Tampereen yliopisto

Kalle Virtanen

ATDD-METODOLOGIAN PILOTOINTI

ERP-liitynnän toteutus tehdasautomaatiojärjestelmään

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Professori Kari Systä
Yliopistonlehtori Terhi Kilamo
Maaliskuu 2021

TIIVISTELMÄ

Kalle Virtanen: ATDD-metodologian pilotointi
Diplomityö
Tampereen yliopisto
Tietotekniikan DI-tutkinto-ohjelma
Maaliskuu 2021

Hyväksymistestivetoinen ohjelmistokehitys (engl. Acceptance Test-Driven Development, ATDD) on ohjelmistokehitysmetodologia, jonka tarkoituksena on luoda määrittely toiminnallisista vaatimuksista hyväksymistestejä käyttäen. Tässä työssä perehdytään, kuinka metodologia tuotiin osaksi ohjelmistokehitystä Fastems Oy Ab:lla. Työn tarkoituksena on selvittää hyväksymistestivetoisen mallin vaikutuksia ohjelmistokehitykseen ja tunnistaa seuraavat askeleet kohti tehokkaampaa prosessia.

Tutkimus jakautuu kolmeen osaan. Ensimmäisessä osassa tarkastellaan hyväksymistestivetoisen mallin käyttöönottamista asiakasprojektissa pohjautuen kehitystiimin, laadunvarmistustiimin ja tuoteomistajan havaintoihin. Toisessa osassa tehdään kirjallisuuskatsaus kahteen tapaustutkimukseen projekteista, joissa onnistuneesti hyödynnettiin hyväksymistestivetoista ohjelmistokehitystä. Kolmannessa osassa käsitellään verkkokyselyn tuloksia, joiden avulla kartoitettiin hyväksymistestivetoisen ohjelmistokehityksen tilannetta Fastemsilla.

Tutkimuksesta selviää, että Fastemsin MMS:n (engl. Manufacturing Management System) ohjelmistokehitys on siirtynyt ATDD-malliin. Fastemsin tapa toteuttaa ATDD:ta perustuu niin kutsutun kolmikon (engl. Triad tai Three amigos) toimintaan. Tähän ryhmään kuuluvat toimitusomistaja, joka vastaa asiakkaan asiantuntemuksesta, testaaja, jonka vastuulla on hyväksymistestien manuaalinen suorittaminen, sekä ohjelmistokehittäjä, jonka vastuulla on luoda testejä vastaava toteutus. Kaikki kolme ovat yhdessä vastuussa hyväksymistestien kirjoittamisesta.

Tutkimus osoittaa, että uuden metodologian mukaisen ohjelmistokehitysprosessin seurauksena asiakasvaatimuksista tunnistetaan epäselvyydet aikaisemmin, jo ennen kehitystyön aloittamista. Prosessin havaittiin myös tukevan asiakkaan vaatimuksiin liittyvän tiedon keruuta ja ohjelmistokehitystyön todentamista. Sen sijaan, että tieto olisi hajallaan, ohjelmistokehityksen kannalta tarpeellinen tieto on uuden prosessin seurauksena löydettävissä ominaisuudesta (engl. Feature) ja sen hyväksymistesteistä.

Rajoittavaksi tekijäksi osoittautui asiakkaan puuttuminen prosessista. Tämän seurauksena määrittelyt eivät vastanneet asiakkaan vaatimuksia niin hyvin kuin ne olisivat voineet, prosessi vei enemmän aikaa, eikä hyväksymistestejä hyväksytetty asiakkaalla. Tutkimuksesta selviää, että hyväksymistestien seurauksena asiakasvaatimusten muuttumisesta aiheutuva päivitystyö pahimmillaan moninkertaistuu. Näiden lisäksi projektiräätälöintien hyväksymistestien rajallisen automatisoinnin havaittiin aiheuttavan puutteita regressiotestauksessa.

Asiakkaan mukaan ottaminen, koulutus ja ohjeistus, sekä hyväksymistestien kattavampi automatisointi muodostuivat seuraaviksi askeleiksi kohti tehokkaampaa ATDD-metodologian mukaista prosessia.

Avainsanat: ATDD, hyväksymistestivetoinen kehitys, testivetoinen kehitys, Gherkin

ABSTRACT

Kalle Virtanen: Commissioning of ATDD-methodology
Master's Thesis
Tampere University
Master's Degree Programme in information technology
Mar 2021

Acceptance test-driven development (ATDD) is requirements engineering methodology which creates specification out of functional requirements using acceptance tests. This study describes how the methodology was commissioned for software development at Fastems Oy Ab. Purpose of this study is to discover the effects of acceptance test-driven model on the software development and to find the next steps for making the process more efficient.

This study is divided into three parts. First part focuses on the commissioning acceptance test-driven model in a customer project based on the observations of software development team, quality assurance team and delivery owner. Second part delves into two case studies of projects that were successfully using acceptance test-driven development. In the third part results of an online survey are processed for mapping the status of acceptance test-driven development at Fastems.

Study shows that Fastems' MMS (Manufacturing Management System) software development has moved to using ATDD-model at Fastems. Fastems' way for using ATDD is based on the workings of the triad or three amigos. This group consists of delivery owner who is responsible for customer expertise, quality assurance specialist who is responsible for executing acceptance tests manually and software developer who is responsible for creating implementation that passes the tests. Everyone is responsible for writing the acceptance tests together.

Based on the study, credit to the software development process using principles of the methodology ambiguities from the customer requirements are discovered earlier, even before the development process has started. Process was also shown to support gathering requirements from the customer and helping verifying software development. Instead of information being scattered, necessary information for software development can be found from feature and its acceptance tests due to the new acceptance test-driven process.

Customer being left out of the process was discovered being a hindering factor. This caused the specifications not to match customer requirements as well as they could have, process took longer time and acceptance tests were not verified with the customer. Study shows that at worst acceptance tests multiply the work needed to keep specifications up to date when customer requirements change. In addition, the limited automation of project specific acceptance tests causes deficiency in regression testing.

Inclusion of the customer, providing training and instructions and more comprehensive automation of the tests formed as the next steps towards more efficient process following ATDD methodology.

Keywords: ATDD, acceptance test-driven development, test-driven development, Gherkin

ALKUSANAT

Tämän työn valmistumiseen vaadittiin 193 päivää, 208 committia ja reilu annos viskiä. Alkusanoiksi tarkoitukseni oli liittää lista versionhallinnan commit-viesteistä, mutta akateeminen paperi ei ole tehty kestämään niiden sisältämää kieltä. Sen sijaan käytän nämä alkusanat kiittääkseni seuraavia henkilöitä: Kalapuikko-tiimiä ATDD-menetelmän harjoittelusta ja hiomisesta, Jani Nietulaa työn katselmoinnista ja kehitysideoista, Antti Mertsolaa menetelmään liittyvästä ideoinnista ja vastauksista testaukseen liittyvissä kysymyksissä, Kari Systää todella nopeasta kommentoinnista, joka mahdollisti tämän työn nopean aikataulun, ja Jetta Sirolaa Kieli-oppiKatselm0innista ja yleisestä kannustamisesta.

Tampereella, 4.3.2021

Kalle Virtanen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

ATDD	engl. Acceptance Test-Driven Development, hyväksymistestivetoinen ohjelmistokehitys
BDD	engl. Behavior Driven Development, käyttäytymisvetoinen ohjelmistokehitys
CDS	engl. Clinical Decision Support, kliinisen päätöksen tukityökalu
DO	engl. Delivery Owner, toimitusomistaja, tuoteomistaja Fastemsilla
EHR	engl. Electronic Health Records, sähköinen potilastietokirjasto
ERP	engl. Enterprise Resource Planning, toiminnanohjausjärjestelmä
FMS	engl. Flexible Manufacturing System, Fastemsin tarjoama tehdasautomaatiointegraatio
MMF	engl. Minimum Marketable Feature, nopeasti kehitettävä ominaisuus, joka tarjoaa arvoa käyttäjälle
MMS	engl. Manufacturing Management System, Fastemsin tarjoama tehdasautomaatioon tarkoitettu ohjelmisto
NC	engl. Numerical Control, työstökoneiden automatisoitu ohjaus
PLC	engl. Programmable Logic Controller, ohjelmoitava logiikkaohjain, tietokone tarkoitettu tuotantolaitteiden ohjaukseen
QA	engl. Quality Assurance, laadunvarmistus
STDD	engl. Story Test-Driven Development, Käyttäjätarinatestivetoinen ohjelmistokehitys
SW	engl. Software, ohjelmisto
TDD	engl. Test-Driven Development, testivetoinen ohjelmistokehitys
UTDD	engl. Unit Test-Driven Development, yksikkötestivetoinen ohjelmistokehitys
XML	engl. Extensible Markup Language, rakenteellinen kuvauskieli

1. JOHDANTO

Luvussa 1.1 perehdytään työn taustaan ja kerrotaan mitä ja miksi tehdään. Luvussa 1.2 määritellään työn tutkimuskysymykset. Seuraavaksi luvussa 1.3 kerrotaan, miten työ on rajattu. Viimeiseksi luvussa 1.4 avataan työn rakennetta ja annetaan lyhyt kuvaus jokaisesta luvusta.

1.1 Työn tausta

Ohjelmistoprojektit epäonnistuvat liian usein sen vuoksi, että toimitettu tuote ei vastaa asiakkaan vaatimuksia. Asiakkaan näkökulmasta kehittäjät eivät kuuntele, mitä heiltä pyydetään. Toisaalta kehittäjien mielestä asiakas ei kerro, mitä he haluavat tai he eivät edes tiedä, mitä he haluavat. Kehittäjän näkökulmasta asiakasvaatimukset ja toiminnalliset kuvaukset toimitettavasta järjestelmästä voivat olla hajallaan ja piilotettuina useamman eri dokumentoinnin hallintaa ”helpottavan” työkalun taakse, jolloin kehittäjän on äärimmäisen vaikeaa saada ajantasaista ja yhtenevää tietoa asiakkaan vaatimuksista ja toteuttaa asiakkaan tarpeita vastaava järjestelmä. Hyväksymistestivetoinen ohjelmistokehitys tarjoaa prosessin, joka ohjaa kaikkia sidosryhmiä keskustelemaan toteutettavista ominaisuuksista yhteisellä kielellä ja luo vaatimuksista automatisoitavat hyväksymistestit, jotka muodostavat toimitettavan järjestelmän määrittelyn.

Fastems Oy Ab, johon myöhemmin viitataan nimellä Fastems, toimittaa monipuolisia tehdasautomaatioratkaisuja globaalisti ja yrityksen tarjoama ohjelmisto MMS (Manufacturing Management System) on vain yksi kerros tarjottavaa monitasoista järjestelmäkokonaisuutta. Aloittaessani työskentelemään Fastemsilla 2017, yrityksessä oltiin vasta ottamassa käyttöön ketteriä menetelmiä. Testaus oli yksittäisen ohjelmoijan vastuulla, eikä testaukseen ollut selvää yhteistä ohjeistusta. Aikaisempaa työskentelyprosessia, jossa testauksen tapaan ohjelmistokehitys oli keskittynyt projektikohtaisesti yksittäisiin ohjelmoijiin, oltiin vaihtamassa scrum-tiimivetoiseen sprinteissä etenevään ketterään malliin. Työskentelyprosessia pyritään jatkuvasti kehittämään paremmaksi ja tämän myötä esimerkiksi testiautomaatio on saatu integroitua osaksi ohjelmistokehitystä ja muodostettua laadunvarmistustiimi, millä yhdessä varmistetaan toimitettavien järjestelmien laadukkuus.

Nyt vuorossa on hyväksymistestivetoinen ohjelmistokehitys ja tämä työ käsittelee tämän menetelmän käyttöönottamista osaksi ohjelmistotiimien työskentelyprosessia. Työssä

keskitytään projektiin, jossa luodaan liityntä asiakkaan toiminnanohjausjärjestelmän ja Fastemsin tarjoaman tehdasautomaatiojärjestelmän välille. Projektityöskentelyn lisäksi tietoa menetelmästä haetaan tarkastelemalla kahta tapaustutkimusta, joissa projektit on viety loppuun onnistuneesti menetelmää käyttäen. Viimeiseksi kaikille Fastemsilla menetelmää käyttäneille lähetetään verkkokysely, jolla kartoitetaan menetelmän hyötyjä ja haasteita yritystasolla.

Menetelmän käyttöönottoon osallistuu muutama ohjelmistokehitystiimi, laadunvarmistustiimi ja toimitusomistaja. Mukana on myös Fastemsin ulkopuolelta palkattu yritys, joka on perehdyttänyt ja auttanut tiimit alkuun hyväksymistestivetoisen kehityksen osalta.

1.2 Työn tavoite

Työn tavoitteena on perehtyä hyväksymistestivetoiseen kehitykseen ja selvittää, miltä osin se voidaan tuoda osaksi nykyistä työskentelyprosessia Fastemsilla. Prosessi on tarkoitus ottaa käyttöön siten, että vuoden 2020 loppuun mennessä 50 % sovelluskehityksen alaisista projekteista tehtäisiin noudattamalla menetelmää. Vuoden 2021 loppuun mennessä tavoite on 90 %. Tavoitteena on myös tutkia, mitkä ovat metodologiasta saatavat hyödyt ja mitkä sen haasteet. Lisäksi selvitetään, mitä prosessin muuttaminen edellyttää laajamittaisemmin ja minkälaisia muutoksia se aiheuttaa nykyisissä rooleissa ja mihin rooleihin muutos vaikuttaa. Lopuksi selvitetään, miltä osin prosessia voidaan viedä eteenpäin. Työn seurauksena syntyy yhteinen ohjeistus hyväksymistestivetoiselle kehitykselle.

Edelliseen pohjautuen, tämä työ pyrkii vastaamaan seuraaviin kysymyksiin.

- Minkälainen on Fastemsin hyväksymistestivetoinen prosessi?
- Mitkä ovat Fastemsin hyväksymistestivetoisesta prosessista saadut edut?
- Mitkä ovat Fastemsin hyväksymistestivetoisen prosessin ongelmat?
- Miten Fastemsin hyväksymistestivetoista prosessia voidaan tulevaisuudessa kehittää eteenpäin?

Kysymyksiin vastataan yhdistelemällä havaintoja ja tuloksia tässä työssä käsiteltävästä asiakasprojektista, kahdesta tapaustutkimuksesta ja verkkokyselystä.

1.3 Työn rajaus

Tutkimus rajoittuu yksittäisen projektitiimin havaintoihin projektista, jossa hyväksymistestivetoinen kehitys otettiin ensimmäistä kertaa osaksi prosessia, kahteen tapaustutkimukseen onnistuneista projekteista, sekä pilotointijaksoon osallistuneilta kerättyyn verkkokyselyyn ja sen pohjalta tehtyihin johtopäätöksiin.

1.4 Työn rakenne

Tämän työn rakenne on seuraava:

1. luvussa esitellään työn aihe, tavoite ja rajaus.
2. luvussa esitellään työssä käytettävä aineisto ja tutkimusmenetelmät.
3. luvussa perehdytetään lukija testiautomaatioon ja Gherkiniin, sekä tehdään lyhyt katsaus testaukseen Fastemsilla.
4. luvussa käydään läpi eri testivetoisia menetelmiä ja niiden eroavaisuuksia.
5. luvussa käydään läpi, miten hyväksymistestivetoinen ohjelmistokehitys otettiin käyttöön Fastemsilla ja mitkä ovat menetelmän hyödyt ja haitat.
6. luvussa perehdytään kahteen tapaustutkimukseen, joissa projektit ovat onnistuneet ja niissä on hyödynnetty hyväksymistestivetoista ohjelmistokehitystä.
7. luvussa käydään kyselyn tulokset läpi ja vedetään niistä johtopäätökset.
8. luvussa tehdään yhteenveto työstä.

2. AINEISTO JA TUTKIMUSMENETELMÄT

Tässä luvussa käydään läpi työssä käytettävä aineisto ja tutkimusmenetelmät. Luvussa 2.1 käydään aineistoa läpi ja perustellaan verkkokyselyn käyttö ja käydään sen rakennetta läpi. Luvussa 2.2 tehdään katsaus tutkimusmenetelmiin, joita hyödynnetään aineiston tulkitsemisessa.

2.1 Aineisto

Tässä työssä hyödynnetään kolmea eri aineistoa. Ensimmäisen aineisto perustuu asiakasprojektissa toimineiden henkilöiden havaintoihin hyväksymistestivetoisesta menetelmästä ja on luonteeltaan empiiristä tietoa. Toisena aineistona käytetään kahta tapaustutkimusta onnistuneista ohjelmistoprojekteista, joissa käytettiin hyväksymistestivetoista ohjelmistokehitystä.

Kolmantena aineistona käytetään verkossa täytettävää Microsoft Forms -kyselyä, joka lähetetään kaikille Fastemsin työntekijöille ja alihankkijoille, jotka ovat työskennelleet hyväksymistestivetoisia menetelmiä käyttäen. Verkkokysely on nopea tapa kerätä tietoa suureltakin määrältä vastaajia ja luonteensa vuoksi jokainen kyselyyn osallistuva voi vastata siihen silloin, kun se heille parhaiten sopii. Kyselyn etuna on myös se, että se voidaan toteuttaa lähes ilman kuluja, maksaen vain vähän kyselyyn vastaavien aikaa. Verkkokyselyllä saatu data on myös välittömästi saatavilla analysoitavaksi, vähentäen aikaa, joka muuten kuluisi vastausten kokoamiseen. (Walsh 2016)

Kyselyn mukana osallistujille lähetetään aineisto, johon on kerättynä Fastemsin nykyinen ohjeistus liittyen työskentelyyn hyväksymistestivetoista ohjelmistokehitystä käyttäen. Ohjeistuksessa on kuvattu eri rooleissa toimivien vastuualueita (sovelluskehittäjä, testaaja, toimitusomistaja) sekä koottu yleisiä käytäntöjä. Ohjeistus on toteutettu yhteistyössä laadunvarmistustiimin kanssa. Ohjeistuksen on jatkossa tarkoitus toimia tukipilarina hyväksymistestivetoisten menetelmien käytössä, mutta verkkokyselyn kannalta sen on tarkoitus pohjustaa kyselyn kysymyksiä ja herättää ajatuksia, joiden avulla ohjeistusta itseään voidaan parantaa ja prosessia kehittää.

Kysely on luotu yhteistyössä laadunvarmistustiimin ja menetelmään perehtyneen esimiehen kanssa. Kyselyn tavoitteena on löytää hyväksymistestivetoisen menetelmän kipukohdat ja kartoittaa hyväksi havaittuja tapoja toimia. Kysely on rakennettu siten, että aluksi vastaajan taustat kartoitetaan, selvittämällä vastaajan rooli yrityksessä ja kuinka

paljon tällä on käytännön kokemusta menetelmästä. Jatkokysymykset riippuvat vastanneen roolista, sillä menetelmään liittyvät vastuualueet ovat roolikohtaisia, minkä vuoksi kaikki kysymykset eivät ole oleellisia kaikkien vastaajien kesken.

Tämän jälkeen selvitetään yleisellä tasolla vastaajien tyytyväisyyttä menetelmän eri osa-alueisiin. Karkean yleiskuvan jälkeen kartoitetaan yksityiskohtaisemmin, minkä koetaan onnistuneen ja mitkä asiat tuottavat ongelmia. Suljettuihin kysymyksiin vastataan Likert-asteikolla, joka kuvastaa, kuinka vahvasti vastaaja kokee olevansa samaa tai eri mieltä esitetyn väittämän kanssa. Lopuksi kyselyssä on avoimia kysymyksiä, joiden avulla kerätään vapaamuotoisempaa kvalitatiivista eli laadullista tietoa. Flickin (2018) mukaan laadullisella tiedolla tarkoitetaan joukkoa kielellistä tai visuaalista materiaalia, jonka avulla voidaan analysoida subjektiivisia ja kollektiivisia kokemuksia ja näihin pohjautuvia päätöksentekoprosesseja. Laadullista tietoa käytetään myös selvittämään ja kuvaamaan ongelmia, rakenteita ja prosesseja tavoissa ja käytännöissä. Näiden kysymysten joukkoon kuuluvat myös palaute ja kehitysideat, joiden avulla yhteistä ohjeistusta ja prosessia tullaan kehittämään eteenpäin.

2.2 Tutkimusmenetelmät

Tässä työssä käsitelty projekti viedään läpi käyttäen hyväksymistestivetoista ohjelmistokehitystä. Projektista saatu empiirinen tieto saavutetaan tapaustutkimuksen keinoin. Projektissa toimineilta henkilöiltä keskusteluina kerätty tieto ja havainnot analysoidaan ja niiden pohjalta muodostetaan johtopäätökset hyväksymistestivetoiseen ohjelmistokehitykseen liittyen.

Verkkokyselyn suljetuista kysymyksistä kerättyyn tietoon hyödynnetään tilastollisia analyysimenetelmiä ja ne ristiintaulukoidaan vastaajan roolin mukaan kolmeen eri kategoriaan: ohjelmistokehitys (engl. Software, SW), laadunvarmistus (engl. Quality Assurance, QA) ja toimitusomistaja (engl. Delivery Owner, DO). Tällä tavoin voidaan eritellä menetelmän hyötyjä ja ongelmakohtia roolikohtaisesti. Vastaukset ristiintaulukoidaan erikseen myös sen mukaan, kuinka monta kertaa vastaaja on osallistunut testienkirjoitussessioihin. Tämän avulla saadaan vertailtua, lievenevätkö ongelmat kokemuksen karttuessa tai löytyykö sen myötä mahdollisesti uusia ongelmia. Avointen kysymysten vastauksiin käytetään sisällönanalyysia kategorisoimalla vastauksia niiden ilmaantuvuuden mukaan. Tämän tutkimusmenetelmän proseduurien avulla voidaan muodostaa vapaasta tekstistä oikeat johtopäätökset (Weber 1990).

3. KETTERÄ TESTAUS JA TESTIAUTOMAATIO

Ohjelmistoalan kehittyessä myös asiakkaiden vaatimukset ja odotukset ovat kasvaneet. Tilatuilta järjestelmiltä vaaditaan entistä korkeampaa laatua lyhyemmässä ajassa ja pienemmällä hinnalla. Tämän kasvaneen kaupallisen paineen seurauksena ohjelmistoalan yritykset ovat joutuneet kehittämään prosessejaan tehokkaammiksi ja kilpailukykyisemmiksi, sekä etsimään uusia tapoja viedä ohjelmistoprojekteja loppuun nopeammin, laadukkaammin ja halvemmalla. (Watkins 2009)

Aikaisemmin laajassa käytössä olleesta vesiputousmallista (Waterfall), jossa tarkkaan määritellyt vaiheet etenevät peräkkäin vaiheesta toiseen, on monissa yrityksissä siirrytty käyttämään ketteriä menetelmiä (Agile). Vesiputousmallia on kritisoitu sen puutteellisesta reaktiivisuudesta asiakkaan toiveisiin ja siitä, että testaus tapahtuu liian myöhäisessä vaiheessa prosessia. Yleisesti ajatellaan, että jokainen vaihe, jossa vika epäonnistutaan löytämään, vian korjaaminen kallistuu kymmenkertaisesti seuraavassa vaiheessa. Näistä syistä vesiputousmallilla tehdyt projektit ovat todennäköisiä viivästymään, aiheuttamaan lisätöitä ja ylittämään budjetin. (Watkins 2009)

Aikaisemmin testausta saatettiin pitää vain irrallisena asiana, joka tehtiin toteutuksen jälkeen, jos sitä tehtiin ollenkaan. Nykyään testaus nähdään oleellisena osana ohjelmistokehitystä ja sen toimittamista. On kuitenkin ymmärrettävä, että kehnosti toteutettu testaus voi olla yhtä huono tai huonompikin asia, kuin testauksen puuttuminen. Huonosti toteutettu testaus voi maksaa huomattavia määriä aikaa, rahaa ja vaivaa, lopulta kuitenkaan parantamatta testattavan järjestelmän laatua. (Watkins 2009)

Monissa yrityksissä, joissa näennäisesti on otettu käyttöön tarkkaan dokumentoitu prosessi parhaista mahdollisista sovelluskehityksen ja testauksen menetelmistä, voi tiimien kesken tai sisällä olla täysin toisistaan poikkeavat testaustekniikat, dokumentointi ja termistö. Yleisin ja todennäköisin syy tähän on, että yrityksissä ei katsota asiaa kauaskantoisesti, vaan pyritään vain edistämään projektia nopeimmalla mahdollisella tavalla, eikä katselmoida ja korjata todellista rakenteellista ongelmaa. (Watkins 2009)

3.1 Testaus Fastemsilla

Fastemsilla testiautomaatiota hyödynnetään kolmella eri tavalla. Kaikissa tavoissa käytetään Bitbucket-versionhallintaa ja automaatiopalvelin Jenkinsiä. Bitbucket on Git-pohjainen koodin isännöintiin ja tiimien yhteistyöhön tarkoitettu työkalu (Bitbucket, 2021).

Jenkins on avoimen lähdekoodin automaatiopalvelin, jolla voidaan automatisoida ohjelmistojen kääntämiseen, testaamiseen, toimittamiseen tai käyttöönottoon liittyviä tehtäviä (Jenkins, 2021). Kun ohjelmistokehittäjä puskee versionhallintaan tekemänsä koodimuutokset ja luo muutoksistaan katselmointi- ja yhdistämispyynnön (engl. pull request, myöhemmin PR) toiseen versionhallinnan haaraan, testiautomaatioputki yhdistää haarojen koodit etukäteen, kääntää ne ja ilmoittaa onnistuiko kääntäminen. Bitbucket tunnistaa yhdistämiskonfliktit (engl. merge conflict), mutta se ei takaa koodien kääntymistä, minkä vuoksi on tärkeää, että PR:n yhteydessä muutoksille suoritetaan kyseinen testaus.

Kun ennalta määritettyyn versionhallinnan haaraan tulee muutoksia, se laukaisee toisen testiautomaatioputken, joka Jenkinsin toimesta pystyttää virtuaalipalvelimelle Docker-kontin. Docker-kontti pakkaa ohjelmakoodin ja sen ajoon vaaditut riippuvuudet siten, että kontin sisältö voidaan suorittaa ympäristöstä riippumatta (Docker 2021). Konttiin pakataan käännetty ohjelmakoodi muutoksineen. Projektista riippumatta konttia vasten ajetaan smoke-testit, minkä tarkoitus on varmistaa ohjelman tärkeimpien ominaisuuksien toiminnallisuus. Testisettiin voidaan tarvittaessa lisätä tuotannon ajoon liittyvät testit. Projektien räätälöinnit kuitenkin herkästi rikkovat tämän testisetin, jolloin myös testisetti tulee räätälöidä vastaamaan projektin muutoksia. Kyseisen testiautomaatioputken testiajo on toteutettu Python-pohjaisella Robot Frameworkilla, joka on geneerinen avoimen lähdekoodin automatisointikehys, jolla voidaan käyttää testi- ja robottiprosessiautomaatioon (Robot Framework 2021). Ohjelmistokehitys Fastemsilla tapahtuu pääosin C#-kielellä, jonka vuoksi testausta on alettu siirtämään Robot Frameworkista Microsoftin .NET-ympäristöön.

Kolmas testiautomaatioputki on tarkoitettu rajapinta- ja hyväksymistestaukseen. Itse testiautomaatioputki on sama kuin edellisessä, mutta ajettavat testit sijaitsevat samassa .NET-ympäristössä kuin muu ohjelmakoodi. Hyväksymistestit ovat tuotavissa Jirasta Zephyriä (aikaisemmin Test Management for Jira) ja SpecFlowta käyttämällä C#-kielelle. Jira on tuoteperhe, joka koostuu kaikentyyppisistä työn hallintaan liittyvistä työkaluista, jotka ovat suunniteltu erityisesti ohjelmisto-, IT-, liiketoiminta- ja Ops-tiimeille (Jira 2021). Zephyr on joustava testienhallintaratkaisu Jirassa, joka on kohdistettu testien suunnitteluun, suoritukseen ja automaatioon (Zephyr 2021). SpecFlow on .NET-ympäristöön tarkoitettu avoimen lähdekoodin sovelluskehys testivetoisia menetelmiä varten, minkä avulla voidaan yhdistää testiautomaatio ominaisuustiedostoihin (SpecFlow 2021). Käyttöliittymän testaukseen kaikissa tapauksissa käytetään Seleniumin ChromeDriveria,

jonka avulla voidaan ohjata Chrome-selainta käyttäjän tapaan (Selenium 2021). Laadunvarmistustiimi päättää automatisoidun testauksen laajuudesta projektikohtaisesti ja muuttaa projektin asetuksia Bitbucketissa sen mukaisesti.

Fastemsin laadunvarmistustiimi on aloittanut ajamaan manuaalisesti projektien hyväksymistestejä. Ohjelmistokehittäjä on vastuussa, että hänen toteutuksensa läpäisee tehtävään liittyvät hyväksymistestit, mutta ominaisuuden valmistuessa se siirtyy kehitystiimiltä laadunvarmistukseen. Laadunvarmistustiimin vastuulla on varmentaa, että ominaisuus läpäisee kaikki sille asetetut hyväksymistestit. Hyväksymistestejä ajetaan kolmessa eri ympäristössä: simuloitu (ei oikeita laitteita sovelluksen vastinparina), tehdastestit (rajattu määrä oikeita laitteita) ja käyttöönottotestit (kaikki laitteet).

3.2 Gherkin

Gherkin on liiketoimintatason kieli, jolla voidaan kuvata toiminnallisuutta menemättä toteutusteknisiin yksityiskohtiin. Gherkin määrittelee testit selko- ja toimialuekohtaista kieltä. Gherkin auttaa ymmärtämään vaatimuksia asiakkaan näkökulmasta. (Nicieja 2018)

Gherkin on oleellinen osa Fastemsilla käytettävää hyväksymistestivetoista ohjelmistokehitystä. Se luo yhteisen rakenteen ominaisuuksien toiminnallisuuksista keskustelemiseen ja sama rakenne mahdollistaa hyväksymistestien automatisoinnin. Myöhemmin tässä työssä Gherkinin ominaisuudesta (engl. feature) tullaan käyttämään termiä hyväksymistesti, sillä se vastaa paremmin Fastemsin menettelyä ja termistöä. Fastemsilla ominaisuudella tarkoitetaan kokonaisuutta, johon kuuluu useampia kehitystehtäviä ja hyväksymistestejä.

Kuva 1: Gherkinin Given-When-Then-rakenne

Gherkinillä voidaan tuoda liiketoiminnan vaatimukset lähemmäksi koodia ja automatisoida vaatimusten testaus. Gherkinissä käytetään alakohtaista sanastoa, mutta lukijalla ei tarvitse olla teknistä taustaa ymmärtääkseen sitä. Itseasiassa yksi Gherkinin tarkoituksista on muodostaa yhteinen kieli kaikkien sidosryhmien kesken ja auttaa kehittäjiä

ymmärtämään vaatimuksia asiakkaan näkökulmasta. Yksinkertaisesti Gherkin on suoritettava määrittely, joka jakautuu kahteen eri kerrokseen: määrittelyyn ja automaatioon. Määrittelykerroksella tarkoitetaan Gherkinillä kirjoitettuja hyväksymiskriteereitä ja automaatiokerroksella suoritettavia hyväksymistestejä, jotka varmistavat, että järjestelmä vastaa sen määrittelyitä. Gherkin järjestää hyväksymiskriteerit skenaarioiksi, jotka noudattavat kuvassa 1 esitettyä Given-When-Then-rakennetta. (Nicieja 2018)

```
Feature: Gherkin-syntaksin esittely

    Tämän ominaisuuden on tarkoitus esitellä
    Gherkin-syntaksia lukijalle.

    Background:
        Given Lukija lukee kappaletta 3
            And Lukija näkee kuvan, jossa tämä ominaisuus esitellään


    Scenario: Given-When-Then -rakenteen esittely

        Tämä esimerkki esittelee lukijalle Given-When-Then -rakenteen, jota
        jokaisen testitapauksen on noudatettava

        Given Skenaario koostuu vain pakollisista Gherkin-avainsanoista
        When Lukija lukee skenaarion
        Then Lukija ymmärtää Given-When-Then -rakenteen

    Scenario: Examples-avainsanan esittely

        Tässä esimerkissä esitellään lukijalle Examples-avainsanan käyttö

        Given Skenaariossa on käytetty Examples-avainsanaa
            And arvoksi on annettu <Arvo_1>
        When Lukija lukee skenaarion
        Then Lukija huomaa, että arvosta <Arvo_1> seuraa <Arvo_2>
            And Lukijalle kerrotaan lisäksi <Kolmas_parametri>

        Examples:
        | Arvo_1 | Arvo_2 | Kolmas_parametri |
        |   1    |    2   | Lue testi uudelleen toisen rivin arvoilla |
        |   2    |    4   | Nyt ymmärrät miten Examples-avainsana toimii |
```

Kuva 2: Esimerkki Gherkin-syntaksista

Gherkin koostuu joukosta avainsanoja, jotka muodostavat hyväksymistestin rakenteen. Näitä avainsanoja ovat esimerkiksi kuvassa 1 esitetyt Given, When ja Then. Näitä avainsanoja seuraa liiketoimintalogiikkaa kuvaavaa vapaamuotoista kieltä, jota kutsutaan askeleeksi (step). Esimerkkinä kuvassa 2 Given on Gherkinin avainsana ja sitä seuraava lause ”Skenaario koostuu vain pakollisista Gherkin-avainsanoista”, on askel. Askelta vastaava toiminnallisuus tulee löytyä automaatiokerroksen testikoodista, joka toteuttaa hyväksymistestin. (Nicieja 2018)

Feature-avainsanalla merkitään uuden määrittelyn alkamiskohta. Tätä avainsanaa seuraa hyväksymistestin nimi ja vapaamuotoista tekstiä, joka kuvaa hyväksymistestin toiminnallisuutta. Kuvan 2 esimerkissä ominaisuutena on ”Gherkin-syntaksin esittely” ja sen alla hyväksymistestin kuvaus. Perinteisesti yhtä tiedostoa (feature file) kohden on vain yksi hyväksymistesti. (Nicieja 2018)

Given-avainsanalla määritetään skenaarion tai testitapauksen ennakkoehdot, joiden tulee täyttyä, jotta skenaario voidaan suorittaa. Useampia ehtoja voidaan yhdistää And-avainsanalla, jolloin kaikkien näiden ehtojen tulee olla voimassa. Vastaavasti But-avainsanaa voidaan käyttää ehtojen yhdistämiseen. Toiminnallisuudeltaan But ja And vastaavat toisiaan, mutta sitä avainsanaa pyritään käyttämään, joka on semantiikan kannalta sopivampi ja muodostaa selkeämmän testitapauksen. (Nicieja 2018)

When-avainsanalla määritetään skenaarion mekanismi (action), joka tapahtuu Givenin määrittelemässä kontekstissa. Mekanismeja voidaan yhdistää samalla tavalla kuin ennakkoehtoja, mutta on suositeltavaa käyttää vain yhtä, jotta skenaariot pysyvät selkeämpinä ja helpompina lukea. Then-avainsanalla määritetään skenaarion lopputulema, joka on seurausta skenaarion ennakkoehdoista ja mekanismista. Useampia lopputulemia voidaan yhdistää samalla tavalla kuin ennakkoehtoja. (Nicieja 2018)

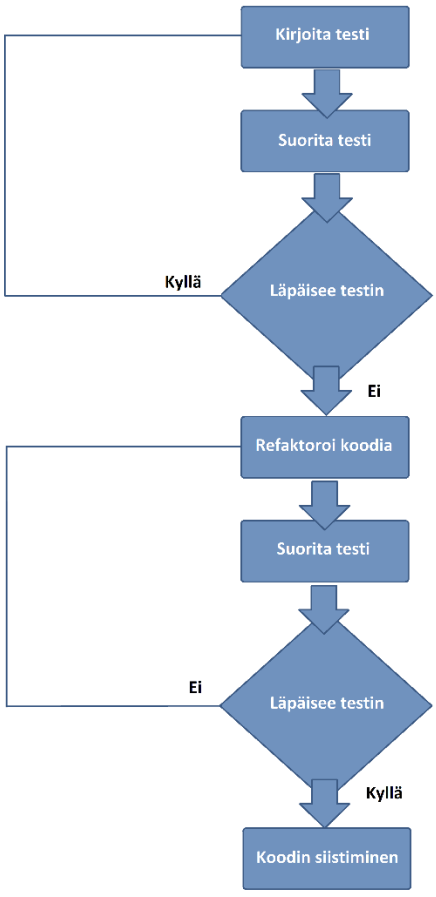
Scenario-avainsanalla eritellään useampia testitapauksia hyväksymistestin sisällä. Scenario-avainsana vastaa Feature-avainsanaa monella tapaa. Molemmat nimetään ja molemmille kirjoitetaan kuvaus. Kuvassa 2 on yhteen hyväksymistestiin kirjoitettu kaksi eri skenaarioita: ”Given-When-Then-rakenteen esittely” ja ”Examples-avainsanan esittely”. Jokaista skenaariota kohden automaatiokerroksessa tulee löytyä yksi tai useampi testitapaus. Background-avainsanalla voidaan määrittää yhteiset askeleet hyväksymistestin kaikkien skenaarioiden kesken, jolloin niitä ei tarvitse jokaiseen skenaarioon kirjoittaa erikseen. (Cucumber 2021)

Examples-avainsanalla voidaan kuvan 2 mukaisesti parametrisoida hyväksymistestin skenaarioita. Kenttään nimetään haluttu määrä sarakkeita, joihin testiskriptissä voidaan viitata sarakkeen nimellä, joka on kulmasulkeiden ympäröimä. Sarakkeille sitten määritetään arvot järjestyksessä ja pystyviivoin eroteltuna toisistaan. Jokainen näin määritetty rivi vastaa yhtä variaatiota skenaarioista. Automaatiokerroksessa jokainen tällainen variaatio on skenaariota vastaava oma testitapauksensa. (Cucumber 2021)

4. TESTIVETOISET MENETELMÄT

TDD (Test-Driven Development), BDD (Behavior-Driven Development) ja ATDD (Acceptance Test-Driven Development) ovat kaikki ketterän ohjelmistokehityksen menetelmiä. Kaikki kolme ohjelmistokehitysmenetelmää noudattavat niin kutsuttua shift-left periaatetta (Firesmith 2015). Periaatteella pyritään aikaistamaan ja tuomaan testaus osaksi koko projektin elinkaarta sen sijaan, että se olisi vain yksittäinen vaihe sovelluskehityksen lopussa (Firesmith 2015). Projektin alkuvaiheessa luodut testit itsessään jo muodostavat pohjan ohjelmiston toteutukselle. Tämä mahdollistaa lyhyemmän palautesilmukan, kun vasta valmistunutta toteutusta voidaan välittömästi testata aikaisemmin luotuja testejä vasten. (Moe 2019)

Nämä kolme menetelmää ovat keskenään hyvin samankaltaisia ja niiden nimiä voidaan helposti käyttää ristiin (Gojko 2011). Tässä kappaleessa perehdytään näiden neljän menetelmän yhteneväisyksiin ja eroavaisuuksiin. Näin saadaan muodostettua parempi kokonaiskuva siitä, miten ATDD sopii osaksi tätä kokonaisuutta.



Kuva 3: TDD-, BDD- ja ATDD-vuokaavio

4.1 Testivetoinen ohjelmistokehitys (TDD)

Testivetoinen ohjelmistokehitys, josta voidaan myös käyttää termiä UTDD (Unit Test-Driven Development) perustuu lyhyisiin sykleihin, joissa luodaan ensin yksikkötestit ja sitten koodi, joka läpäisee testattavan toiminnallisuuden. Ohjelmistoon lisättävää toiminnallisuutta ei katsota hyväksyttäväksi toteutukseksi ennen kuin se läpäisee sille kohdistetut testit ja kaikki ohjelmiston aikaisemmin läpäisemät yksikkötestit. TDD ei ole testaustekniikka, vaan ohjelmointi- ja suunnittelukäytäntö. Tässä menetelmässä ohjelmistoarkkitehtuurilliset ja korkean tason suunnitteluratkaisut tehdään ennen yksikkötestejä läpäisevää ohjelmointia. Tämä suunnitteluvaihe ei kuitenkaan ota kantaa toteutukseen yksityiskohtaisella tasolla. TDD:n iteratiivisen ja inkrementaalisen prosessin seurauksena järjestelmän matalan tason rakenne muodostuu ja kehittyy itsestään sovelluskehityksen aikana. Menetelmän yksittäinen sykli rakentuu seuraavasti:

1. Kirjoitetaan yksikkötesti (ei ohjelmakoodia vielä tässä vaiheessa).
2. Ajetaan luotu testi, jotta nähdään, että se epäonnistuu ja samalla varmennetaan, että toiminnallisuus ei ole jo olemassa.
3. Luodaan vähimmäistoteutus, joka läpäisee kirjoitetun testin.
4. Ajetaan testi ja kaikki aikaisemmat ohjelmakoodia vastaavat testit. Mikäli yksikin testitapaus epäonnistuu, palataan takaisin kohtaan 3.
5. Refaktoroidaan ohjelmakoodia siten, että sen kompleksisuus vähenee ja ymmärrettävyys sekä ylläpidettävyys kasvaa.
6. Ajetaan kaikki testitapaukset uudelleen. Mikäli yksikin testitapaus epäonnistuu, palataan kohtaan 5 ja korjataan virheet.
7. Siirrytään seuraavaan toiminnallisuuteen ja aloitetaan kohdasta 1.

Artikkelissaan Latorre tiivistää TDD:n vaikutuksia sovelluskehitykseen. TDD:n on havaittu parantavan ohjelmiston laatua, mutta samalla vähentävän tuottavuutta. Tuottavuuden väheneminen on perusteltu ajalla, joka kuluu automatisoitujen yksikkötestien kirjoittamiseen. Toisaalta, mikäli kattavampi testaus tuottaa laadukkaampia ohjelmistoja, voidaan argumentoida, että testien kirjoittamiseen kulutettu aika saadaan takaisin vähentyneiden vikakorjausten muodossa. (Latorre 2014)

4.2 Käyttäytymisvetoinen ohjelmistokehitys (BDD)

Käyttäytymisvetoinen ohjelmistokehitys perii kaikki TDD:n hyödyt ja se pohjautuukin moniin TDD:n periaatteisiin ja menetelmiin. BDD kuitenkin vie menetelmän astetta pidemmälle yhdistelemällä siihen asioita toimialuevetoisesta kehityksestä (domain-driven-design, DDD) ja oliokeskeisestä analyysistä. Näistä saatujen jaettujen työkalujen ja prosessin avulla kehittäjät sekä liiketalouden henkilöstö voivat yhdessä osallistua ohjelmistokehitykseen. Menetelmän kehittäjää, Dan Northia mukaillen BDD:tä voidaan kuvailla seuraavasti: BDD on toisen sukupolven, ulkoapäin sisälle, vetopohjainen, monisidosryhmäinen, moniskaalautuva, korkeasti automatisoitava, ketterä metodologia. Se kuvastaa syklin vuorovaikutuksia tarkkaan määritellyin lopputulemin, jonka ansiosta saadaan toimitettua toimiva, testattu ohjelmisto, jolla on merkitystä. (Ye 2013)

BDD:ssä keskitytään luomaan ominaisuus, joka on minimi siitä, mitä voidaan myydä (Minimum Marketable Feature, MMF), mutta josta on saatavissa irti suurin mahdollinen hyöty. BDD luo yhteisen kielen kehittäjien ja liiketalouden henkilöstön kesken, jolloin väärinymmärrysten, ylimääräisen koodin ja toiminnallisuuden määrää vähenee huomattavasti. (Ye 2013)

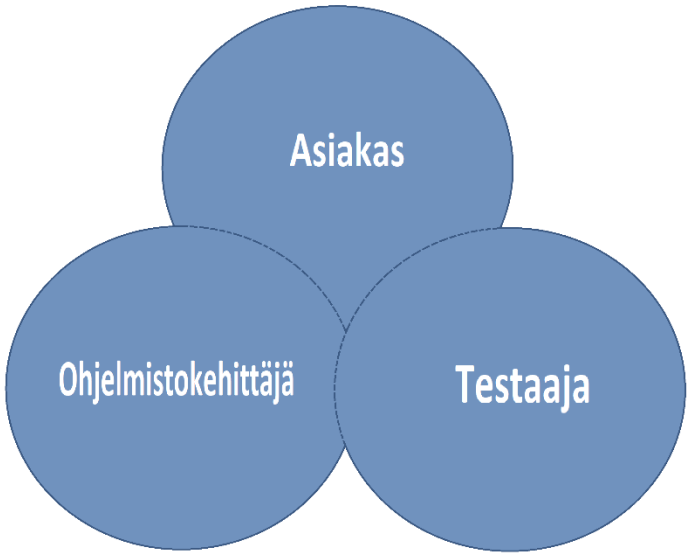
Toisin kuin TDD:ssä, tässä menetelmässä testiä luodessa kirjoitetaan ominaisuus, joka kuuluu käyttäjätarinaan, joka taas kuvastaa miten ominaisuuden tulisi toimia. Ominaisuuden tulee olla liiketoiminnan näkökulmasta luettava ja käyttää toimialuekohtaista kieltä. Tämän jälkeen ajetaan testi ja varmistetaan, että se ei mene läpi. Kun testin pätevyys on varmistettu, kehittäjä voi luoda ominaisuuden, joka läpäisee aikaisemmin luodun testitapauksen samaan tapaan kuin TDD:ssä. Pohjimmiltaan BDD on erikoistunut versio TDD:stä, joka keskittyy ohjelmistoyksiköiden käyttäytymismäärittelyihin. (Ye 2013)

4.3 Hyväksymistestivetoinen ohjelmistokehitys (ATDD)

ATDD, eli hyväksymistestivetoinen ohjelmistokehitys on ohjelmistokehitysmetodologia, jonka tarkoituksena on luoda yksiselitteinen määrittely asiakkaan vaatimuksista hyväksymistestien avulla (Sauvé & Neto 2008). ATDD:sta voidaan myös joskus käyttää termiä STDD (Story Test-Driven Development), mutta tässä työssä metodologiaan viitataan lyhenteellä ATDD (Sauvé & Neto 2008). ATDD pohjautuu joihinkin Lean- ja Ketterän kehityksen -periaatteisiin (Pugh 2011). Noudatettaessa ATDD:ta asiakkaan vaatimuksista muodostetaan korkean tason toiminnallisuuden testit ennen mitään toiminnallista toteutusta (Latorre 2014). Näitä hyväksymistestejä kutsutaan myös toiminnallisiksi testeiksi

(Sauvé & Neto 2008). Hyväksymistestit keskitetään liiketoimintalogiikkatasolle, jossa ne kuvastavat testattavia vaatimuksia ja määrittelyjä (Latorre 2014).

Testejä ovat yhdessä luomassa sovelluskehittäjä, testaaja, sekä asiakas. Asiakkaan roolissa voi toimia kuka tahansa henkilö, joka voi varmuudella edustaa asiakkaan vaatimuksia ja tarpeita. Tähän ryhmään viitataan myöhemmin termillä kolmikko (Triad). Tällä tavoin luotuja testejä voidaan myös luonnehtia ”asiakkaan kirjoittamina hyväksymistesteinä” (Latorre 2014). Asiakkaan toiminnallisuusmäärittely kertoo ”mitä” vaaditaan, kun taas kehittäjältä saadaan tieto ”kuinka” toiminnallisuus toteutetaan (Latorre 2014). Testit kirjoitetaan yksiselitteisellä kielellä, jota kaikki kolmikkoon kuuluvat ymmärtävät, riippumatta kenenkään teknisestä taustasta. Näin testien luomisen kautta muodostuu yhteinen termistö kaikkien sidosryhmien kesken, josta seuraa yksiselitteinen määrittely (Pugh 2011).



Kuva 4: ATDD:n kolmikko

Mitä kauemmin kellään sidosryhmään kuuluvalla menee aikaa lukea ja ymmärtää testien sisältö ja tarkoitus, sitä enemmän tuhlaantuu aikaa ja sitä vähemmän jää aikaa tuottavalle työlle. Mitä vähemmän aikaa jää tuottavalle työlle, sitä enemmän syntyy ulkopuolista painetta, joka lisää todennäköisyyttä, että testien kirjoittamiseen käytetään vähemmän aikaa. Tästä seuraa epäselvempiä testejä, jotka edelleen voimistivat tätä kierrettä. Tämän vuoksi testien ymmärrettävyys on ensisijaisen tärkeää. (Gärtner 2013)

Kattavamman ymmärrettävyytensä lisäksi hyväksymistestit eroavat alemman tason testeistä, kuten yksikkötesteistä, siten että ne eivät muutu kehityksen aikana, ellei vaati-

musta itseään muuteta. Jokaista uutta vaatimusta kohden tulisi luoda uusia hyväksymistestejä ja hyväksymistestien tulisi kattaa kokonaan järjestelmän toiminnalliset vaatimukset. (Pugh 2011)

Julkaisussaan Sauvé ja Neto (2008) ovat kirjanneet ylös lyhyen toimintakuvauksen ATDD:n käytöstä.

1. Kirjoita yksinkertaisin mahdollinen toteutus, joka läpäisee hyväksymistestit.
2. Mikäli kuvittelet, että hyväksymistestissä on virhe, lähetä testi asiakkaalle uudelleen arvioitavaksi.
3. Muuta toteutusta vain, jos se ei toteuta uutta hyväksymistestiä tai joudut refaktoroimaan koodia.
4. Jos hyväksymistestien läpäisemiseksi joudut luomaan rakenteita, jotka eivät ole suoraan saatavilla testattaviksi, kirjoita ensin yksikkötestit. Tee toteutus hyväksymistestien sijaan näiden pohjalta, käyttäen listan ohjeita.

ATDD mahdollistaa ylläpidettävämmän järjestelmän rakentamisen. Testeissä käytetty termistö muodostaa yhteisen kielen asiakkaan, kehittäjän ja testaajan kesken. Jos saman termistön käyttöä jatketaan myös toteutustasolla, koodin ja vaatimusten suhdetta toisiinsa on helpompi ymmärtää. (Pugh 2011)

Kun testit luodaan ennen toteutusta, on kirjoitetun koodin läpäistävä kyseiset testit. Tämän seurauksen koodista tulee testattavampaa. Testidokumentaatio toimii kuvauksena järjestelmän toiminnallisuudesta. Testejä voi pitää ajettavana määrittelynä, joka ei vanhene, ellei määrittely muutu. Koska kaikille ominaisuuksille kirjoitetaan testitapaukset, mikään järjestelmän osa-alue ei jää testaamatta. Testivetoisella menetelmällä saadaankin aikaa parempi kattavuus järjestelmän testattavuudessa. (Pugh 2011)

Kun testien luonnissa on mukana vähintään kolmen sidosryhmän edustajat, jokaista kirjoitettua hyväksymistestiä on tarkasteltu kolmesta eri näkökulmasta ennen kuin se on valmis. Kirjoitusprosessista syntyvä keskustelu nostaa esiin kysymyksiä, jotka olisivat muuten jääneet löytymättä näin aikaisessa vaiheessa. Pelkkien testien kirjoittaminen voi paljastaa mahdollisia ongelmakohtia ja virheitä määrittelystä tai toteutuksesta, jos sitä olisi alettu työstämään ennen testejä. (Pugh 2011)

Yhteistyön onnistumisen kannalta jokaisen kolmikkoon kuuluvan tulee ymmärtää, mikä on kirjoitettavien testien tarkoitus ja hyöty. Jos vain yksi jäsenistä pitää prosessia oleellisena, vaihdos käyttämään ATDD:ta voi olla haastavaa. Useissa tapauksissa asiakkaat voivat kokea, että heillä ei ole tarpeeksi yksityiskohtaista tietoa testien kirjoittamiseen. Vaihtoehtoisesti voi olla, että asiakkaalla ei ole tarpeeksi aikaa tai kokemusta kirjoittaa

testejä ATDD:n vaatimalla tarkkuudella. Tällöin henkilöt (asiakkaan puolella), joilla on riittävä määrä tietoa ja aikaa, tulee tunnistaa ja tuoda yhteistyön piiriin. Vaikka asiakas kuuluu kolmikkoon, se ei välttämättä tarkoita, että hänen tulee olla paikalla. Suunnittelussa hänet voi korvata edustaja, joka pystyy toimimaan kolmikossa asiakkaan tarpeiden asiantuntijana. (Pugh 2011)

Joissakin tapauksissa voi olla, että kommunikaatio asiakkaan kanssa on olematonta ja testaajaa ei ole saatavilla. Tällöin kehittäjällä ei ole saatavilla hyväksymistestejä, eikä myöskään vaatimuksia. Jos ohjelmistokehitys on välttämätöntä aloittaa näistä olosuhteista huolimatta, tulisi kehittäjän luoda itse hyväksymistestit parhaan tietämyksensä pohjalta. (Pugh 2011)

Huolimatta ATDD:n mahdollisista hyödyistä uuden toimintatavan käyttöönotto voi aiheuttaa sekaannusta ja vastarintaa yrityksen sisällä. Tämä kumpuaa muutoksista työntekijöiden rooleissa. Asiakas joutuu osallistumaan enemmän antamalla tarkennuksia ja esimerkkejä, testaajat joutuvat kirjoittamaan testit järjestelmälle tai toiminnallisuudelle, jota ei ole vielä tehty. Tämän lisäksi kehittäjät joutuvat testaamaan enemmän. (Pugh 2011)

Hyväksymistestit eivät ole ongelmattomia, etenkään automatisoidut testit. Testit vaativat resursseja niiden ylläpitoon ja muuttuvat määrittelyt rikkovat aikaisemmat testit. Jos rikkuneiden testien määrä on korkea, voi olla, että niiden korjaamisille ei löydy aikaa ennen seuraavaa julkaisua. Jos testejä ei korjata, niiden avulla löydettyjä virheitä voidaan alkaa jättää huomiotta. Vaihtoehtoisesti rikkinäiset testit voivat antaa ymmärtää, että rikkinäiset testit ovat hyväksyttäviä. (Pugh 2011)

5. HYVÄKSYMISTESTIVETOINEN TYÖSKENTELY

Tässä luvussa käydään läpi, miten ATDD:ta lähdettiin käyttöönottamaan Fastemsilla ja minkälaisia hyötyjä, ongelmia ja muutoksia sen myötä nousi esille. Kappale keskittyy yksittäisen projektitiimin tekemiseen ja huomioihin. Esimerkit kappaleeseen on otettu projektille toteutetusta toiminnanohjausjärjestelmäliitynnästä. Lukuihin 5.4 ja 5.5 kootut havainnot hyödyistä ja haasteista pohjautuvat projektitiimin, testaajan ja toimitusomistajan kanssa käytyihin keskusteluihin.

5.1 Lähtökoulutus

Fastems palkkasi ulkopuolisen konsulttiyrityksen pitämään koulutuksen ja toimimaan tukena alkupään ATDD-tapaamisissa. Sama koulutus järjestettiin kaksi kertaa eri ryhmille, joihin molempiin osallistui vajaa kaksikymmentä henkeä. Osallistujista suuri osa oli ohjelmistokehityksen puolelta ja useammasta eri projektitiimistä. Mukana oli myös henkilöitä Vanilla/Platform- ja QA-tiimeistä, sekä DO:ita. Koulutuksessa käytiin alkuun teoriaa testauksesta yleisellä tasolla, jonka jälkeen tarkasteltiin ATDD:ta tarkemmin. Koulutuksen pitäjälle käytiin myös läpi Fastemsin nykyinen tapa toimia ohjelmistokehityksessä.

Suuri osa koulutuksesta käytettiin käytännönläheiseen harjoitukseen, jossa kirjoitettiin testitapauksia ominaisuudelle, joka Vanilla/Platform-tiimin oli tarkoitus toteuttaa ja tuotteistaa. Valmis tuote oli myöhemmin tarkoitus ottaa osaksi asiakasprojektia. Kyseisen projektin DO esitteli toteutettavan ominaisuuden toiminnallisuuden ja vaatimukset. Testien kirjoittaminen ei edennyt hyvin, vaan keskustelu jäi jumiin teknisiin yksityiskohtiin. Osaksi tämä johtui siitä, että paikalla oli paljon ohjelmistokehittäjiä ja ominaisuus oli toiminnallisuudeltaan monimutkainen. Oli myös selvää, että toiminnallisuudessa oli avoimia kysymyksiä, jotka edelleen hankaloittivat mielekkäiden testien kirjoittamista. Koulutus antoi kuitenkin perusteet jatkaa ATDD:n harjoittelua tiimien sisällä.

5.2 Toiminnanohjausjärjestelmäliityntä

Tarkasteltavassa projektissa toimitettiin yhdysvaltalaiselle asiakkaalle tehdasautomaatioratkaisu, joka integroi asiakkaan toiminnanohjausjärjestelmän, NC-ohjelmat (Numerical Control), työkaluhallinnan, työstökoneet ja muut kappaleen valmistukseen liittyvät laitteet osaksi FMS-järjestelmää (Flexible Manufacturing System), jonka sovelluskerroksesta käytetään nimeä MMS. Järjestelmässä kappaleet liikkuvat robottien avulla laitteelta toiselle MMS:n ohjauslogiikan määrittelemänä. Ohjauslogiikka pohjaa toimintansa

olemassa oleviin tilauksiin ja niiden prioriteetteihin, sekä käytettävissä oleviin resursseihin, kuten raakamateriaaleihin, työkaluihin, NC-ohjelmiin ja saatavilla oleviin laitteisiin.

Tarkasteltavan projektitiimin osaksi tuli toteuttaa liityntä asiakkaan toiminnanohjausjärjestelmän ja Fastemsin toimittaman tehdasautomaatiojärjestelmän välille (MMS). Liityntänän avulla asiakkaan tulisi pystyä hallitsemaan oman järjestelmänsä kautta MMS-järjestelmän tuotantoa. Viestiliikenne olisi kaksisuuntaista ja MMS järjestelmä ilmoittaisi tuotantoon liittyvistä muutoksista ja aikatauluista asiakkaan hallitsemaan järjestelmään. Liitynnän avulla asiakas pystyisi mm. luomaan uusia kappaleita, työkaluja, tilauksia ja hallitsemaan näihin liittyviä tietoja. MMS puolestaan ilmoittaisi takaisin kappaleiden tuotantovaiheista ja suunnitellusta tuotannosta.

Kuva 5: MMS – Manufacturing Management System (Fastems 2021)

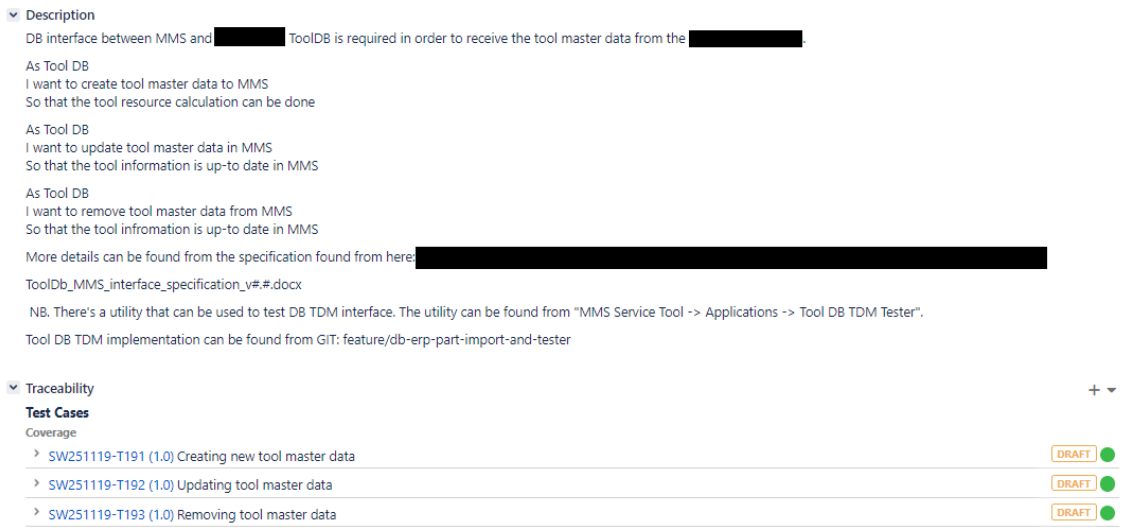
Liityntä oli aluksi tarkoitus toteuttaa XML-tiedostojen (Extensible Markup Language) avulla, jossa viestiliikenne MMS:n ja asiakkaan toiminnanohjausjärjestelmän välillä tapahtuisi jaettujen hakemistojen kautta. Asiakkaan pyynnöstä viestiliikenne siirrettiin kulkemaan yhteisen tietokannan kautta. Koska tiedostopohjainen liityntä on yksi Fastemsin tarjoamista standarditoteutuksista, ja jotta toimintaa päästiin esittelemään nopeammin asiakkaalle, pohjatoteutus päätettiin tehdä XML-pohjaisena. Tämän lisäksi katsottiin, että suurta osaa XML-toteutukseen tehtävistä muokkauksista voitaisiin käyttää pohjana tietokantapohjaiselle toteutukselle. ATDD:n osalta tämä aiheutti lievää lisätyötä, sillä kaikki XML-toteutukseen tehdyt hyväksymistestit jouduttiin kirjoittamaan uudelleen vastaamaan tietokantapohjaista toteutusta.

Haasteeksi liitynnän toteutuksessa ilmeni perustiedon (valmistettavan kappaleen yksityiskohtainen kulku järjestelmässä raakamateriaalista valmiiksi tuotteeksi) tuonti liitynnän yli. Haasteen aiheuttivat monimutkaiset robottiliikkeet, liikkeiden parametrit ja näiden puuttuminen asiakkaan päästä. Ongelmia lisäsi myös uusi monimutkainen konetyyppi, joka tuli ottaa huomioon perustiedon tuonnissa, mutta lopullinen toteutus kyseiselle laitteelle valmistui vasta projektin loppupuolella. Liitynnän laajuus kasvoi projektin aikana sisällyttäen nyt myös työkalutietojen siirron, mutta tästä ei aiheutunut merkittäviä lisähaasteita, sillä rakenteet liitynnälle olivat jo tehtynä.

5.3 Toimintakäytännöt

Harjoitussyistä testien kirjoittamiseen osallistui testaajan ja DO:n lisäksi koko projektitiimin ohjelmistokehittäjät. Muutamassa alkupään tapaamisista koulutuksista vastannut konsultti oli myös mukana tukemassa tekemistä. Jatkossa tarkoitus on pitäytyä kolmikossa (ohjelmistokehittäjä, testaaja, asiakas/toimitusomistaja), sillä testitapaamisiin kulutettu aika kertaantuu jokaisesta ylimääräisestä osallistujasta.

Jira valikoitui luonnollisesti testien kirjoitusalustaksi, sillä sitä oli käytetty jo ennestään muun työn hallintaan Fastemsilla. Jiraan ladattava Zephyr-lisäosa helpotti testien hallintaa ja mahdollisti niiden viemisen ominaisuustiedostoiksi (feature file) alustan ulkopuolelle. Tiedostotyyppi pitää sisällään yksittäisen testitapauksen testiskriptin, joka voidaan Nunitin ja Specflow'n avulla tuoda osaksi C#-projektia.



Kuva 6: Työkalun hallintaan liittyvä ominaisuus testeineen Jirassa

Projektin alussa ERP-liityntää (Enterprise Resource Planning) koskevia ominaisuuksia ei ollut valmisteltu testien kirjoittamista varten. Osa ominaisuuksista oli kuitenkin valmiiksi

luotuna Jiraan. Testisessioissa aloitettiin tämän vuoksi ominaisuuksien kuvauksen ja siihen kuuluvien käyttäjätarinoiden kirjoittamisesta, sekä puuttuvien ominaisuuksien luomisesta. Monella tapaa ensimmäiset ATDD-tapaamiset olivat yhdistelmä käyttäjätarinakartoitus- ja hyväksymistestikirjoitussessioita. Monesti keskustelu käyttäjätarinoista johti kysymyksiin, joihin toimitusomistajan läsnäolosta huolimatta ei heti ollut saatavilla vastauksia. Näissä tilanteissa havaittiin hyväksi käytännöksi kirjoittaa avoimet kysymykset osaksi ominaisuuden kuvausta. Käsitellyt kysymykset tuli yliviivata ja kirjoittaa vastaus näiden perään. Kysymykset haluttiin jättää muistuttamaan jo käydystä pohdinnasta, jotta samoihin asioihin ei tarvitsisi palata uudelleen. Kuvassa 7 on kuvattuna ERP-liitynnän toteuttamaan työkaluhallintaan liittyvä ominaisuus ja kolme sen hyväksymistestiä työkalutiedon luonnista, päivittämisestä ja poistosta.

BDD - Gherkin Script

```
Given Tool data is written into shared database by customer
  And action is create or update
  And tool does not exist in MMS
  And data is <validity>
When Poller detects unprocessed row in shared database
Then MMS updates row processing result to <result_1>
  And then <result_2>

Examples:
| valididty | result_1 | result_2 |
| valid | OK | tool master data is created to MMS |
| invalid | NOK | error reason is written and tool master data is not created |
```

Kuva 7: Testiskripti työkalun luomisesta EPR:in kautta

Ominaisuuden kuvauksen ja siitä käydyn keskustelun pohjalta sille voitiin aloittaa hyväksymistestien luominen. Testiä luodessa se nimettiin, sen tyypiksi annettiin hyväksymistesti ja sen sijainniksi asetettiin projektikohtainen ja ominaisuuskohtainen hakemisto. Nimeämiskäytäntönä testit nimettiin kuvaamaan ominaisuuden toimintaa ja mikäli mahdollista, loppukäyttäjän näkökulmasta. Lisäksi sovittiin, että testitapauksiin, jotka kuvaavat virheellistä toiminnallisuutta, lisättiin jälkiliite (NOK). Näin saatettiin luoda kaksi testitapausta vastaavin nimin, joista toinen kuvasi ominaisuuden onnistunutta ja toinen epäonnistunutta toiminnallisuutta.

Testiskriptien kirjoittaminen tiimin sisällä oli aluksi todella hidasta. Tämä saattoi olla seurausta samoista asioista, jotka hidastivat testien kirjoittamista lähtökoulutuksessa. Liian monta ohjelmistokehittäjää paikalla ajoi keskustelun teknisiin yksityiskohtiin ja pohdinta

siirtyi toiminnallisuudesta toteutuspuolen ongelmiin. Loppukäyttäjäperspektiivin palauttaminen keskusteluun joudutti kirjoittamisprosessia, huomion kääntyessä takaisin itse toiminnallisuuteen.

Testejä kirjoittaessa havaittiin, että alkuasetelman (Given) keksiminen testille oli monesti haastavaa. Hankalissa tapauksissa, jotka selvästi seisauttivat tekemisen, siirryttiin kirjoittamaan testiä sen loppuvaiheesta (Then) ja tapahtumasta (When). Kun muut osuudet testistä olivat kirjoitettu, alkuvaihe testille oli huomattavasti helpompi keksiä.

Havaittiin, että testien kieliasun pitäminen yhdenmukaisena ei vain helpota luettavuutta, vaan myös testien automatisointia. Tämä korostuu erityisesti testin alkuasetelman suhteen, joka voi jakaa yhtäläisyyksiä monen muun testitapauksen kanssa. Mikäli testitapauksissa on käytetty keskenään samanlaisia testiaskeleita, testien automatisoija pystyy helpommin uudelleen käyttämään jo valmiiksi toteutettuja funktioita.

Testiskriptit kirjoitettiin käyttämällä Gherkin-syntaksia, jonka rakenteeseen ja avainsanoihin perehdyttiin aikaisemmin tämän työn 3. kappaleessa. Vaikka Zephyr tukee But- ja *-avainsanojen käyttöä, päätettiin näitä olla käyttämättä, jotta kieliasu pysyisi mahdollisimman yhdenmukaisena kirjoittajasta riippumatta. Kaikki But- tai * -avainsanat voidaan korvata And-avainsanalla, sillä ne eroavat toisistaan vain semanttisesti.

Tapauksissa, joissa useampia tapahtumia (When) haluttiin yhdistää, havaittiin, että oli joko syytä kirjoittaa useampi testitapaus tai vaihtoehtoisesti tapahtuman voisi kirjoittaa osaksi testitapauksen aloitustilaa. Aikaisemmassa lähtökoulutuksessa oli myös suositeltu, että kahta useampaa When-avainsanaa ei kannata käyttää yhdessä testitapauksessa. Näistä syistä päätettiin välttää useamman tapahtuman yhdistämistä, ellei tapahtumat todella ole rinnakkaisia.

Testitapauksia ei alkuun parametrisoitu ja tämä johti useamman hyväksymistestin kirjoittamiseen, kuin todellisuudessa oli tarvetta. Myös tästä syystä osa testeistä oli muutamaa poikkeavaa sanaa lukuun ottamatta lähestulkoon identtisiä. Hyödyntämällä Gherkinin Examples-avainsanaa, yksittäiseen hyväksymistestiin pystyttiin kirjoittamaan useampi testitapaus, kun osa sen testiaskeleista parametrisoitiin. Kenenkään osallistujan mielestä parametrisointi ei vaikeuttanut testitapauksen ymmärrettävyyttä. Vaarana tässä menettelyssä on kuitenkin liikaparametrisointi, joka voi hankaloittaa testin luettavuutta. Jos parametrisoituja arvoja tai testivariaatioita (yksittäinen Examples-esimerkkirivi) on paljon, seurauksena on sekava testi. Sovittiin, että parametrisoinnin tarkoituksena ei ole luoda kaiken kattavaa testausta, vaan tuoda esiin ominaisuuden eri toiminnallisuuksia. Esimerkiksi ei ole mielekästä kirjoittaa testitapauksia kaikille arvoille, jotka synnyttävät

virhetilanteen, vaan vain niille, joiden virheiden käsittely poikkeaa asiakkaan näkökulmasta toisistaan.

Gherkinin avainsanat Feature, Scenario ja Background eivät ole käytössä, sillä Zehpyr ei ainakaan toistaiseksi tue näitä avainsanoja. Zephyrin verkkosivuilla on lista käyttäjien ehdotuksista ja siellä puuttuvien avainsanojen tuontia osaksi sovellusta on jo ehdotettu. Tulevaisuudessa puuttuvat avainsanat voivat siis olla mahdollisia käyttää. Feature-avainsanalla voidaan antaa kuvaus ominaisuudesta, jota ollaan testaamassa. Tämä tieto on jo saatavilla ominaisuudesta itsestään, johon testitapaus on linkitetty. Tästä syystä avainsanasta olisi hyötyä lähinnä testien automatisoinnissa, sillä vain testiskriptin sisältö säilyy, kun se siirretään Jirasta C#-projektiin.

Scenario-avainsanalla voidaan määrittää samaan testiskriptiin useampia testitapauksia, skenaarioita. Skenaario on yksittäisen testitapauksen nimi tai kuvaus testiskriptissä, joka erottaa sen muista testitapauksista. Scenario olisi hyödyllinen samasta syystä kuin Examples. Se vähentää toistoa vähentämättä luettavuutta. Background-avainsanalla voidaan määrittää kaikille testiskriptin skenaarioille yhteiset askeleet, jotka suoritetaan ennen jokaisen skenaarion suorittamista. Tällä tapaa voidaan vähentää toistoa testiskriptin sisällä, mikäli skripti sisältää useampia testitapauksia/skenaarioita.

Aikaisemmin esitetyssä kuvassa 7 on esitettynä testiskripti työkalutiedon luomiselle tietokantapohjaisen ERP-liitynnän kautta. Esimerkissä testin suorittamiseen vaadittu tila on:

- Työkalutieto on kirjoitettuna jaettuun tietokantaan asiakkaan toimesta
- Kirjoitetun viestin toiminnoksi on merkattu luonti tai päivitys
- Työkalutiedossa mainittua työkalua ei löydy MMS-järjestelmästä
- Työkalutiedossa ei ole/on puutteita tai viallista tietoa

Esimerkistä on nähtävissä abstraktiotaso, jolle testit pyritään kirjoittamaan. Testiskriptissä itsessään ei ole määriteltynä, mikä on validia dataa ja mikä ei. Testissä todetaan, että data on joko validia tai ei. Testi määrittää vain toiminnallisuuden, joka on seurausta esimerkiksi datan validiudesta. Kuvatussa testitapauksessa neljännellä rivillä olisi voitu käyttää But-avainsanaa, koska yksi sen mahdollisista arvoista on kielteinen (invalid). Toisaalta sen toinen arvo on myönteinen (valid), jolloin And-avainsana on sopivampi. Kun aina käytetään And-avainsanaa, kyseisestä semantiikasta ei tarvitse välittää.

Esimerkissä testin tapahtumana (When) on:

- Järjestelmän ajastettu valvontamekanismi huomaa uuden rivin jaetussa tietokannassa

Testit pyritään kirjoittamaan siten, että loppukäyttäjä on niissä toimijana. Esimerkkitestin tapauksessa toimijana on käytetty järjestelmää. Esimerkin tapauksessa viestin lähettämistä olisi voinut käyttää tapahtumana, mutta silloin alkutilan määrittäminen viestiin liittyen olisi ollut hankalampaa ja tapahtumaan olisi joka tapauksessa jouduttu laittamaan myös järjestelmän havainto lähetetystä viestistä.

Esimerkissä lopputulemana (Then) on:

- MMS päivittää viestirivin prosessoinnin tulokseksi ”OK”/”NOK”
- Työkalutiedot luodaan MMS:ään/virheen syy kirjoitetaan ja työkalutietoja ei luoda

Testissä nähdään kaksi lopputulemaa, jotka molemmat riippuvat aikaisemmasta muuttujasta. Rivillä 7 And-avainsanan jälkeen on jouduttu kirjoittamaan ”then”, sillä Zephyr ei osaa tulkita parametrisointia välittömästi avainsanan jälkeen.

Kun ominaisuudelle on luotu kaikki sen hyväksymistestit, sille voidaan kirjoittaa tehtävät (Task). Tehtävien kirjoitus on ohjelmistokehittäjien vastuulla ja näihin voidaan liittää teknisiä yksityiskohtia toteutuksen suhteen. Vaikka kehittäjä tekee toteutuksensa siten, että se vastaa hyväksymistestiä, hänen vastuullaan ei ole ajaa testiä. Kun ominaisuus tulee valmiiksi kehitystyön osalta, se siirretään laadunvarmistuksen tarkistettavaksi, joka sitten ajaa hyväksymistestit. Tämän projektin tapauksessa kaikki testit ajettiin manuaalisesti.

5.4 Havainnot

Hyväksymistestivetoisen ohjelmistokehityksen harjoittelu käynnistyi hyvin projektitiimin osalta ja kaikki osalliset olivat valmiita kokeilemaan uutta menetelmää. Testien kirjoittaminen ja toiminnallisuudesta keskusteleminen nosti esiin kysymyksiä ja epäselvyyksiä, joiden havaitseminen olisi ilman uutta prosessia siirtynyt projektin osalta myöhemmäksi. Epäselvyyksien havaitseminen edellytti, että kaikki osallistuivat keskusteluun ja ilmoittivat epäselvyyksistä tai ongelmista niitä huomatessaan. Prosessin tarkoitus on antaa kaikille osallisille järjestelmän toiminnallisuudesta yksiselitteinen kuva ja tämä ei tapahdu, mikäli osallistujat eivät kysy, jos jokin on heille epäselvää. Harjoittelu osoitti, että yhteisymmärrys toiminnallisuudesta saavutetaan paremmin hyväksymistestivetoista prosessia hyödyntämällä kuin itsenäisellä toiminnallisen kuvauksen tai määrittelyn läpikäynnillä.

Ohjelmointitehtävien luominen ominaisuudelle helpottui kirjoitettujen hyväksymistestien avulla. Testeistä pystyi varmistamaan vaaditun toiminnallisuuden ohjelmointitehtäviä

varten. Vastaavasti hyväksymistestit helpottivat varmistumaan jälkikäteen siitä, että toteutus vastasi vaatimuksia.

Koska asiakasta ei otettu hyväksymistestivetoiseen prosessiin mukaan, joutui toimitusomistaja Fastemsilta toimimaan asiantuntijana asiakkaan vaatimusten ja prosessin suhteen. Toimitusomistaja oli avainroolissa ja vastuussa vastausten selvittämisestä avoimiin kysymyksiin. Toimitusomistajan rooli oli hankala, sillä itse asiakaskaan ei olisi välttämättä tiennyt projektin alussa, miten heidän prosessinsa tulisi toimia Fastems-integraation kanssa. Projektissa tieto kulki välikäden kautta, eikä hyväksymistestejä voitu realistisesti hyväksyttää asiakkaalla, sillä he eivät olleet mukana niitä luomassa. Havaittiin, että asiakkaan ymmärtämistä ja vastausten saamista helpottaisi merkittävästi, mikäli asiakas olisi osa hyväksymistestivetoista prosessia.

Asiakkaan puuttumisen seurauksena myös testien kirjoittamiseen kului ylimääräistä aikaa, kun keskeneräisiin testeihin jouduttiin palaamaan jälkikäteen. Testien kirjoittamiseen kului ilman asiakkaan puuttumistakin runsaasti aikaa. Harjoittelun seurauksena testien kirjoittaminen kuitenkin nopeutui ja testien kuvaavuus parantui.

Keskusteluissa selvisi, että asiakkaan integrointia prosessiin tulee hankaloittamaan Fastemsin nykyinen hinnoittelumalli. Ohjelmistomuutoksia myydään kiinteällä hinnalla, jolloin toimituksen laajuus määräytyy ennen kuin asiakas pääsee määrittelemään järjestelmän toiminnallisuutta kolmikon kesken. Toisaalta, jos asiakas olisi osa ATDD-prosessia myynnin jälkeen, asiakkaalla olisi mahdollisuus vaikuttaa myytyyn toiminnallisuuteen ilman vaikutusta alkuperäiseen myyntihintaan. Kumpikaan näistä skenaarioista ei ole otollinen ja tästä syystä myös myynti tulisi integroida osaksi ATDD-prosessia.

Projektin hyväksymistestejä ei automatisoitu, vaan prioriteettina olivat MMS:n pohjatoteutuksen ohjelmisto-optioiden ja rajapintojen testauksen automatisointi. Keskusteluissa selvisi, että vaikka hyväksymistesteistä hyödytään jo pelkästään niiden määrittelyarvon vuoksi, niin osa testien hyödyistä häviää, kun niitä ei automatisoida. Projektissa havaittiin vikoja, jotka olisi voitu välttää automatisoidulla regressiotestauksella.

Yhdeksi prosessin hyödyksi havaittiin, että toiminnallisuuteen liittyvä tieto oli keskitetympää kuin aikaisemmin. Projektin aikana tuli aikaisempaa vähemmän tilanteita vastaan, jossa tarkempaa kuvausta joutui etsimään ominaisuuden ulkopuolelta toteutuksen loppuun saattamiseksi. ATDD-mallin rinnalla käytettiin kuitenkin myös järjestelmän toiminnallista kuvausta. Tästä seurasi, että vaatimuksissa tapahtuvat muutokset aiheuttivat kaksinkertaisen työn, kun hyväksymistestit ja toiminnallisen kuvaus molemmat jouduttiin päivittämään muutosten osalta. Työ olisi ollut kolminkertainen, mikäli projektin hyväksymistesteille olisi luotu automatisoidut testit.

6. TAPAUSTUTKIMUKSET

Tässä luvussa käsitellään kahta oikean elämän tapausta, joissa hyväksymistestivetoista menetelmää on käytetty osana ohjelmistokehitystä. Kuvattujen tapaustutkimusten on tarkoitus antaa vertailukohtia siihen, miten toiset ovat onnistuneet hyödyntämään hyväksymistestivetoisia menetelmiä ja mitkä asiat ovat tuottaneet haasteita. Luvun lopussa tapauksia verrataan Fastemsin tapaan työskennellä ATDD:ta käyttäen.

6.1 Ravintolaketjun applikaatioiden keskittäminen verkkoon

Artikkelissaan Latorre (2014) esittelee projektin, jossa aikataulupaineiden vuoksi aikaisempi yritys erotettiin tehtävästä ja artikkelin yritys palkattiin erotetun tilalle. Tarkoituksena oli valmistaa ravintolaketjulle verkkosovellus, joka keskittäisi yksittäisten ketjun ravintoloiden käyttämät sovellukset yhteen paikkaan. Aikataulu oli tiukka ja vajaiden määrittelyiden vuoksi yritys päätti kokeilla UTDD:n ja ATDD:n yhdistelmää strategianaan näiden menetelmien tunnetuista riskeistä huolimatta. Strategialla pyrittiin saavuttamaan seuraavia asioita:

- Liiketoimintavaatimusten keruu ja selkeä ymmärrys niistä
- Alustavan järjestelmäluonnoksen nopeampi toteutus
- Vikojen havaitseminen aikaisessa vaiheessa

Avainroolissa projektin onnistumisen kannalta toimi asiakkaan puolelta nimetty henkilö, joka teki täysipäiväisesti yhteistyötä määrittely- ja validointitehtäviin liittyen. Henkilöllä oli teknistä osaamista ja hän tunsi asiakkaan liiketoiminnan. (Latorre 2014)

Ongelmaksi nousi aikaisemman kokemuksen puute testaa-ensin-kehityksistä. Tiimin jäsenillä oli useamman vuoden kokemus tavanomaisista testaa-viimeiseksi menetelmistä. Kehittäjille järjestettiin yhden viikon mittainen käytännönläheinen koulutus aiheesta. Osallistujille vaikeinta oli sisäistää tarve kirjoittaa yksikkötestit etukäteen ja lyhyiden inkrementaalisten syklien tärkeys kirjoitetun testin ja tätä vastaavan koodin välillä. Pieneksi ongelmaksi muodostui myös uusien testiautomaatiotyökalujen opettelu. Tämä korjattiin esimerkkien ja tiimiläisten avustuksella. (Latorre 2014)

Tiimissä määriteltiin, että vähintään yksi hyväksymistesti tulee olla kirjoitettuna ominaisuudelle ennen kuin sitä voidaan alkaa toteuttaa. Latorre (2014) mainitsee, että ATDD:n toteuttamistapoja on monia ja he muokkasivat ATDD:n omiin tarpeisiinsa sopivaksi. Tes-

tiautomaatiotyökaluiksi harkittiin seuraavia: FitNess, Concordion ja Cucumber. Tiimi päätyi käyttämään Concordion:ia, koska sitä pidettiin intuitiivisena ja helppona oppia. (Latorre 2014)

ATDD auttoi määrittelemään projektin rajauksen ja vaatimusten validoinnin määrittelyvaiheessa. Korkean tason toiminnalliset testit helpottivat tuoteomistajaa ymmärtämään ja validoimaan vaatimuksia selventäen epäselviä kohtia. Concordion:in avulla asiakas pystyi lukemaan järjestelmän koko toiminnallisuuden HTML-tiedostosta ja validoimaan sen esimerkkien avulla, jotka olivat suoraan linkitettyjä Junit-testeihin. Tämä vähensi väärinkäsityksiä ja väärinymmärryksiä kehityksen aikana. Näiden etujen saavuttamiseksi asiakkaan läheinen yhteistyö oli avainasemassa ja ilman tätä ATDD:n kaikista ominaisuuksista olisi ollut vaikea hyötyä. Latorre (2014) lisää vielä, että ATDD toimi projektissa hyvin, koska kaikilla vaatimuksilla oli määritettävissä olevat seuraukset. Projekteissa, joissa tämä ei pidä paikkaansa, ATDD ei välttämättä toimi yhtä hyvin.

Yhteenvetona ATDD:sta Latorre (2014) listaa:

- ATDD vaatii kattavaa yhteistyötä asiakkaalta.
- ATDD auttaa kirjaamaan ja validoimaan liiketoimintavaatimukset asiakkaan kanssa määrittelyvaiheessa.
- ATDD auttaa ohjelmistokehittäjiä ymmärtämään vaatimuksia paremmin.
- Ajettavat hyväksymistestit helpottavat järjestelmän todentamista.

6.2 Kliinisen päätöksen tukityökalu (CDS)

Artikkelissaan Mujeeb ja kumppanit (2018) tarkastelee hyväksymistestivetoisen kehityksen ja regressiotestauksen hyödyntämistä kliinisen päätöksen tukityökalun (engl. Clicinical Decision Support, CDS) kehityksessä. Menetelmiä käyttämällä pyrittiin kasvattamaan ohjelmiston laatua, esittämään vaatimukset hyväksymistesteinä, regressiotestaamaan sekä luomaan elävän suunnitteludokumentoinnin (Living Documentation).

Avustajatyökaluun toteutettiin ominaisuus, jonka avulla hoitajalle voidaan välittää varoitus, mikäli tämä on antamassa oraalilääkitystä potilaalle, jolla epäillään akuuttia halvausta. Halvauksesta kärsivällä voi olla heikentynyt nielaisumekanismi, joka voi johtaa lääkityksen joutumisen potilaan keuhkoihin. (Mujeeb et al. 2018)

Keskeisenä laatukriteerinä ominaisuudelle pidettiin annettujen hälytysten paikkansapitävyyttä. Väärät hälytykset voivat johtaa niin sanottuun ”hälytysväsymykseen” (Alert Fatigue), jossa hoitava henkilö saattaa turtua CDS:n ilmoittamiin varoituksiin ja jättää ne

huomiotta. Testien automatisointiin käytettiin FitNesseä ja dbFit-laajennusta, jolla haettiin dataa sähköisestä terveyskirjastosta (engl. Electronic Health Records, EHR). (Mujeeb et al. 2018)

Kokeilun tuloksena havaittiin, että hyväksymistestien läpikäynti sairaalahenkilökunnan kanssa mahdollisti heidän tietotaitonsa hyödyntämisen alustavassa CDS:n suunnittelussa, sekä myöhemmin ohjelmistokehityksen aikana lyhytsyklisissä iteraatioissa. Automatisoidun testivetoisen ohjelmistokehityksen havaittiin olevan yksi vaikuttava tekijä luotettavan EHR:n saavuttamisessa. (Mujeeb et al. 2018)

6.3 Tapaustutkimusten arviointi

Molemmissa tapaustutkimuksissa tiivis yhteistyö asiakkaan kanssa nousee keskeiseen rooliin projektin onnistumisen kannalta. Fastemsilla asiakkaan roolista ATDD-prosessissa vastaa DO. Kuten tämän työn 5. luvussa ja Latorren (2014) artikkelissa mainitaan, ATDD:sta on hankalaa saada kaikkia hyötyjä irti ilman läheistä yhteistyötä asiakkaan kanssa.

Latorren (2014) artikkelin projektissa työskenteleville tarjottiin viikon mittainen koulutus, joka oli tässä tapauksessa riittävä määrä kehittäjille omaksua ATDD osaksi heidän työskentelyprosessiaan ilman suurempia ongelmia. Fastemsilla on järjestetty kaksi noin päivän mittaista koulutusta, joihin kaikilla ohjelmistokehittäjillä ei ollut mahdollisuutta osallistua. Tämän seurauksena monelta voi puuttua oleellinen tieto hyväksymistestivetoisessa ohjelmistokehityksessä työskentelystä ja koulutus voisi tarjota tähän ongelmaan ratkaisun.

Molemmissa tapaustutkimuksissa testien automatisointi nousee oleelliseksi tekijäksi projektin onnistumisessa. Ensimmäisessä tapaustutkimuksessa asiakas pystyi ajamaan hyväksymistestejä ja varmentamaan järjestelmän toiminnallisuutta itsenäisesti. Mujeebin (2016) artikkelissa automatisoiduilla testeillä nähtiin olevan yhteys luotettavamman ohjelmiston toteuttamisessa. Fastemsilla projektikohtaista automatisoitua testausta tehdään pääsoin MMS:n pohjatoteutuksen osalta. Projektiräätälöinteihin liittyviä hyväksymistestejä automatisoidaan rajoitetusti, sillä niiden hyötysuhde katsotaan monessa tapauksessa pieneksi.

7. KYSELYTULOKSET

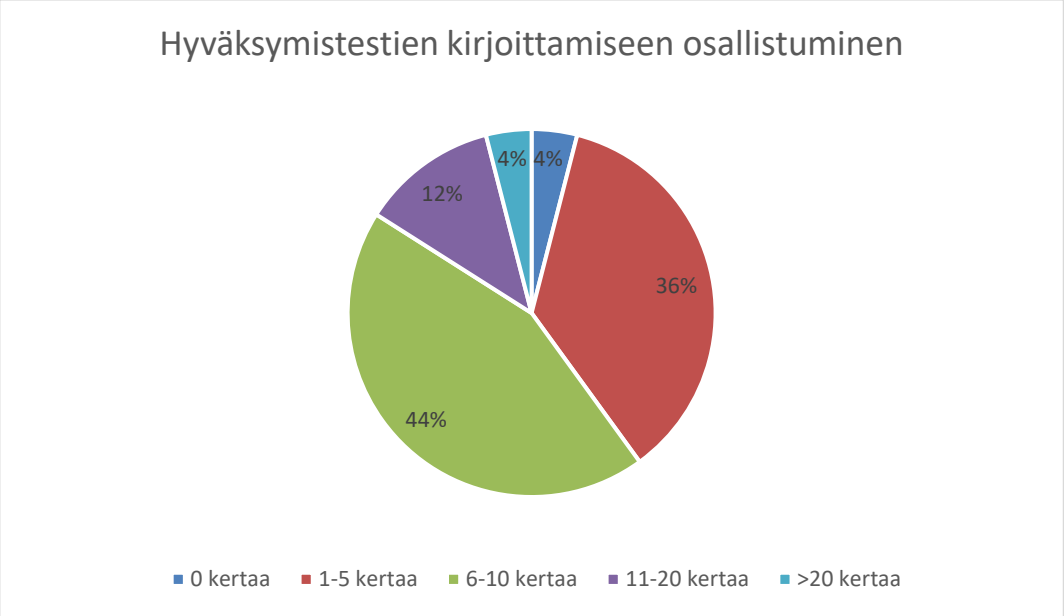
Kysely (liite A) jaettiin neljään osioon. Ensimmäisessä osiossa kartoitettiin vastaajan taustatietoja. Seuraavassa osiossa vastattiin väittämiin ATDD-prosessin hyödyistä ja haitoista. Väittämissä 6–8 oli eroavaisuuksia riippuen vastaajan roolista, jonka tämä oli valinnut ensimmäisessä osassa. Tässä osiossa kysymykset oli jaoteltu SW-, QA-, ja DO-ryhmäkohtaisesti, sillä näiden kolmen ryhmän vastuualueet ATDD:hen liittyen poikkeavat toisistaan. Kolmannessa osiossa selvitettiin vastaajien (ja tiimien, jossa ovat työskennelleet) osaamista hyväksymistestien kirjoittamiseen ja käyttöön liittyen. Neljäs osio oli varattu avoimille kysymyksille menetelmän ongelmista ja hyvistä menettelytavoista. Tässä osioissa myös tarkasteltiin vastaajien halukkuutta ja tarvetta osallistua ATDD-koulutuksiin, sekä pyydettiin antamaan menetelmälle kokonaisarvosana. Kappaleissa 7.1–7.4 käydään läpi osakohtaiset tulokset, joista vedetään johtopäätökset kappaleessa 7.5.

7.1 OSA I: Taustojen kartoitus

Verkkokysely lähetettiin 60 henkilölle ja vastauksia kyselyyn tuli kaikkiaan 25. Vajaan 42 % vastausprosenttia voidaan pitää korkeana, sillä kyselyyn pyydettiin vastaamaan vain niitä, jotka olivat työskennelleet menetelmää käyttäen tai osallistuneet koulutukseen aiheesta. Tähän kategoriaan sopivien henkilöiden määrä ei suuresti poikkea saavutetusta 25 vastauksesta, vaikka tarkka osallistuneiden lukumäärä ei ole tiedossa. Kyselyyn vastanneista valtaosa (20) kuului sovelluskehittäjien ryhmään, johon lasketaan kuuluvan sovelluskehittäjien (software developer) lisäksi myös harjoittelijat (software trainee), sovelluspäälliköt (software lead/chief), sekä sovellusarkkitehdit (software architect). Tähän ryhmään viitataan myöhemmin lyhenteellä SW. Laadunvarmistuksen, eli QA-ryhmään kuuluivat seuraavat roolit: harjoittelija (QA trainee), laadunvarmistusasiantuntija (QA specialist) sekä laadunvarmistuspäällikkö (QA lead/chief). Pienen kokonsa vuoksi tältä ryhmältä saatiin vain kolme vastausta, joka on kuitenkin 100 % nykyisen tiimin koosta. Viimeinen tarkasteltava ryhmä, johon viitataan myöhemmin lyhenteellä DO, koostui pelkästään toimitusomistajasta (delivery owner) ja tältä ryhmältä saatiin kaksi vastausta, joka on 50 % kaikista tämän ryhmän edustajista, joilla oli edellytykset vastata kyselyyn.

Valtaosa (80 %) vastaajista kuului SW-ryhmään ja vastaavasti 12 % QA-ryhmään ja 8 % DO-ryhmään. 68 %:lla vastaajista oli aikaisempaa kokemusta testitapausten kirjoittamisesta Gherkin-syntaksilla.

Kuvassa 8 on esitettynä vastaajien osallistuminen hyväksymistestien kirjoitussessioihin. Nähdään, että 80 % vastaajista on osallistunut 1–10 tapaamiseen. Vain 16 %:lla vastaajista on tätä enemmän osallistumisia, joista vain 4 %:lla osallistumisia on enemmän kuin 20.

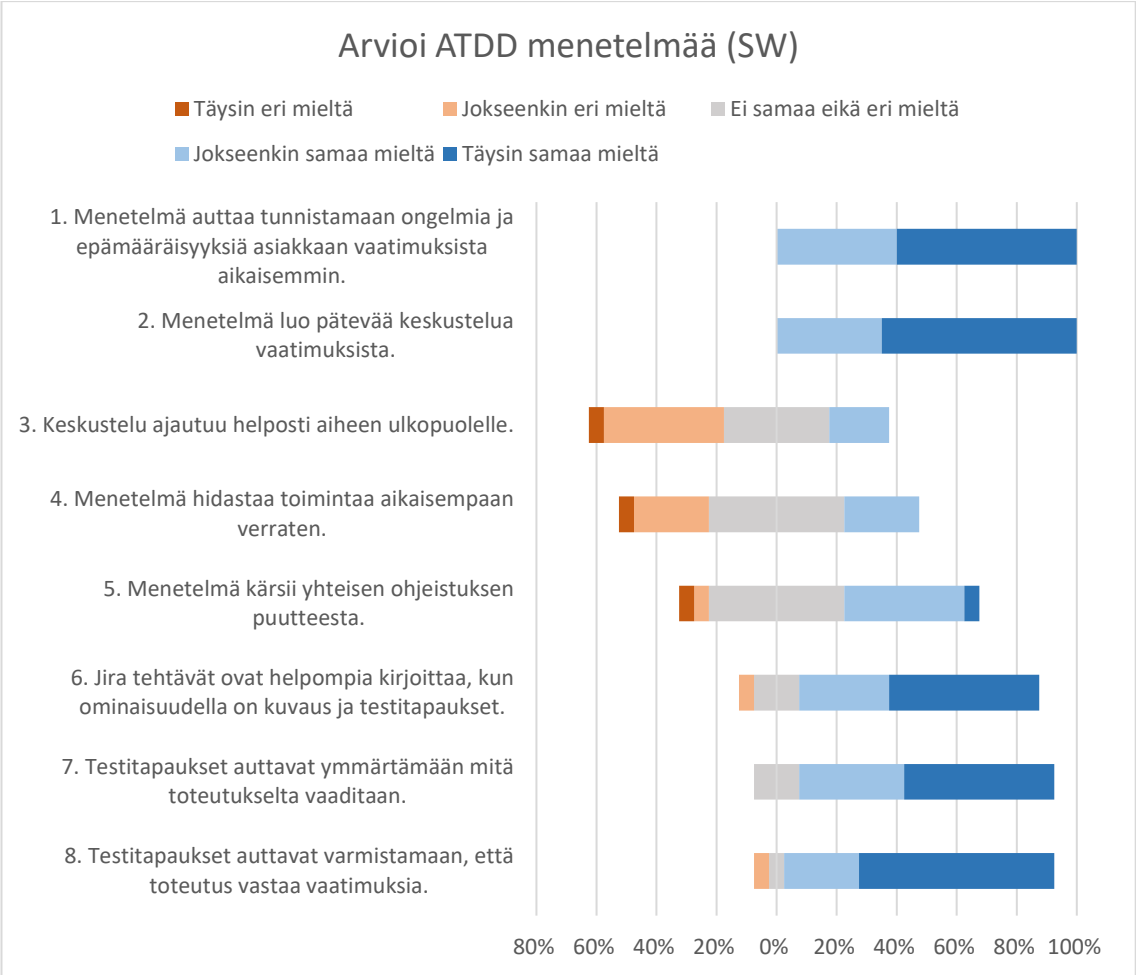


Kuva 8: Osallistumiskerrat hyväksymistestien kirjoitustapaamisiin

7.2 OSA II: Menetelmän arviointi

Väittämät 1–5 olivat kaikille rooleille menetelmän arvioinnin suhteen samat. Väittämät tästä ylöspäin olivat roolikohtaisia. Väittämät on käännetty suomeksi kuviin 9–11. Alkuperäiset englanninkieliset väittämät löytyvät liitteestä A.

Kuvasta 9 nähdään, että SW-ryhmään kuuluvat olivat voimakkaasti samaa mieltä myönteisten väittämien kanssa. Kielteiset väittämät 3 ja 4 jakavat mielipiteitä, mutta painottuvat voimakkaammin kielteisiin vastauksiin. Kielteisen väittämän 5 kanssa ollaan vain lievästi samaa mieltä.

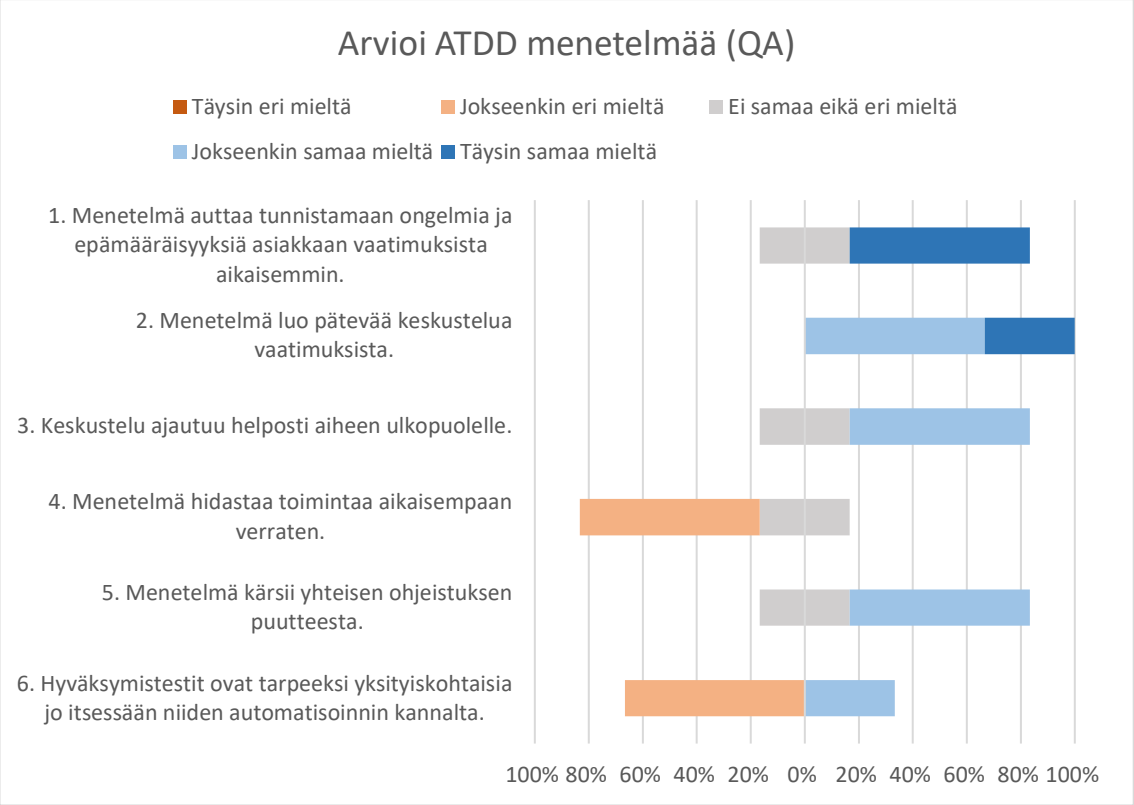


Kuva 9: SW-ryhmään kuuluvien vastaukset ATDD-menetelmästä

QA-ryhmään kuuluvien vastaukset (kuva 10) poikkeavat yhteisten väittämien osalta SW-ryhmästä 3. väittämän osalta, eli keskustelun nähdään ajautuvan aiheen ulkopuolelle. Tähän ryhmään kuuluvat eivät pidä nykyisiä testejä tarpeeksi yksityiskohtaisina, jotta ne voitaisiin automatisoida ilman testin ulkopuolista kuvausta tai tietoa.

Kuvasta 11 nähdään, että DO-ryhmään kuuluvien vastaukset ovat linjassa myönteisten väittämien osalta aikaisempien ryhmien kanssa. Kielteisten väittämien 3–5 suhteen DO-ryhmään kuuluvat ovat vahvemmin samaa mieltä kuin SW- ja QA-ryhmät. Roolikohtaiseen myönteisyyteen väittämään (6. väittämä) saadaan DO-ryhmältä melko vahva luottamus.

Ensimmäisessä väittämässä menetelmän avusta löytämään ongelmia, valtaosa SW-ryhmään kuuluvista vastaajista (60 %) oli täysin samaa mieltä. Loput 40 % vastaajista olivat hekin jokseenkin samaa mieltä väittämän kanssa. Vastaavasti QA-ryhmässä 67 % vastaajista oli täysin samaa ja DO-ryhmässä 100 % vastaajista jokseenkin samaa mieltä.

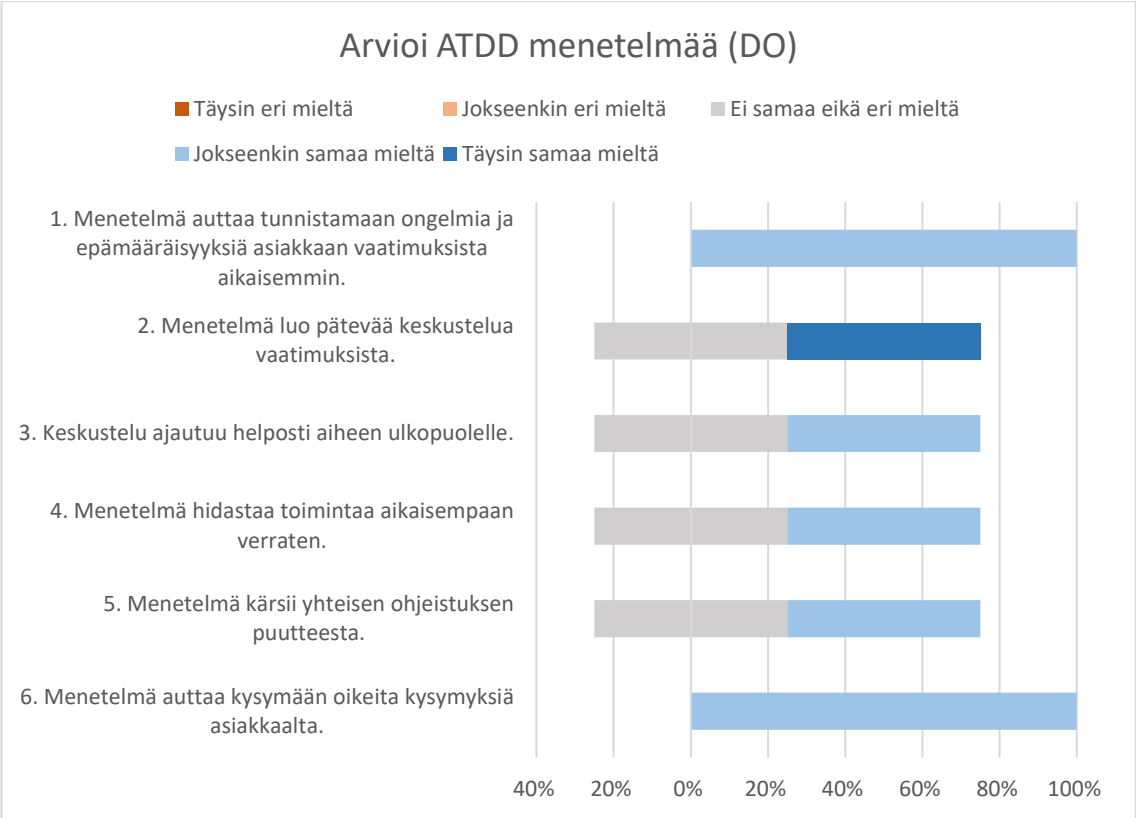


Kuva 10: QA-ryhmään kuuluvien vastaukset ATDD-menetelmästä

Toiseen väittämään vastattiin ensimmäistä väittämää vielä lievästi samanmielisemmin. 65 % SW-kuuluvista oli täysin samaa mieltä, että menetelmä auttaa luomaan pätevää keskustelua vaatimuksista. Vastaavasti QA-ryhmässä 33 % oli täysin samaa mieltä ja loput 67 % jokseenkin samaa mieltä. DO-ryhmässä oltiin puoliksi täysin samaa mieltä ja puoliksi ei samaa eikä eri mieltä.

Kolmannessa väittämässä menetelmän synnyttämän keskustelun ajautumisesta aiheen ulkopuolelle tuli enemmän hajontaa kuin kahden ensimmäisen väittämän kohdalla. 45 % SW-ryhmästä oli joko täysin tai jokseenkin eri mieltä, 20 % jokseenkin samaa mieltä ja 35 % ei samaa eikä eri mieltä. Vastaavasti QA-ryhmällä enemmistö (67 %) oli jokseenkin samaa mieltä väittämän suhteen ja DO-ryhmällä vastaukset jakautuivat tasan jokseenkin samaa mieltä ja ei samaa eikä eri mieltä välille.

Neljännessä kysymyksessä menetelmän väitettiin hidastavan prosessia aikaisempaan menettelyyn nähden. SW-ryhmässä vastaukset jakautuivat tasaisesti jokseenkin eri ja samaa mieltä olevien kesken. 45 %:lla tästä ryhmästä ei kuitenkaan ollut mielipidettä. QA-ryhmässä oltiin 67 %:sti jokseenkin eri mieltä väittämän kanssa ja DO-ryhmässä vastaukset jakautuivat kolmannen väittämän tapaan.



Kuva 11: DO-ryhmään kuuluvien vastaukset ATDD-menetelmästä

Viidennessä väittämässä kartoitettiin vastaajien näkemystä yhteisen ohjeistuksen puuttumisen merkityksestä. SW-ryhmässä 45 %:sti oltiin joko täysin tai jokseenkin samaa mieltä, että yhteisen ohjeistuksen puuttuminen vaikuttaa kielteisesti. 45 %:lla ei ollut asiaan mielipidettä ja vain 10 % vastaajista oli joko jokseenkin tai täysin eri mieltä. QA-ryhmässä oltiin 67 %:sti samaa mieltä ja DO-ryhmässä 50 %:sti jokseenkin samaa mieltä.

Kuudennessa väittämässä SW-ryhmälle esitettiin, että testitapaukset ja ominaisuuden kuvaus yhdessä auttavat luomaan Jira-tehtäviä. 80 % vastaajista oli väittämän kanssa jokseenkin tai täysin samaa mieltä. Vain 5 % vastaajista oli väittämän kanssa eri mieltä.

Seitsemännessä väittämässä SW-ryhmälle testitapausten väitettiin auttavan ymmärtämään mitä toteutukselta vaaditaan ja 85 % vastaajista oli jokseenkin tai täysin samaa mieltä asian kanssa. Kahdeksannessa väittämässä SW-ryhmälle testitapausten väitettiin auttavan varmistamaan, että luotu toteutus vastaa vaatimuksia. 90 % vastaajista oli jokseenkin tai täysin samaa mieltä väittämän kanssa ja vain 5 % vastaajista oli eri mieltä.

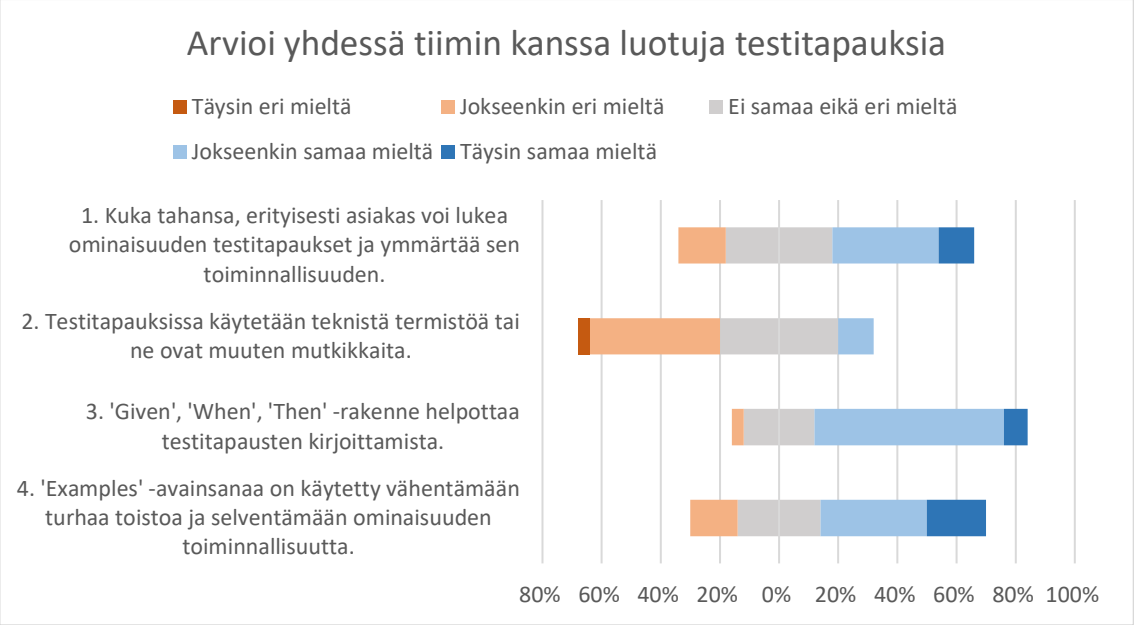
Kuudennessa väittämässä QA-ryhmälle testitapausten väitettiin olevan tarpeeksi yksityiskohtaisia, jotta ne voitaisiin automatisoida ilman testitapauksen ulkopuolista tietoa. 67 % vastaajista oli jokseenkin eri mieltä ja 33 % jokseenkin samaa mieltä väittämän kanssa. Kuudennessa väittämässä DO-ryhmälle prosessin väitettiin auttavan kysymään

oikeita kysymyksiä asiakkaalta ja 100 % vastaajista oli väittämän kanssa jokseenkin samaa mieltä.

Täysin eri tai samaa mieltä olevien vastausten osuus kaikkien SW-ryhmän väittämien osalta oli 41 % ja mielipiteettömien vastausten osuus oli 20 %. Vastaavat prosentit QA-ryhmälle olivat 17 % ja 23 %, sekä DO-ryhmälle 8 % ja 33 %.

7.3 OSA III: Testitapausten arviointi

Väittämät hyväksymistestien kirjoittamisesta olivat samat kaikille rooleille, eikä niitä tästä syystä ole eritelty kuvaan 12. Väittämät on käännetty suomeksi kuvaan ja alkuperäiset englanninkieliset väittämät löytyvät liitteestä A.

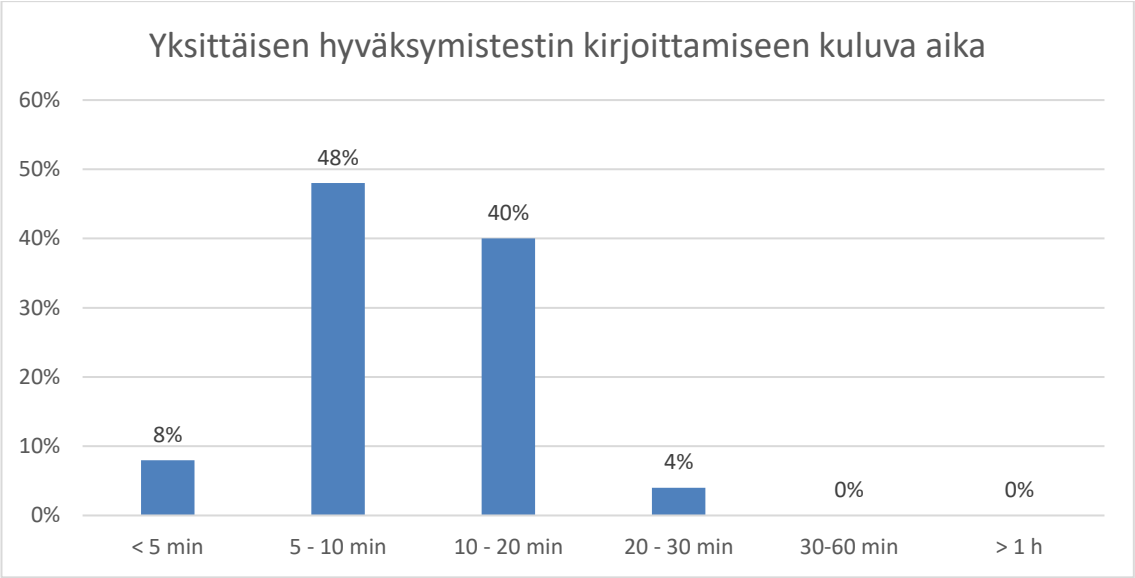


Kuva 12: Vastaukset väittämiin luoduista testitapauksista

Ensimmäinen väittämä käsitteli tapaamisissa luotujen testitapausten selkeyttä kuvata ominaisuuden toiminnallisuutta. 48 % piti luomiaan testitapauksia jokseenkin tai täysin väittämän mukaisina. 36 %:lla vastaajista ei kuitenkaan ollut mielipidettä väittämään.

Toisessa väittämässä kartoitettiin luotujen testitapausten kompleksisuutta. 48 % vastaajista oli väittämän kanssa jokseenkin tai täysin eri mieltä. Vain 12 % vastaajista oli jokseenkin samaa mieltä. Ei samaa eikä eri mieltä vastanneiden määrä oli 40 %.

Kolmanteen väittämään Gherkin-syntaksin hyödyllisyydestä testien kirjoittamisessa valtaosa vastaajista (72 %) vastasi olevansa joko jokseenkin tai täysin samaa mieltä. Vain 16 %:lla ei ollut väittämään mielipidettä tai olivat jokseenkin eri mieltä.



Kuva 13: Yksittäisen hyväksymistestin kirjoittamiseen kuluva aika

Neljännessä väittämässä selvitettiin Examples-avainsanan käyttöä ja hyödyllisyyttä. Lievä enemmistö (56 %) oli väittämän kanssa jokseenkin tai täysin samaa mieltä. 44 % vastaajista oli kuitenkin ilman mielipidettä tai korkeintaan jokseenkin eri mieltä.

Täysin eri mieltä tai samaa mieltä olevien vastausten osuus kaikkien väittämien osalta oli vain 11 %, kun ei samaa eikä eri mieltä vastanneiden osuus oli 32 %.

Kuvasta 13 nähdään, että yksittäisen hyväksymistestin luomiseen kuluva aika on jakautunut melko tasaisesti välille 5–20 min. Vain 8 % prosenttia vastaajista on ilmoittanut kirjoittamisen tapahtuvan tätä nopeammin ja 4 % tätä hitaammin.

7.4 OSA IV: Avoimet kysymykset

Tässä osiossa esitetyt kysymykset ja vastaukset ovat suomennettuja. Niiden alkuperäiset kirjoitusasut löytyvät liitteistä A ja B. Tässä kyselyn osassa vain kysymykset lisäkoulutustarpeesta (13. kysymys) ja yleisarvosanasta (14. kysymys) olivat pakollisia vastata. Keskimäärin 43 % kyselyyn osallistuneista vastasi myös avoimiin kysymyksiin.

Ensimmäisessä kysymyksessä kartoitettiin menetelmän ongelmia:

”10. Mitkä ovat tämänhetkiset ongelmat ATDD:hen liittyen?”

Toistuvina ongelmina esiin nousi erityisesti kokemuksen ja käytännönläheisen koulutuksen puute, sekä menetelmän hitaus ja vaikeus luoda testitapauksia joillekin ominaisuuksille. 68 % vastasi tähän kysymykseen. Taulukkoon 1 on koottu joitakin vastauksia suomeksi.

”Jotakin työtä on vaikea kuvailla testien muodossa. Esimerkiksi palvelun toteutus, jollekin uudelle koneelle.”
”Noin yhdessä soveltamissessiossa olen ollut mukana, mutta en ole koskaan saanut koulutusta menetelmään. Kyseisestä sessiosta ei tullut yhtään mitään.”
”Kaikki eivät käytä, ollaan vielä opettelu vaiheessa.”
”Ominaisuuden määrittely, jota yhden ATDD testin tulisi testata on hankalaa. Esimerkiksi, erottelemme PLC (Programmable Logic Controller) ja MMS ominaisuudet, vaikka loppukäyttäjä näkee ne yhtenä.”
”Vaatimukset muuttuvat jatkuvasti, jolloin kaikki Jiraan tehty työ on pian vanhentunutta. Meillä ei ole aikaa päivittää kaikkea ajan tasalle.”
”On olemassa monia kysymyksiä, joihin emme aina saa vastauksia ja meidän tulee jatkaa epävarmuudessa, siitä miten asiakas tulee järjestelmää käyttämään. Näin tapahtuu etenkin silloin, kun työskennellään välittäjän kautta.”
”Meillä ei ole tapaa seurata tunteja, joita käytetään ATDD:hen. Meidän täytyy kyetä arvioimaan, kuinka paljon ATDD aiheuttaa lisätyötunteja ja kuinka paljon se vähentää kehitystyötunteja.”
”Gherkin-syntaksi on kaksiteräinen miekka, se pakottaa kirjoittamaan tietyn tyyppisiä testejä, mutta on jokseenkin monimutkainen. Käytön perusteleminen automatisoidulla testaamisella ei ole pätevä argumentti, sillä olemme vielä kaukana siitä. Ehkä voisimme arvioida toista tapaa kirjoittaa testejä.”

Taulukko 1: Koottuja vastauksia kysymykseen 10

Seuraavassa kysymyksessä kartoitettiin vastaajien ryhmäkohtaisia ja hyväksi havaittuja menettelyjä:

”11. Onko sinulla hyviä menettelytapoja, joita ei mainita ohjeistuksessa?”

Vain 28 % vastasi tähän kysymykseen ja vastauksista vain 43 % kuvasi hyväksi havaittua menettelyä. Vastauksissa kehotettiin käyttämään jotakin toista tekstieditoria, kuten Notepadia Jiran sijasta. Jiran käyttämistä testien kirjoittamiseen pidettiin kömpelönä. Pidettiin myös hyödyllisenä, että olisi olemassa esimerkki MMS:n ominaisuudesta valmiiksi tehtynä testitapauksineen sekä linkit Jiran työkaluihin, jotka ovat menetelmässä käytössä. Taulukkoon 2 on koottuna joitakin vastauksia.

”Voisimmeko saada valmiiksi tehdyn esimerkin MMS:n ominaisuudesta. Tämän ominaisuuden tulisi olla yksinkertainen ja eristetty. Sen tulisi olla hyvin dokumentoitu.”
”En usko, että tätä on vielä mainittu, mutta olen ohjeistanut tiimejä käyttämään Exceliä/ Notepadia, tai vastaavaa ATDD tapaamisessa, jolloin aikaa ei kulu Jirassa klikkailuun.”
”Ruudun voi jakaa siten, että toisella puolella on auki Jira tai toiminnallinen kuvaus ja toisella testit auki tekstieditorissa.”

Taulukko 2: Koottuja vastauksia kysymykseen 11

Jatkokysymyksenä edelliseen pyydettiin kertomaan ongelmista ohjeistukseen liittyen:

”12. Oletko eri mieltä jonkin ohjeistuksessa esitetyn menettelyn kanssa, miksi?”

Yhdeksi ongelmaksi nousi, että ohjeistuksessa ominaisuudet on kuvattu teknisestä näkökulmasta loppukäyttäjän sijasta. Toinen ongelma nähtiin ajatuksessa, että ominaisuudella ei saisi olla avoimia kysymyksiä siinä vaiheessa, kun toteutusta aletaan tekemään. Koettiin, että oletuksia joudutaan tekemään, jotta kehitystyö pysyisi aikataulussa. Tähän kysymykseen vastasi 32 %, joista 38 % oli merkityksellisiä kysymyksen kannalta. Alla olevaan taulukkoon on koottuna joitakin vastauksia kysymykseen liittyen.

”Ominaisuudet ovat kuvattuna teknisestä näkökulmasta loppukäyttäjän sijasta.”
”Kun ominaisuuksia luodaan, on epätodennäköistä, että ”ei avoimia kysymyksiä” -tilanne voitaisiin realistisesti saavuttaa. Tämän tulisi todellakin olla tavoite, mutta usein oletuksia joudutaan tekemään, jotta voidaan jatkaa eteenpäin ja pitää kehitystyö aikataulussa.”

Taulukko 3: Koottuja vastauksia kysymykseen 12

Tarvetta uusille ATDD-koulutuksille kartoitettiin seuraavassa kysymyksessä. Valtaosa (80 %) vastaajista toivoi lisää koulutuksia.

Tämän jälkeen pyydettiin antamaan menetelmälle kokonaisarvosana: ”14. Arvioi ATDD:n sopivuutta Fastemsille yhdellä arvosanalla. Ota arvosanassa huomioon menetelmän tuomat hyödyt, ongelmat ja potentiaali.” Asteikolla 1–10 keskiarvoksi saatiin 8,16, josta matalin yksittäinen annettu arvosana oli 6 ja korkein 9. 84 % kaikista vastaajista antoi arvosanaksi 8 tai 9.

Osion viimeisessä kysymyksessä sai antaa vapaata palautetta. 44 % kaikista kyselyyn osallistuneista vastasi tähän. Esiin nousi taulukossa 4 listattuja asioita.

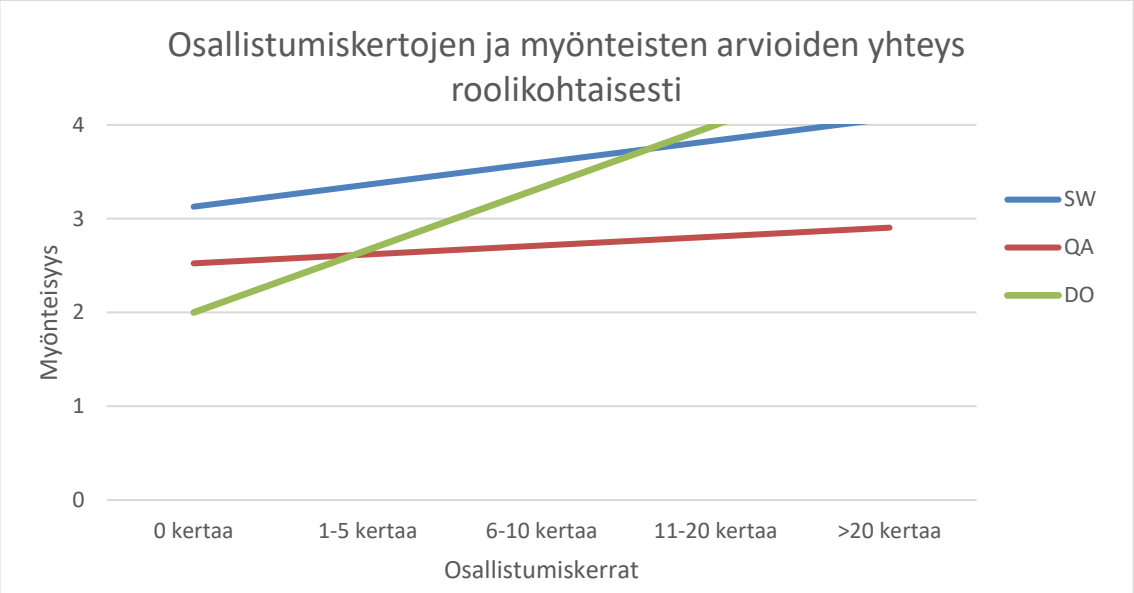
”Koulutuksiin liittyen, ei välttämättä ole tarvetta uusille koulutuksille, vaan sessiolle, jossa ”Fastemsin tapa” ATDD:n tekemiseen selvennetään kaikille. Mielestäni kaikkien tiimien tulisi toimia ATDD:n suhteen yhtenäisellä tavalla.”
”Menetämme paljon potentiaalia, kun työskentelemme ATDD:n parissa vain sisäisesti. Meidän tulisi tuoda asiakas mukaan prosessiin, jolloin testitapaukset todella määrittelisivät ominaisuudet. Jos asiakas ei ole osana prosessia, testitapaukset kuvaavat vain, kuinka sisäisesti haluamme ominaisuuden toimivan. Monissa tapauksissa tämä ei ole mitä asiakas haluaa tai tarvitsee. Kiinteä hinnoittelu ohjelmistoille on tietysti hieman ongelmallista.”
”MMS Vanilla -tiimi ei käytä menetelmää ollenkaan. Ehkä pitäisi, ehkä ei.”
”Selkeämmät yksinkertaiset ohjeet kuinka Jirassa tehdään. Laitepuolelle (PLC) myös otettava käyttöön (muutokset eivät näy MMS:ssä). Suurimmat ongelmat asiakkaalla juuri laiterajapinnassa. Integrointitestit, joissa testataan koko toiminnallisuus.”
”Meidän tulisi yhtenäistää testien kirjoittaminen ja lopulta nähdä mitä hyötyjä saamme tästä.”
”Olisi mielestäni turhan raskasta tuoda ATDD vanilla jatkuvaan kehitykseen, mutta voisin kuitenkin kokeilla.”
”Haluaisin korostaa, että ATDD on koko yrityksen kattava muutos, siihen miten työskentelemme. Myynti, johto, kehitys, testaus ja se avaa uusia tapoja kommunikoida asiakkaan kanssa.”
”En usko, että hyödynnämme ATDD:ta sen täyteen potentiaaliin. DO:t ovat avainasemassa aina myynnistä toiminnallisen kuvauksen kirjoittamiseen. Pitäen ”ATDD” mentaliteetin, voisimme tukea koko prosessia paremmin.
”ATDD:n ei tule olla vain ylimääräinen kerros lisättyä työtä. Meidän tulee löytää tapa hyödyntää testejä uudelleen simuloidussa-, tehdas- ja käyttöönotto -ympäristöissä testiautomaation avulla.”

Taulukko 4: Koottuja vastauksia kysymykseen 15

7.5 Johtopäätökset

Kuvassa 14 on esitetty kokemuksen yhteys positiivisiin väittämiin ATDD-menetelmästä roolikohtaisesti. Vastausten Likert-asteikko on muunnettu asteikolle 0–4, jossa 0 vastaa mielipidettä ”täysin eri mieltä” ja 4 mielipidettä ”täysin samaa mieltä”. Näistä on laskettu

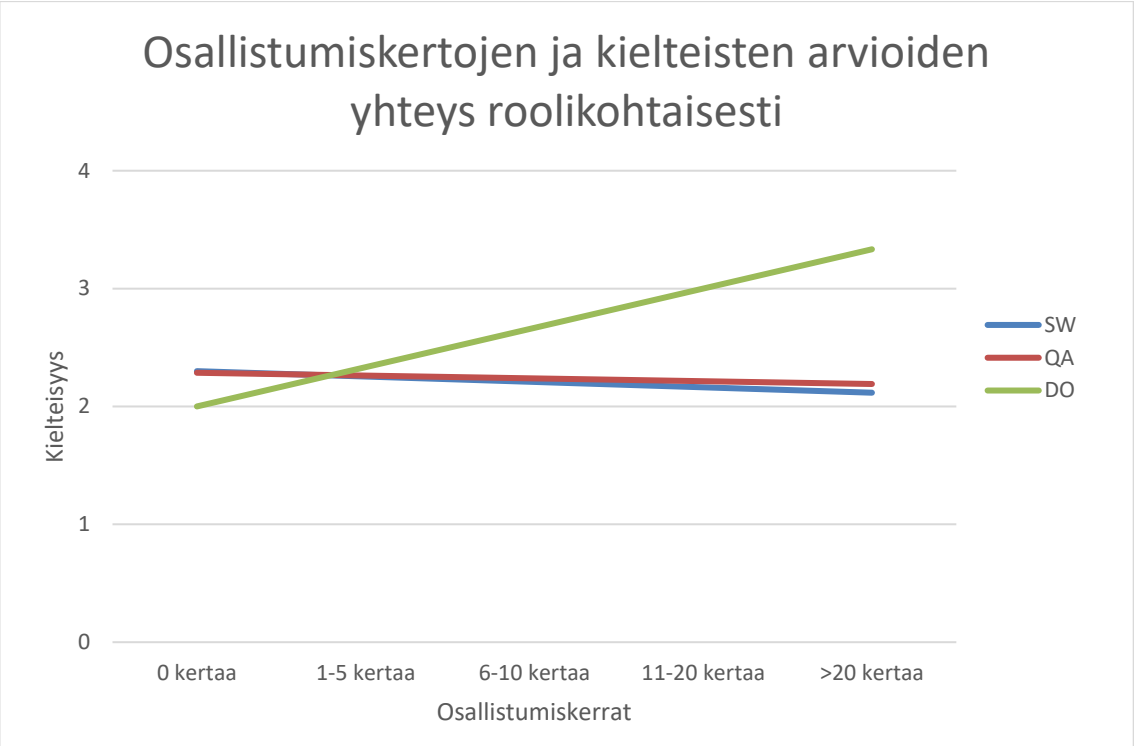
keskiarvot osallistumiskertakohtaisesti ja lineaarinen kuvaaja piirretty kuvaamaan kokemuksen yhteyttä positiivisiin väittämiin. Korkeampi arvo tarkoittaa positiivisempaa kokemusta menetelmän hyödyistä.



Kuva 14: Kokemuksen yhteys positiivisiin väittämiin ATDD-menetelmästä

Kuvassa 14 on tarkasteltu positiivisia väittämiä 1–2 ja 6–8. Kuvasta 14 nähdään, että kokemuksen karttuessa myönteisyys menetelmän hyötyjä kohtaan kasvaa roolista riippumatta. Kuvasta nähdään myös, että SW-ryhmään kuuluvat suhtautuvat keskimäärin myönteisemmin menetelmää kohtaan. DO-ryhmässä kokemuksen karttumisella on voimakkain yhteys myönteiseen suhtautumiseen. QA-ryhmässä kokemuksen lisäämisellä on verrokkiryhmistä heikoin yhteys myönteisyyteen. Tässä ryhmässä on myös keskimäärin heikoin myönteisyys. Kaikki ryhmät ovat kuitenkin keskimäärin kaikissa kokemusluokissa myönteisiä menetelmän hyötyjä kohtaan.

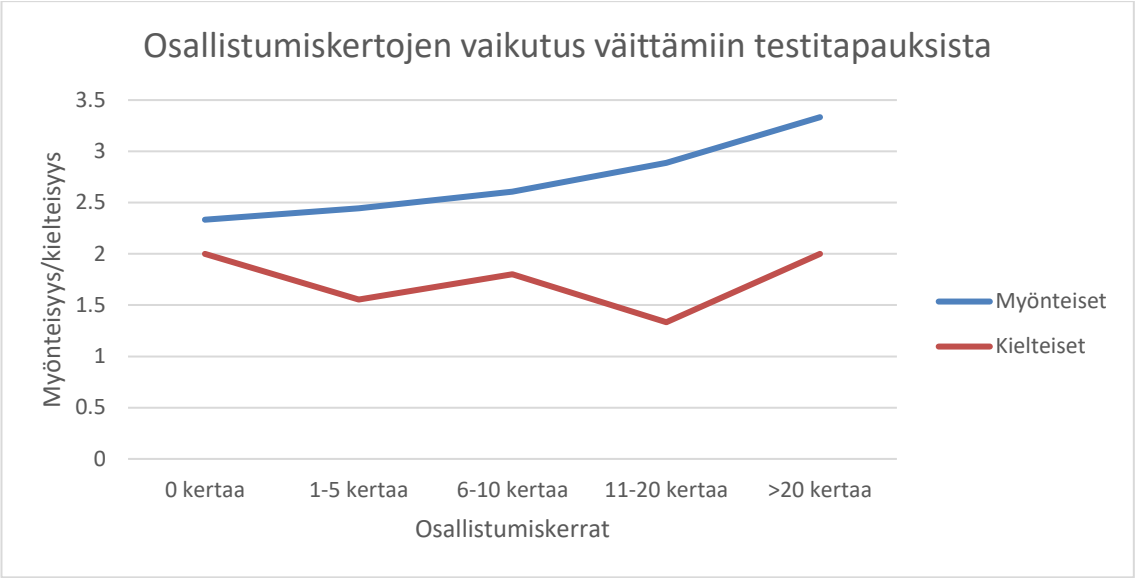
Kuvan 15 tarkastelun kohteena ovat kielteiset väittämät 3–5. Korkeampi arvo tarkoittaa vahvempaa kokemusta menetelmän ongelmista.



Kuva 15: Kokemuksen yhteys negatiivisiin väittämiin ATDD-menetelmästä

Nähdään, että kokemuksen karttuessa SW- ja QA-ryhmillä kokemus menetelmän ongelmista heikkenee, kun taas DO-ryhmään kuuluvilla kokemus vahvistuu. Huomion arvoista on se, että kuvaajat ovat hyvin lähellä arvoa 2. Tämä vastaa Likert-asteikolla vastausta ”ei samaa eikä eri mieltä”. Keskimäärin vastaajilla SW- ja QA-ryhmissä ei ole siis ollut vahvaa mielipidettä kielteisiin väittämiin. Tämä voi olla seurausta vastaajien kokemuksen puutteesta tai kysymysten huonosta asettelusta.

Kuvaan 16 on piirretty kuvaajat kyselyn testeihin liittyvistä myönteisistä ja kielteisistä väittämistä. Kuten kahdessa aikaisemmassa kuvassa, myös tässä Likert-asteikko on muutettu arvovälille 0–4, jotta vastausten keskiarvot on voitu laskea. Kuvasta nähdään, että kokemuksen karttuminen vahvistaa myönteisyyttä positiivisia väittämiä kohtaan. Keskimääräisesti positiivisiin väittämiin ollaan kaikissa kokemuskategorioissa myönteisiä. Negatiivisia väittämiä kohtaan ollaan kaikissa kategorioissa lievästi eri mieltä, mutta kokemuksen lisääntymisellä ei näytä olevan merkittävää yhteyttä tähän.



Kuva 16: Kokemuksen yhteys positiivisiin ja negatiivisiin väittämiin testeistä

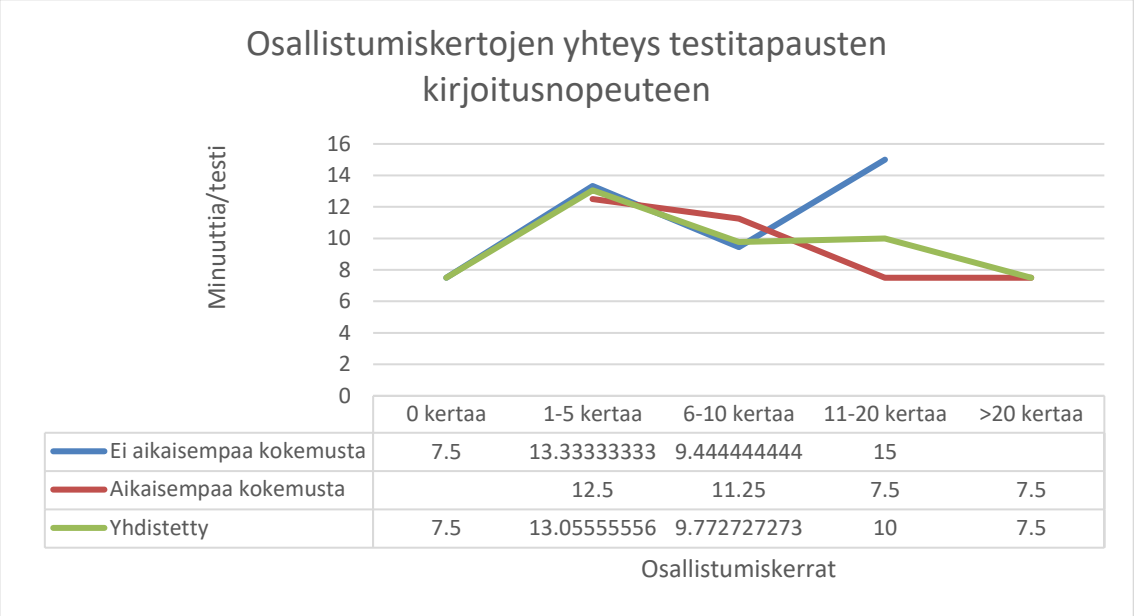
Kuvassa 17 on esitetty kokemuksen yhteys nopeuteen, jolla hyväksymistestejä saadaan luotua. Kuvaa tulkitessa tulee huomioida, että testit kirjoitetaan ryhmissä, mutta vastaukset on annettu yksilöinä. Tästä seuraa, että harvoihin tapaamisiin osallistuneen vastaus voi heijastaa ryhmän tulosta, jossa testit on kirjoitettu, eikä kyseisen yksilön. Tämä jokseenkin vääristää tulkittavia tuloksia yksilötasolla.

Toinen huomioitava asia on, että 80 % vastaajista oli osallistunut 1–10 tapaamiseen ja ääripäät 0 ja yli 20 kertaa edustavat vain 8 % koko otoksesta. Lisäksi 0 kertaa osallistuneella ei voi olla luotettavaa tietoa testien kirjoittamisesta. Luotettavin arvioitava väli onkin siis 1–20. Käytetty aika testiä kohden on laskettu siten, että vastausvaihtoehtojen aikavälit kuten 5–10 minuuttia on muutettu aikavälin keskiarvoksi. Edellä mainittu arvo olisi siis 7,5 minuuttia. Näistä on sitten laskettu kokemuskohtainen keskiarvo.

Kuvaan on tuotu vielä erilliset kuvaajat riippuen, ilmoittiko vastaaja, että hänellä oli aikaisempaa kokemusta testien kirjoittamisesta Gherkin-syntaksilla. Kuvasta nähdään, että siirryttäessä väliltä 1–5 välille 6–10, riippumatta aikaisemmasta kokemuksesta, käytetty aika testitapausta kohden tippuu reilulla kolmella minuutilla, joka on 25 % nopeampaa kuin aikaisemmassa kategoriassa.

Seuraava nopeutuminen nähdään siirryttäessä 11–20 kerrasta yli 20 kertaan. Tällöin nopeutuminen on aikaisempaan nähden jälleen 25 %. Vastaajilla, joilla ei ole aikaisempaa kokemusta testien kirjoittamisesta, käytetty aika kasvaa siirryttäessä 6–10 kerrasta 11–20 kertaan yli viidellä minuutilla, joka on 59 % hidastuminen aikaisempaan kategoriaan nähden. Tätä voi selittää, että testitapauksiin on alettu kiinnittämään enemmän huomiota, kun ymmärrys siitä, mitä niiltä vaaditaan, on kasvanut. Kyseessä voi olla myös

vain yksittäisen ryhmän edustajat, sillä vain 12 % vastanneista ilmoitti osallistuneensa 11–20 kertaa.



Kuva 17: Kokemuksen yhteys yksittäisen testin kirjoittamiseen kuluvaan aikaan

Avoimissa kysymyksissä on erotettavissa useampia kategorioita, joihin liittyen saatiin useampia vastauksia. Yksittäinen vastaus voi kuulua useampaan eri kategoriaan samanaikaisesti. Merkittävät kategoriat ja niiden prosenttiosuus kaikista annetuista vastauksista:

1. Koulutuksen, ohjeistuksen tai harjoituksen puute (~37 %)
2. Epäselvyydet tai vaikeudet prosessin käyttöönottamisessa (~17 %)
3. Testien kirjoittamiseen liittyvät ongelmat (~17 %)
4. Kasvanut ajankäyttö (~14 %)
5. Menetelmän arvioinnin hankaluus (~11 %)

Ensimmäiseen kategoriaan, eli koulutuksen, ohjeistuksen tai harjoituksen puutteeseen liittyviä vastauksia tuli selvästi eniten. 80 % vastaajista kertoi haluavansa lisää koulutusta menetelmään liittyen (kysymys 13). Tämän lisäksi kuvista 14–16 on selvästi nähtävissä kokemuksen yhteys positiivisempaan suhtautumiseen menetelmää ja testejä kohtaan. On siis hyvin todennäköistä, että kattavammalla koulutuksella, yhtenevällä ohjeistuksella ja harjoittelulla saadaan kehitettyä osaamista yksilö- ja ryhmätasolla, sekä tuotua voimakkaammin esiin ATDD:n positiivisia puolia.

Toiseen kategoriaan liittyi epäselvyydet tai vaikeudet siitä, miten ATDD kokonaisuudessaan tai osiltaan istuu osaksi nykyistä työskentelyä. Osaltaan vastaukset liittyivät myös

Fastemsilla tämän työn ulkopuolella tehtävään Master Backlog -uudistukseen, joka yksinkertaistettuna muuttaa sitä, miten ominaisuuksien kanssa tullaan työskentelemään jatkossa. ATDD on kuitenkin osa tätä prosessimuutosta ja siksi tähänkin liittyvät vastaukset ovat merkityksellisiä. Tähän liittyviä ongelmia voidaan osaltaan myös korjata yhteisellä ohjeistuksella ja koulutuksella. Kategoriaan sisältyy myös ongelmat ATDD-mallin laajentamisesta yritystasolla, jotta menetelmä ei jäisi pelkästään ohjelmistokehityksen piiriin. Toimiakseen ATDD edellyttää, että kaikki sidosryhmät mukautuvat muunnokseen. Taulukossa 4 esitetty vastaus, jossa puhutaan siitä, kuinka ATDD:n tulisi olla osa myyntiä, johtoa, kehitystä ja testausta on esimerkki tästä. Toisessa vastauksessa samaan kategoriaan liittyen mainitaan, kuinka oleellista on asiakkaan saaminen osaksi prosessia. Asiakkaan sitouttamisesta uuteen malliin onkin Fastemisilla jo tekeillä uusi lopputyö, joka tullaan tekemään DO:n näkökulmasta.

Kolmanneksi kategoriaksi muodostui haasteet hyväksymistestien luomisessa. Yhtenä ongelmana tässä kategoriassa nähtiin, että asiakaslähtöinen näkökulma testeihin katoaa helposti ja testeistä tulee hyvin teknisiä. Tästä huolimatta vain 12 % kaikista vastaajista oli samaa mieltä tämän näkökulman kanssa, kuten kuvasta 14 jo aiemmin havaittiin. Toisena haasteena pidettiin Gherkin-syntaksia, joka pakottaa kirjoittamaan testit tietyllä rakenteella. Koettiin, että joissakin tapauksissa testattava ominaisuus voi olla toiminnallisuudeltaan sellainen, että sille on vaikeaa kirjoittaa testiä, joka noudattaisi vaadittua rakennetta. Toisaalta kuvasta 14 aikaisemmin nähtiin, että 72 % vastaajista piti rakennetta helpottavana tekijänä ja vain 4 % oli asiasta eri mieltä.

Neljänteen kategoriaan liittyi menetelmästä aiheutunut kasvanut ajankäyttö. Kuvasta 9 nähdään, että vain 25 % vastanneista SW-ryhmässä oli jokseenkin samaa mieltä, että menetelmä hidastaa tekemistä. Kuvista 10 ja 11 nähdään vastaavat lukemat 0 % ja 50 % pienemmille QA- ja DO-ryhmille. SW-ryhmästä 45 %:lla, QA-ryhmästä 33 %:lla ja DO-ryhmästä 50 %:lla ei ollut mielipidettä asiaan. Vahvojen mielipiteiden vähyys voi olla seurausta kokemuksen vähyydestä, joka jo itsessään esiintyy merkittävimpänä kategoriana avoimissa kysymyksissä. Kokemuksen puutteesta on voinut seurata, että vastaajalle ei ole vielä kehittynyt selvää kuvaa siitä, kuinka paljon menetelmä lopulta vaikuttaa ajankäyttöön.

Viidenteen kategoriaan, eli menetelmän arvioinnin hankaluuteen liittyviä vastauksia oli muutamia. Vastauksissa nousi esille ongelma siitä, että ATDD:hen käytettyjen tuntien yhteyttä koko projektin kehitystyötunteihin on äärimmäisen vaikea arvioida. Pitkällä aikavälillä ATDD:n yhteyttä voidaan tarkastella vertaamalla aikaisempien projektien käyttöönottojen venymisiä projekteihin, joissa ATDD:ta on käytetty.

8. YHTEENVETO

Luvussa 8.1 tarkastellaan työstä saatuja vastauksia työn tutkimuskysymysten kannalta. Tässä luvussa tehdään yhteenveto luvuista 5 Hyväksymistestivetoinen työskentely, 6 Tapaustutkimukset ja 7 Kyselytulokset saadun tiedon perusteella. Seuraavaksi luvussa 8.2 tarkastellaan kriittisesti tulosten luotettavuutta ja yleistettävyyttä. Viimeiseksi luvussa 8.3 esitetään jatkotutkimusideoita työhön liittyen.

8.1 Työn tulokset

Tämän työn tarkoituksena oli tutkia, miten hyväksymistestivetoisen ohjelmistokehityksen käyttöönotto on edennyt Fastemsilla, mitä hyötyjä ja mitä ongelmia menetelmästä on noussut esille ja miten uutta prosessia voitaisiin kehittää edelleen. Näistä työn tutkimuskysymyksiksi muodostuivat seuraavat:

- Minkälainen on Fastemsin hyväksymistestivetoinen prosessi?
- Mitkä ovat Fastemsin hyväksymistestivetoisesta prosessista saadut edut?
- Mitkä ovat Fastemsin hyväksymistestivetoisen prosessin ongelmat?
- Miten Fastemsin hyväksymistestivetoista prosessia voidaan tulevaisuudessa kehittää eteenpäin?

Ensimmäiseen tutkimuskysymykseen saadaan vastaus luvusta 5, jossa avataan ATDD:n käyttöä Fastemsilla. Hyväksymistestivetoinen ohjelmistokehitys perustuu kolmikon toimintaan, johon kuuluvat toimitusomistaja, testaaja sekä ohjelmistokehittäjä. Toimitusomistaja vastaa ryhmässä asiakkaan järjestelmän ja vaatimusten asiantuntijuudesta. Testaaja on vastuussa hyväksymistestien suorittamisesta manuaalisesti. Ohjelmistokehittäjän vastuulla on luoda toteutus, joka vastaa kirjoitettuja testejä. Kaikki ovat vastuussa asiakasvaatimusten kirjoittamisesta Gherkinillä selkeiksi ja kuvaaviksi hyväksymistesteiksi.

Toiseen tutkimuskysymykseen saadaan vastauksia luvuista 5 ja 7, joista jälkimmäisessä perehdytään kyselyn tuloksiin. Uuden hyväksymistestivetoisen prosessin avulla pystytään tunnistamaan epäselvät tai muuten ongelmalliset asiakasvaatimukset ennen kuin niitä aletaan toteuttaa. Menetelmä synnyttää keskustelua vaatimuksista, mikä auttaa luomaan yhteisymmärryksen osallistujien kesken siitä, mitä toimitettavalta järjestelmältä odotetaan. Tämän myötä toimitusomistajan on helpompi kysyä hyödyllisiä kysymyksiä

asiakkaalta selvittäessään vastauksia nousseisiin epäselvyyksiin. Menetelmä auttaa ohjelmistokehittäjiä luomaan ohjelmointitehtäviä ja varmentamaan, että toteutus vastaa vaatimuksia. Hyväksymistestien myötä toteutukseen tarvittu tieto on ollut paremmin yhdestä paikasta saatavilla.

Kolmanteen tutkimuskysymykseen saadaan vastauksia luvuista 5 ja 7. Viidennessä luvussa asiakkaan puuttuminen ja siitä seurannut ylimääräinen työ nostettiin yhdeksi menetelmän ongelmista. Toiseksi haasteeksi ilmeni viidennessä ja seitsemännessä luvuissa määrittelyiden ylläpitäminen vaatimusten muuttuessa. Menetelmän myötä hyväksymistestit ovat uusi paikka, johon muuttuvat määrittelyt tulee päivittää. Mikäli hyväksymistestit ovat automatisoitu myös niitä vastaavat testikoodit tulee päivittää. Seitsemännessä luvussa koulutuksen ja yhteisen ohjeistuksen puute nousi merkittäväksi puutteeksi.

Neljänteen tutkimuskysymykseen saadaan vastauksia luvuista 5, 6 ja 7, joista kuudennessa käsitellään kahta ATDD:ta onnistuneesti hyödyntänyttä projektia. Näistä kolmesta luvusta nousee esille kolme merkittävää kehitysehdotusta.

Asiakkaan mukaan tuominen nousee esille jokaisessa luvussa. Viidennessä luvussa havaittiin asiakkaan puuttumisen aiheuttavan lisätyötä ja estävän hyväksymistestien hyväksyttämisen asiakkaalla. Kuudennessa luvussa molemmissa tapaustutkimuksissa onnistumisen kannalta avainasemaan nostettiin yhteistyö asiakkaan kanssa. Seitsemännessä luvussa tuli ilmi, että ATDD:ta ei hyödynnetä sen täyteen potentiaalin ilman mukana olevaa asiakasta. Viidennessä luvussa nostettiin esille ristiriita asiakkaan integroinnin ja kiinteän hinnoittelun välillä. Asiakkaan mukaan ottaminen osaksi ATDD-prosessia tarkoittaisi myös muutoksia Fastemsin myyntiin.

Tarve lisäkoulutuksille ja yhteiselle ohjeistukselle tulee selväksi seitsemännestä luvusta. 80 % vastanneista toivoi lisää koulutuksia ATDD:hen liittyen. 37 % prosenttia kaikista avointen kysymysten vastauksista oli luokiteltavissa kategoriaan ”Koulutuksen, ohjeistuksen tai harjoituksen puute”. Kuudennen luvun tapaustutkimuksessa myös mainitaan viikon mittaisesta koulutuksesta, joka antoi projektitiimille työkalut toimia ATDD-mallin mukaisesti.

Kaikissa luvuissa mainitaan hyväksymistestien automatisoinnista tai uudelleen hyödyntämisestä. Viidennessä luvussa nostetaan esille tarve regressiotestaukseen myös projektiräätälöintien osalta. Kuudennessa luvussa molemmissa tapaustutkimuksissa hyödynnettiin testien automatisointia, jonka seurauksena toteutus oli luotettavampi ja sen toiminnallisuus oli helpompi todentaa. Seitsemännessä luvussa mainitaan, että tulee löytää tapa hyödyntää testejä uudelleen eri ympäristöissä.

8.2 Tulosten kriittinen arviointi

Viidennen luvun käytännöt ja havainnot pohjautuivat yksittäisen viisihenkisen ohjelmistotiimin, testaajan sekä toimitusomistajan näkemyksiin. Tämä on pieni määrä niistä henkilöistä, jotka ovat käyttäneet menetelmää. Tästä syystä luvussa esitetyt väittämät eivät välttämättä ole yleistettävissä yritystasolla. Kyselyn avulla ryhmän pienuuden kielteisiä vaikutuksia tulosten luotettavuuden suhteen voidaan kuitenkin vähentää.

Ensimmäisen tapaustutkimuksen tulosten yleistettävyyttä Fastemsilla tulee tarkastella varauksella, sillä vaikka tapaustutkimuksen projektin kehittäjillä ei ollut aikaisempaa kokemusta ATDD:sta, he kaikki olivat kokeneita ohjelmistokehittäjiä, joilla oli useamman vuoden kokemus ohjelmistoprojekteissa työskentelystä. Sama ei päde kaikkiin Fastemsin ohjelmistokehittäjiin.

Kyselystä saatuja johtopäätöksiä tulee tarkastella varauksella, sillä etenkin QA- ja DO-ryhmien otokset olivat niin pieniä, että näistä saatuja tuloksia ei voida yleistää. Toisaalta QA-ryhmään kuuluvien vastausprosentti oli 100, mikä tarkoittaa, että vastaukset antavat väittämien ja kysymysten piirissä tarkan kuvan ryhmän suhteesta menetelmään.

Pienen otoskoon lisäksi tuloksia vääristää vastauksissa havaittava positiivinen puolueellisuus, jolla tarkoitetaan yksilön taipumusta arvostella todellisuutta myönteisesti ja kohdella tuntemattomia asioita positiivisella oletuksella (Hoorens 2014). Kyselyn osioihin 2 (väittämät menetelmästä) ja 3 (väittämät testeistä) annetuista vastauksista myönteisiin väittämiin neutraaleja vastauksia oli vain 8 %, kun kielteisten väittämien suhteen vastaava luku oli 41 %.

Työssä käytettyjen kolmen aineiston tulokset tukivat toisiaan, eikä merkittäviä ristiriitoja näiden välillä ilmennyt. Tämä vahvistaa yksittäisestä aineistosta saatujen tulosten luotettavuutta tämän työn osalta.

8.3 Jatkotutkimus

Tätä työtä tulisi seurata uusi kysely, jonka avulla nähtäisiin, mihin suuntaan näkemykset ovat kehittyneet, ovatko prosessiin liittyvät ongelmakohdat hävinneet ja onko esiin noussut uusia ongelmia.

Tärkeää olisi myös tehdä vertailua aikaisempien projektien ja ATDD:ta hyödyntäneiden projektien kesken. Tulisi selvittää, mikä vaikutus ATDD-prosessilla on projektin loppupäässä ilmaantuviin vikoihin ja muihin odottamattomiin tilanteisiin. Tätä tulisi verrata tunteihin, jotka käytetään hyväksymistestien kirjoittamiseen ja tarkastella onko ajankäyttö perusteltua.

Merkittäväksi puutteeksi havaittiin asiakkaan puuttuminen prosessista. Tulisi tutkia siis, miten asiakas voitaisiin tuoda osaksi hyväksymistestivetoista prosessia ja mitä vaikutuksia tällä olisi esimerkiksi nykyiseen tapaan myydä ohjelmistoräätälöintejä.

LÄHTEET

[1] T. Walsh - Online Surveys in Collecting Cross-Cultural Qualitative User Experience Feedback, 2016. Saatavissa: https://trepo.tuni.fi//handle/10024/114835

[2] U. Flick - Doing Qualitative Data Collection - Charting the Routes, 2018. Saatavissa: https://methods-sagepub-com.libproxy.tuni.fi/book/the-sage-handbook-of-qualitative-data-collection

[3] R. P. Weber - Basic content analysis, 1990. Saatavissa: https://methods-sagepub-com.libproxy.tuni.fi/book/basic-content-analysis

[4] J. Watkins - Agile testing how to succeed in an extreme testing environment, 2009. Saatavissa: https://masterworkshop.skillport.com/skillportfe/assetSummaryPage.action?assetid=RW$352:_ss_book:32497

[5] Jenkins, 2021. Saatavissa: https://www.jenkins.io/doc/

[6] Docker, 2021. Saatavissa: https://www.docker.com/resources/what-container

[7] Robot Framework, 2021. Saatavissa: https://robotframework.org/

[8] Jira, 2021. Saatavissa: https://www.atlassian.com/software/jira/guides/getting-started/overview

[9] Zephyr, 2021. Saatavissa: https://marketplace.atlassian.com/apps/1014681/zephyr-for-jira-test-management?hosting=cloud&tab=overview

[10] Specflow, 2021. Saatavissa: https://specflow.org/

[11] Selenium, 2021. Saatavissa: https://www.selenium.dev/documentation/en/webdriver/

[12] K. Nicieja - Writing great specifications using specification by example and Gherkin, 2018. Saatavissa: https://learning.oreilly.com/library/view/writing-great-specifications/9781617294105/

[13] Cucumber, 2021. Saatavissa: https://cucumber.io/docs/gherkin/reference/

[14] D. Firesmith – Four Types of Shift Left Testing, 2015. Saatavissa: https://web.archive.org/web/20150905082941/https://insights.sei.cmu.edu/sei_blog/2015/03/four-types-of-shift-left-testing.html

[15] M. M. Moe - Comparative Study of Test-Driven Development (TDD), Behavior-Driven Development (BDD) and Acceptance Test–Driven Development (ATDD), 2019. Saatavissa: https://www.ijtsrd.com/papers/ijtsrd23698.pdf

[16] A. Gojko - Specification by Example: How Successful Teams Deliver the Right Software, 2011. Saatavissa: https://learning.oreilly.com/library/view/specification-by-example/9781617290084/

[17] R. Latorre - A successful application of a Test-Driven Development strategy in the industrial environment, 2014. Saatavissa: https://www-proquest-com.libproxy.tuni.fi/docview/1516373273?OpenUrlRefId=info:xri/sid:primo&accountid=14242

[18] W. Ye - Instant Cucumber BDD how-to a short and quick guide to mastering behavior-driven software development with Cucumber, 2013. Saatavissa: https://learning.oreilly.com/library/view/instant-cucumber-bdd/9781782163480/

[19] J. P. Sauvé, O. L. A. Neto - Teaching software development with ATDD and easyaccept, 2008. Saatavissa: https://dl-acm-org.libproxy.tuni.fi/doi/abs/10.1145/1352135.1352317

[20] K. Pugh - Lean-agile acceptance test driven development : better software through collaboration, 2011. Saatavissa: https://learning.oreilly.com/library/view/lean-agile-acceptance-test-driven/9780321719478/

[21] Fastems, 2021. Saatavissa: https://www.fastems.com/offering/mms/

[22] M. Gärtner - ATDD by example, 2013. Saatavissa: https://learning.oreilly.com/library/view/atdd-by-example/9780132763219/

[23] M. A. Basit, K. L. Baldwin, V. Kannan, E. L. Flahaven, C. J. Parks, J. M. Ott, D. L. Willett - Agile Acceptance Test-Driven Development of Clinical Decision Support Advisories: Feasibility of Using Open Source Software, 2018. Saatavissa: https://medinform.jmir.org/2018/2/e23/

[24] V. Hoorens - Positivity Bias. In: Michalos A.C. (eds) Encyclopedia of Quality of Life and Well-Being Research. Springer, Dordrecht, 2014. Saatavissa: https://doi.org/10.1007/978-94-007-0753-5_2219

LIITE A: ATDD-KYSELYLOMAKE

8760 Fastems

Acceptance Test Driven Development - survey

Purpose of this survey is to gather information on how well ATDD has been implemented into Fastems software delivery process, finding the common issues and gathering improvement ideas.

This survey is accompanied with current company instructions on working with the ATDD, created in collaboration with the QA team. It is recommended that you read through the instructions before you take this survey. You can find the instructions in Teams: SW Supply > General >Training materials > ATDD or clicking here:

[redacted]

Note: This survey is meant for those employees that have participated in ATDD sessions or trainings.

* Pakollinen

Background

1. I have previous experience in writing test cases with BDD or Gherkin syntax (Given, When, Then). *

- ◯ Yes
- ◯ No

12/19/2020

2. I have participated in acceptance test writing session *

- ◯ 0 times
- ◯ 1-5 times
- ◯ 6-10 times
- ◯ 11-20 times
- ◯ > 20 times

3. My current role at Fastems is *

- ◯ SW Trainee
- ◯ SW Developer
- ◯ SW Lead/Chief
- ◯ SW Architect
- ◯ QA Trainee
- ◯ QA Specialist
- ◯ QA Lead/Chief
- ◯ Delivery owner
- ◯ [ ] Muu

ATDD process [SW]

4. Evaluate the ATDD process *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Process helps identifying issues or ambiguities in customer requirements earlier.	○	○	○	○	○
Process creates valid discussion on the requirements.	○	○	○	○	○
Discussion easily drifts into unrelated matters.	○	○	○	○	○
Process slows things down compared to earlier.	○	○	○	○	○
Process suffers from lack of company wide instructions and guidelines.	○	○	○	○	○
Jira tasks are easier to write when feature has description and test cases.	○	○	○	○	○
Test cases help understand what is required of the implementation.	○	○	○	○	○
Test cases help ensure that implementation matches the requirements.	○	○	○	○	○

ATDD process [QA]

5. Evaluate the ATDD process *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Process helps identifying issues or ambiguities in customer requirements earlier.	○	○	○	○	○
Process creates valid discussion on the requirements.	○	○	○	○	○
Discussion easily drifts into unrelated matters.	○	○	○	○	○
Process slows things down compared to earlier.	○	○	○	○	○
Process suffers from lack of company wide instructions and guidelines.	○	○	○	○	○
Acceptance tests are detailed enough to automate them without external information.	○	○	○	○	○

ATDD process [DO]

6. Evaluate the ATDD process *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Process helps identifying issues or ambiguities in customer requirements earlier.	⬡	⬡	⬡	⬡	⬡
Process creates valid discussion on the requirements.	⬡	⬡	⬡	⬡	⬡
Discussion easily drifts into unrelated matters.	⬡	⬡	⬡	⬡	⬡
Process slows things down compared to earlier.	⬡	⬡	⬡	⬡	⬡
Process suffers from lack of company wide instructions and guidelines.	⬡	⬡	⬡	⬡	⬡
Process helps asking the right questions from the customer.	⬡	⬡	⬡	⬡	⬡

Acceptance tests

8. Evaluate the acceptance tests you and your team have created *

	Strongly disagree	Disagree	Neutral	Agree	Strongly agree
Anyone, especially the customer can read feature's test cases and understand its functionality.	○	○	○	○	○
Test cases use deep technical terminology or are otherwise elaborate.	○	○	○	○	○
'Given', 'When', 'Then' - structure helps writing better test cases.	○	○	○	○	○
'Examples' are used to reduce redundancy and clarify feature's functionality.	○	○	○	○	○

9. Writing a single acceptance test case usually takes *

- ○ < 5 min
- ○ 5 - 10 min
- ○ 10 - 20 min
- ○ 20 - 30 min
- ○ 30 - 60 min
- ○ > 1 h

Overview

You may answer to these questions in finnish or english

10. What are the current issues regarding ATDD?

11. Do you have a good practice that is not mentioned in the instructions?

Instructions: see link in survey description

12/19/2020

12. Is there some practice that you disagree with in the instructions and why?

Instructions: see link in survey description

13. **Would you like to receive more ATDD trainings? ***

- ◯ Yes
- ◯ No

14. Evaluate the ATDD process suitability for Fastems in one grade *

Consider value it brings, issues it has and future potential

1	2	3	4	5	6	7	8	9	10
◯	◯	◯	◯	◯	◯	◯	◯	◯	◯

LIITE B: VASTAUKSIA ALKUPERÄISESSÄ MUODOSSA

”Some work is hard to describe as tests. i.e. Write service implementation for some new machine.”
”Noin yhdessä soveltamissessiossa olen ollut mukana, mutta en ole koskaan saanut koulutusta menetelmään. Kyseisestä sessiosta ei tullut yhtään mitään.”
”Kaikki eivät käytä, ollaan vielä opettelu vaiheessa.”
”Defining what is the feature one ATDD test is testing is hard. For example we are separating PLC and MMS even if the end user sees this as one feature”
“- Requirements keep changing and then all worked done in JIRA is quickly outdated. We haven't had the time to keep features, stories, test cases and tasks up to date when this happens”
“There is many questions we don't always get answers for and we have to continue with uncertainties related to how the customer is planning to use the system. This happens a lot when dealer is between Fastems and final client.”
“We have no way of tracking the hours spent to ATDD. We need to be able to estimate the overhead it produces vs. development time it decreases.”
“Also, formality with Gherkin is a two edged sword. It forces one to write certain kind of tests but is somewhat complicated. Justifying it with automated QA is not a valid point, since we are far, far from there. Maybe we could evaluate another method for writing tests.”

Taulukko 5: Taulukon 1 alkuperäiset vastaukset

“Could we have a complete example of MMS's feature. The feature should simple and isolated. It should be well documented. Links to Jira tools etc.”
“Dont think this was mentioned, but I've guided some teams to work on excel/notepad or similar during ATDD meeting so you don't spend time clicking around in Jira.”
“Split screen where other side you show details from jira/fd etc. and other side have testcases open and all visible in a text editor.”

Taulukko 6: Taulukon 2 alkuperäiset vastaukset

“Features being from technical stack point of view instead of end user perspective”
“When creating features, it is unlikely that "no open questions" state can realistically be reached. This should definitely be the goal, but often assumptions must be made in order to move forward so that the development stays in schedule.”

Taulukko 7: Taulukon 3 alkuperäiset vastaukset

“About trainings: Maybe not any training but one session where "Fastems way" for ATDD is clarified for all. I think every team should implement ATDD similarly.”
“We are missing quite a lot of potential when we implement ATDD only internally. We must bring customers to the process, then the test cases would really justify the feature. If customer is not included, test cases only describes how we internally want the feature to work. In many cases, this is not what customer wants or needs. Of course, fixed pricing in software is little problematic.”
”MMS Vanilla -tiimi ei käytä menetelmää ollenkaan. Ehkä pitäisi, ehkä ei.”
”selkeämmät yksinkertaiset ohjeet kuinka JIrassa tehdään Laitepuolelle (plc) myös otettava käyttöön (muutokset ei näy MMS:ssä) .. Suurimmat ongelmat asiakkaalla juuri laiterajapinnassa. Integrointitestit jossa testataan koko toiminnallisuus.”
”We should unify making the tests. And eventually be able to see the benefits we get for this.”
“Olisi mielestäni turhan raskasta tuoda ATDD vanillan jatkuvaan kehitykseen, mutta voisin kuitenkin kokeilla”
“I'd like emphasize the importance of ATDD that it's hole company's change the way we work. Sales - management - development - testing and it open new ways to communicate with customers.”
“I don't think we still utilize the full potential of ATDD. Delivery Owners are in key role, from sales to writing FD having "ATDD" mindset we could support the whole process better.”
“ATDD isn't just an added layer of work. We need to find a way to really utilize the tests, using same tests on dev simulated environment, factory and site/commissioning with test automation supporting on the way.”

Taulukko 8: Taulukon 4 alkuperäiset vastaukset