelmistokehittäjien tehtävänä ATDD:ssä on toteuttaa järjestelmään halutut toiminnallisuudet hyväksymistestien perusteella. Kehittäjät näkevät testien avulla millä tavalla järjestelmän tulisi käyttäytyä annetuilla syötearvoilla. Ohjelmistokehittäjät ovat myös vastuussa tulkin toteuttamisesta hyväksymistestien ja järjestelmän välille. Eräässä yritysmaailman projektissa ohjelmistokehittäjät olivat myös vastuussa itse hyväksymistestien luonnissa, koska he kokivat tuntevansa järjestelmän osa-alueen parhaiten (Haugset ja Hanssen, 2008).

Empiirisissä tutkimuksissa havaittiin, että ohjelmistokehittäjät pystyvät hyväksymistestien avulla ymmärtämään paremmin järjestelmän vaatimuksia ja FIT-työkalun avulla luodut hyväksymistestit koettiin hyödyllisiksi (Ricca et al., 2009). Samanlaisia tuloksia havaittiin myös muissakin tutkimuksissa (Melnik et al., 2004; Haugset ja Hanssen, 2008). ATDD:n ajettavat hyväksymistesti mahdollistivat regressiotestauksen hyödyntämistä. Tämä vähensi kehittäjien mielestä virheiden määrää ja vähensi manuaalisen testauksen tarvetta (Haugset ja Hanssen, 2008).

Osa kehittäjistä kokivat, että testien tekeminen vaati enemmän työtä kuin esimerkiksi JUnit-testien tekeminen (Melnik ja Maurer, 2007). Kaikenkaikkiaan ohjelmistokehittäjät kuitenkin kokivat menetelmän hyödylliseksi.

Luku 4

Johtopäätökset

Tämän tutkimuksen tavoitteena oli kerätä empiirisistä tutkimuksista tuloksia hyväksymistestivetoisen menetelmän hyödyistä ja haasteista. Katsaus empiirisiin tutkimuksiin osoittaa kuitenkin hyvin vielä sen, että hyväksymistestivetoinen ohjelmistokehitys on hyvin tuore ja maltillisesti omaksuttu suunnittelukäytäntö. Empiiriset tutkimukset ovat suurimmalta osin toteutettu yliopistollisissa ympäristöissä ja teolliset tutkimukset ovat olleet hyvin pienessä skaalassa. Yrityksissä tehdyt projektit ovat useimmat olleet yrityksen sisäisiä ja ne on toteutettu ideaalisen kokoisilla kehitystiimeillä.

Yliopistolliset ja teolliset tutkimukset ovat kuitenkin osoittaneet lupaavia tuloksia siitä, että ATTD parantaa vaatimusmäärittelyä ja vaatimusten ymmärtämistä. Hyväksymistestien avulla ohjelmistokehittäjät ymmärtävät paremmin asiakkaan vaatimusta, koska niiden avulla he näkevät helposti ominaisuuden suunnitellun käyttäytymisen. Tutkimukset myös tukevat väitettä, että ATTD:llä on positiivisia vaikutuksia sidosryhmien kommunikaatioon. Ohjelmistokehitysprojektin aikana muotoutuvat ja jatkuvasti kehittyvät hyväksymistestit kannustavat jatkuvaan keskusteluun eri sidosryhmien välillä.

Tutkimukset ovat myös osoittaneet, että asiakas kykenee luomaan hyväksymistestejä tehokkaasti testaajan tai laadunvarmistajan avustuksella, ja jopa ihan yksinkin. Vaikka joissain tapauksissa asiakkaalla on ollut pieniä ongelmia työmäärän ja menetelmän vaatiman kurin kanssa, niin suurimmalta osin asiakkaat itse ovat olleet innoissaan menetelmästä. Ongelmana näissä tutkimuksissa on ollut se, että asiakkaan roolissa on ollut henkilö yrityksen sisällä ja tällä henkilöllä on ollut yleensä hyvin ohjelmistotekninen tausta. ATDD:ssä asiakkaan rooli on hyvin olennainen, joten lisätutkimusta tarvitaan nimenomaan tämän roolin näkökulmasta. Nykyisissä tutkimuksissa ei ole riittävän hyvin tarkasteltu aidon ulkoisen asiakkaan kokemuksia. Ulkoisen asiakkaan tapauksessa tulisi tarkemmin tutkia

pystyykö asiakas tehokkaasti ja mielekkäästi luomaan hyväksymistestejä.

ATDD:llä on myös havaittu olevan positiivisia vaikutuksia järjestelmän koodin laatuun, koska ATDD mahdollistaa automaattisen testauksen ja regressiotestauksen. Tällöin kehittäjät pystyvät turvallisesti tekemään järjestelmään muutoksia ja varmistamaan, että muutosten jälkeen mikään ei ole mennyt rikki. Tämä näkökulma vaatii kuitenkin lisätutkimusta, koska tulokset ovat pääosin kerätty haastatteluista. ATDD tulosten perusteella vaikuttaa positiivisesti vaatimusmäärittelyyn ja ohjelmistokehittäjien prosesseihin, joten seuraavissa tutkimuksissa voitaisiin selvittää myös kuinka paljon nämä vaikuttavat lopullisen järjestelmän laatuun verrattuna perinteisiin menetelmiin.

Kirjallisuudessa on esitetty pientä huolta siitä, että hyväksymistestien dokumentointi muuttuu ongelmalliseksi, kun testien määrä kasvaa. Jotta testien laadukkuus voidaan pitää yllä, täytyy niiden hallinta hoitua helposti. Tätä näkökulmaa koskevia tutkimuksia ei löytynyt lainkaan kirjallisuudesta ja tätä näkökulmaa olisi syytä tutkia lisää. Myös ATTD:n vaikutuksesta työmäärään on keskusteltu runsaasti. Tutkimukset ovat antaneet viitteitä siitä, että menetelmä vaatii hieman enemmän työtä kuin esimerkiksi TDD, mutta ero työmäärässä ei ole kuitenkaan merkittävä.

ATDD:n mielekkyys ja opittavuus on ollut erittäin positiivista tutkimusten mukaan. Opiskelijoilla tehtyjen haastatteluiden mukaan menetelmä on herättänyt suurta mielenkiintoa ja useimmat opiskelijoista ovat antaneet myönteistä palautetta menetelmästä. Yritysmaailmassa tulokset ovat myös olleet samankaltaisia. Menetelmän oppiminen on toteutettu pienellä alkukoulutuksella ja konsulttien läsnäololla eikä ongelmia ole ollut.

Tutkimusten perusteella ei voida vielä julistaa hyväksymistestivetoista ohjelmistokehitystä menestysreseptiksi, mutta lupaavia tuloksia on havaittavissa. Nämä lupaavat tulokset kuitenkin toivottavasti kannustavat lisätutkimuksiin, jolloin näemme miten suunnittelukäytäntö onnistuu todellisimmissa yritysprojekteissa.

Kirjallisuutta

Johan Andersson, Geoff Bache ja Peter Sutton. Xp with acceptance-test driven development: A rewrite project for a resource optimization system. *Lecture Notes in Computer Science*, osa 2675, sivu 1011, 2003. ISBN 978-3-540-40215-2.

Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston,, 2002. ISBN 0-321-14653-0.

Lisa Crispin. Using customer tests to drive development. *Methods & tools, Global knowledge source for software development professionals*, 2005. URL `http://www.methodsandtools.com/archive/archive.php?id=23`. Viitattu 28.11.2009.

Lisa Crispin, Tip House ja Wade Carol. The need for speed: Automating acceptance testing in an extreme programming environment. *Agile Alliance*, 2001. URL `http://www.agilealliance.org/show/1073`. Viitattu 28.11.2009.

Ward Cunningham. Fit framework-työkalun www-sivusto. URL `http://fit.c2.com`. [Viitattu 28.11.2009].

EasyAccept. Easyaccept- työkalun www-sivusto. URL `http://easyaccept.sourceforge.net`. [Viitattu 28.11.2009].

Bøerge Haugset ja Geir Kjetil Hanssen. Automated acceptance testing: A literature review and an industrial case study. *AGILE '08: Proceedings of the Agile 2008*, sivut 27–38. IEEE Computer Society, 2008. ISBN 978-0-7695-3321-6.

Grigori Melnik ja Frank Maurer. The practice of specifying requirements using executable acceptance tests in computer science courses. *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, sivut 365–370. ACM, 2005. ISBN 1-59593-193-7.

Grigori Melnik ja Frank Maurer. Multiple perspectives on executable acceptance test-driven development. *Lecture Notes in Computer Science*, osa 4536, sivut 245–249, 2007. ISBN 978-3-540-73100-9.

Grigori Melnik, Kris Read ja Frank Maurer. Suitability of fit user acceptance tests for specifying functional requirements: Developer perspective. *Lecture Notes in Computer Science*, osa 3134, sivut 60–72, 2004. ISBN 978-3-540-22839-4.

Grigori Melnik, Frank Maurer ja Mike Chiasson. Executable acceptance tests for communicating business requirements: Customer perspective. *AGILE '06: Proceedings of the conference on AGILE 2006*, sivut 35–46. IEEE Computer Society, 2006. ISBN 0-7695-2562-8.

Rick Mugridge. Managing agile project requirements with storytest-driven development. *IEEE Softw.*, 25(1):68–75, 2008. ISSN 0740-7459.

Shelly S. Park ja Frank Maurer. The benefits and challenges of executable acceptance testing. *APOS '08: Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral*, sivut 19–22. ACM, 2008. ISBN 978-1-60558-021-0.

Juha Rantanen. *Acceptance Test-Driven Development with Keyword-Driven Test Automation Framework in an Agile Software Project.* Väitöskirja, Teknillinen Korkeakoulu, tietotekniikan osasto, 2007.

Kris Read, Grigori Melnik ja Frank Maurer. Student experiences with executable acceptance testing. *ADC '05: Proceedings of the Agile Development Conference*, sivut 312–317. IEEE Computer Society, 2005. ISBN 0-7695-2487-7.

Tracy Reppert. Don't just break software make software: How storytest-driven development is changing the way qa, customers, and developers work. *Better Software, July/August*, sivut 18–23, 2004.

Filippo Ricca, Marco Torchiano, Massimiliano Di Penta, Mariano Ceccato ja Paolo Tonella. Using acceptance tests as a support for clarifying requirements: A series of experiments. *Information and Software Technology*, 51:270–283, 2009. ISSN 0950-5849.

Jacques Philippe Sauvé, Osório Lopes Abath Neto ja Walfredo Cirne. Easyaccept: a tool to easily create, run and drive development with automated acceptance tests. *AST '06: Proceedings of the 2006 international workshop on Automation of software test*, sivut 111–117. ACM, 2006. ISBN 1-59593-408-1.

Selenium. Selenium- työkalun www-sivusto. URL `http://seleniumhq.org`. [Viitattu 28.11.2009].

Richard J. Watt ja David Leigh-Fellows. Acceptance test driven planning. *Lecture Notes in Computer Science*, osa 3134, sivut 43–49, 2004. ISBN 978-3-540-22839-4.

Liite A

Taulukko 1: ATTD:n esitetyt väitteet ja empiirisistä tutkimuksista saadut tulokset.

Vaikutukset vaatimusmäärittelyyn			
Väite	**Ketkä tutkineet?**	**Tutkimustapa (Opiskelija- vai yrityskoe)**	**Todistusaineisto puolesta, vastaan vai ei voida sanoa**
+ Hyväksymistestit helpottavat vaatimusten ymmärtämistä.	Ricca et al. (2009), Haugset ja Hanssen (2008)	Molempia	Puolesta
+ Hyväksymistestien avulla lopullinen järjestelmä vastaa paremmin asiakkaan vaatimuksia.	Melnik et al. (2004), Melnik ja Maurer (2007)	Opiskelijakoe	Puolesta
+ Hyväksymistestit ovat riittäviä itsessään toimimaan vaatimusmäärittelydokumentteina.	Melnik ja Maurer (2007),Read et al. (2005)	Molempia	Puolesta
- Hyväksymistesteillä ei sovellu kaikentyyppiseen testaukseen.	Ei tutkittu	-	-

Vaikutukset kommunikaatioon			
Väite	**Ketkä tutkineet?**	**Tutkimustapa (Opiskelija- vai yrityskoe)**	**Todistus-aineisto puolesta, vastaan vai ei voida sanoa**
+ ATDD parantaa järjestelmän vaatimusten kommunikointia ohjelmistoprojektin sidosryhmien välillä	Melnik ja Maurer (2007); Andersson et al. (2003); Haugset ja Hanssen (2008); Watt ja Leigh-Fellows (2004)	Yrityskoe	Puolesta
+ Ohjelmistokehityksen tilanne on tiedossa kehitystiimille ja asiakkaalle eri projektin vaiheissa.	Ei tutkittu	-	-
Vaikutukset dokumentointiin			
- Suuren hyväksymistestimäärän hallinta ja ylläpito on haasteellista.	Ei tutkittu	-	-
Vaikutukset järjestelmän laatuun			
+ ATDD parantaa koodin laatua.	Melnik ja Maurer (2007); Melnik et al. (2006); Haugset ja Hanssen (2008)	Molempia	Puolesta
+ ATDD mahdollistaa turvallisen ympäristön koodimuutoksille.	Andersson et al. (2003); Haugset ja Hanssen (2008)	Yrityskoe	Puolesta
Vaikutukset työmäärään ja menetelmän opittavuus			
- ATDD:n oppiminen on haasteellista.	Melnik ja Maurer (2007); Melnik et al. (2006)	Molempia	Vastaan
- Hyväksymistestien luominen voi olla liian haastavaa asiakkaalle.	Melnik et al. (2006); Andersson et al. (2003)	Molempia	Ei voida sanoa
- ATDD lisää työmäärää perinteisiin ohjelmistokehitysmenetelmiin verrattuna.	Ricca et al. (2009)	Opiskelijakoe	Ei voida sanoa