Testausautomaation hyödyntäminen taloushallinto-ohjelmiston kehityksessä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus, Hämeenlinnan korkeakoulukeskus
kevät, 2022
Liisa Vuorenmaa

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Hämeenlinnan korkeakoulukeskus

Tekijä Liisa Vuorenmaa Vuosi 2022

Työn nimi Testausautomaation hyödyntäminen taloushallinto-ohjelmiston kehityksessä

Ohjaajat Esa Huiskonen

TIIVISTELMÄ

Opinnäytetyöni käsittelee ohjelmistojen testaamista, taloushallinto-ohjelmiston testaamista ja perehdyttää lukijan testiautomaation hyödyntämiseen ohjelmistokehityksessä. Opinnäytetyöni tarkoituksena oli saada rakennettua dynaaminen testiympäristö toimeksiantajani taloushallinto-ohjelmiston testaamista varten. Testaus on tarkoitus suorittaa taloushallinto-ohjelmiston kehityksen mukana, ja tässä opinnäytetyössä perehdytään pääasiassa ohjelmiston hyväksymistestauksen kehittämiseen ja suunnittelemiseen. Opinnäytetyöni toimeksiantaja oli Talenom Software Oy ja testattava ohjelmisto oli yrityksen kehittämä TiliJaska-niminen taloushallinto-ohjelmisto.

Opinnäytetyön tietopohja koostuu ohjelmistokehitykseen ja automaatiotestaukseen liittyvistä artikkeleista ja ohjelmistotestausta käsittelevistä kirjoista, sekä omista kokemuksistani ja huomioista työn edetessä. Tämä opinnäytetyö on toiminnallinen opinnäytetyö ja siihen kuuluu testaussuunnitelman ja automatisoidun testauksen kehittäminen TiliJaska-ohjelmistolle. Tutkimusmateriaali on kerätty omien havaintojeni pohjalta suorittaessani toiminnallista osuutta. Analysoin taloushallinto-ohjelmiston testauksen automatisointia, testausmenetelmiä ja -hyötyjä omien kokemuksieni kautta. Lopputuloksena syntyi kattava testisuunnitelma ja pohja testiautomaation rakentamiselle ko. ohjelmistolle, ja voidaan todeta, että ohjelmistotestaus näyttelee erittäin tärkeää osaa ohjelmistokehityksen jokaisessa vaiheessa.

Avainsanat Robot Framework, Ohjelmistotestaus, Taloushallinto.

Sivut 29 sivua ja liitteitä 1 sivu

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Abstract

Author Liisa Vuorenmaa Year 2022

Subject The benefits of automation testing in the financial management software development process

Supervisors Esa Huiskonen

ABSTRACT

This thesis presents software testing, software testing for financial management system, and this thesis introduce the reader to the testing process and the utilization of test automation in software development. The purpose of this thesis was to build a dynamic test environment for testing financial management software. Testing is performed in such a way that it can be involved in all future stages of the software development process, and this thesis mainly focuses on the development and design of acceptance testing. This Thesis is done for Talenom Software Oy and the software that this thesis concerns is financial management software called TiliJaska developed by Talenom.

The knowledge of this thesis is based on articles and books related to software development and automation testing, and also on my own experiences and observations while I working.This thesis is a functional thesis, the functional part consists of the development of a test plan and test automation for the TiliJaska- software. As research material, I have collected observations and notes while I worked on the functional part. The result of my research work was a comprehensive test plan and a good foundation for continuing to build test automation and it can be said that software testing is a very important part at every stage of software development.

Keywords Robot Framework, software testing, financial management

Pages 29 pages and attachments 1 page

Sanasto

DevOps	Ohjelmistokehityksen automatisointiin kehitetty toimintamalli
Scrum	Projektinhallintamenetelmä
Selenium	Web-testauskirjasto
BackEnd	kuvastaa ohjelmiston osaa, missä suoritetaan toiminnot ja tiedonkäsittely
ATDD	Acceptance Test Driven Development
BDD	Behaviour Driven Development
HTTP	Hypertext Transfer Protocol
SQA	Software quality assurance

Sisällys

Kuvat

Liitteet

1 Johdanto

Tämä opinnäytetyö antaa lukijalle tietopohjan ohjelmistotestauksen perusteista ja huomioon otettavista asioista testausta suunnitellessa. Työssä käydään läpi automatisoidun testauksen menetelmiä ja haasteita teoriassa ja käytännön esimerkkien kautta.

Teoriaosuudessa kuvataan ohjelmistotestauksen kehitystä historiasta nykypäivään sekä testaamisen periaatteita ja tyylejä. Käyn myös läpi testausprosessin vaiheet ja lukijalle tulee ilmi, miten testaamista voi tehostaa ja miten esittelemiäni asioita hyödynnetään käytännössä. Viimeisessä teorialuvussa johdattelen lukijan myös taloushallinto-ohjelmistojen pariin, ja käyn läpi ohjelmistojen perusperiaatteita sekä yrittäjien vaatimuksia hyvälle taloushallinnolle.

Opinnäytetyöni käytännön osuudessa keskityn testauksen automatisointiin ja testausprojektin suunnitteluun ja toteutukseen toimeksiantajalleni. Kerron miten testausprojektimme sai alkunsa ja mitä kaikkea siihen kuului, sekä esittelen lukijalle, miten testitapauksia luodaan ja automatisoidaan käytännössä. Toimeksiantajani on Talenom Software Oy ja työskentelen ko. yrityksessä testaajana. Työnkuvani on testata taloushallinto-ohjelmistoa manuaalisesti ja kehittää testausstrategiaa. Kehitän myös automatisoituja testitapauksia ohjelmiston hyväksymistestausta, sekä regressiotestausta varten sekä työnkuvaani kuuluu ohjelmistoprojektin hallintaa DevOps-ympäristössä.

Opinnäytetyön antaa vastauksen seuraaviin kysymyksiin:

- Mitä hyötyä on automaatiotestauksesta taloushallinto-ohjelmiston tuotannossa?

- Voidaanko taloushallinto-ohjelmiston testaus automatisoida?

2 Ohjelmistotestauksen teoria

Teoria-luvussa perehdytään ohjelmistokehityksen historiaan ja siihen, minkälainen rooli testauksella on ohjelmistojen kehityksessä. Luvussa syvennytään myös paremmin ohjelmistotestauksen menetelmiin ja vaiheisiin, sekä testauksen suunnitteluun.

2.1 Ohjelmistotestaus

Ohjelmistotestaus on ohjelmiston laadun varmistamista. Ohjelmistoa tulee testata, jotta saadaan varmistettua sen olevan toivotun kaltainen. Ohjelmistotestaus on prosessi, jonka tulisi kulkea mukana ohjelmistokehityksen jokaisessa vaiheessa. Ohjelmistotestaaja tekee työtä varmistaakseen, että ohjelmistokehityksen työntulos vastaa suunnitelmaa, ja hänen tulee raportoida virheistä ja eroavaisuuksista. Ohjelmistotestaus on siis äärimmäisen tärkeä osa ohjelmistokehitystä. (Singh & Singh,2019)

Ohjelmistojen laadusta puhuttaessa, tulee usein esille termi SQA eli Software quality assurance. SQA-termillä tarkoitetaan ohjelmiston laadun varmistamista eli ohjelmistotestaamisen kokonaisuutta ohjelmistoprojektissa. Hyvälaatuiselle ohjelmistolle on asetettu tiettyjä vaatimuksia. (Kautto,1996)

Tyypillisimmät vaatimukset ovat:

- Ohjelmisto vastaa määriteltyä tarkoitustaan
- Ohjelmisto on helposti ymmärrettävä ja käytettävä
- Ohjelmistoa on helppo ylläpitää ja päivittää
- Ohjelmisto on tietoturvallinen
- Ohjelmisto on suorituskykyinen sekä tunnistaa virheet ja rasituksen

Yleensä ohjelmiston täydellinen läpikäynti testaamalla on mahdotonta, joten järkevintä on aina keskittyä mahdollisimman kattavan ja järkevän testauksen suunnitteluun. Testauksen suunnittelu

tulee toteuttaa siten, että testitapaukset osoittavat ohjelmiston virheet ja poikkeavuudet mahdollisimman suurella todennäköisyydellä (Kautto,1996)

2.1.1 Ohjelmistotestauksen historia

Ohjelmistotestausta ei juurikaan tarvittu ennen vuotta 1956, tosin jotain yksinkertaisia laskentakoneiden testauksia saatettiin jo tuolloin tehdä. Tietotekniikan kehittyessä myös ohjelmistotestauksen tarve kasvoi ja 60-luvun alussa alettiin tarvita jo enemmän testaamista. Tietotekniikka jatkoi kasvamistaan ja kuusikymmenluvun lopussa sanotaan valloillaan olleen ”ohjelmistokriisi”. Koska tekniikka kehittyi nopeasti tehokkaammaksi, se synnytti monimutkaisempia ohjelmia useammilla toiminnoilla, minkä vuoksi niistä löytyi myös yhä useammin virheitä ja ongelmia ja testausta alettiin tarvita yhä enemmän. (Kasurinen, 2013)

Vasta 1980-luvun lopussa ohjelmistotestausta alettiin miettiä enemmän ennaltaehkäisevältä kannalta, ja tämä kanta on nykyisen ohjelmistotestauksen peruselementti. Ennaltaehkäisevällä testauksella tarkoitetaan sitä, että pyritään projektin alkuvaiheesta asti etsimään virheitä ja havainnollistamaan riskejä, tarkoituksena ennaltaehkäistä mahdollisia vikoja. Näin toimimalla todettiin virheiden korjaamisen olevan kustannustehokkaampaa ja nopeampaa kuin lähteä korjaamaan vikoja jälkikäteen valmiista ohjelmistosta. (Gelperin, Hezel 1988)

Tänä päivänä testauksesta on olemassa lukemattomia ohjeita, kirjoja ja artikkeleita. Testauksesta on tullut systemaattisempaa ja ohjelmistokehittäjät arvostavat testaamista eri lailla kuin menneinä vuosikymmeninä. Suuret yritykset haluavat panostaa testaamiseen ja on myös yrityksiä, jotka myyvät systemaattista testausta palveluna ohjelmistoprojekteihin. Nykyään myös on paljon erilaisia järjestöjä ja sertifiointeja, joilla testaajien ammattitaitoa pyritään nostamaan ja voidaan todeta, että nykyaikana yrityksillä on hyvä mahdollisuudet panostaa huolelliseen testaamiseen ja näin rakentaa varmoja ohjelmistoja kustannustehokkaammin. (Testauksen osaamisyhteisö,n.d)

2.1.2 Ohjelmistotestaus mukana ohjelmistokehityksessä

Testaus on tärkeä osa ohjelmistoprojektia. Testauksella voidaan nostaa ohjelmiston tasoa, luotettavuutta ja poistaa virheitä. Testauksen tulisi kulkea ohjelmiston kehityksen mukana koko kehitysprojektin ajan, ja ohjelmaa tulisi voida testata kehityksen jokaisessa vaiheessa tarkasti ja riittävästi. Testauksella osoitetaan ohjelmistossa olevia vikoja, tosin ei voida luottaa siihen, ettei vikoja olisi, vaikka testauksella ei löydettäisikään yhtään vikaa. Voidaan siis todeta, että testauksella osoitetaan vikoja, mutta ei sitä, että vikoja ei ole. Testaus perustuu aina riskien kartoittamiseen ja se ei ole ikinä täydellistä. (Kasurinen,2013)

Kokonaisuutena testaus on laajempi osa ohjelmistoprojektia kuin esimerkiksi ohjelmoijan työnkuva. Testaajan työ voi olla hyvin värikästä ja hänen toimenkuvansa vaihtelevat paljon yrityksen ja testattavan tuotteen mukaan. Testaukseen kuuluu myös mm. suunnittelua, analysointia, raportointia ja tiedonhallintaa. (Kasurinen, 2013)

Testaus kannattaa aina painottaa ensin niille osille ohjelmistoa, joissa vikoja on eniten. Testitapauksia aina päivitetään, lisätään ja kehitetään ohjelmiston kehityksen mukana, joten periaatteessa testaus ei ole ikinä valmis. (Kasurinen, 2013).

Ohjelmistotestaukseen on kehitetty erilaisia testausmalleja, suosituin on ns. ketterän kehityksen malli. Ketterä testaus voi tarkoittaa monenlaisia asioita, pääasia ketterässä testauksessa kuitenkin on, että ohjelmistoa ei testata vain ja ainoastaan vanhan ideaalin suunnitelmaohjatun testauksen mukaan. (Vuori, 2010)

Suunnitelmaohjatussa testauksessa ohjelmistoa testataan vain määrittelyjen pohjalta, jossa on päätetty jo aluksi, että mitä testataan ja miten. Ketterä testaus yleensä ottaen tarkoittaa mitä tahansa testausta, joka ei ainoastaan perustu ennalta suunniteltuihin testitapauksiin, tutkivaa testausta, missä testaaja edistää testaamista omien havaintojensa perusteella. Myös ohjelmoijien tekemä testaus kehityksen yhteydessä luetaan ketterään testaukseen. (Vuori, 2010)

Ketterä testaus voi myös tarkoittaa testaamista, jota suoritetaan ketterän ohjelmistokehityksen yhteydessä. Ketterä kehitys ohjelmistoprojektissa tarkoittaa, että täsmällisten

ennakkosuunnitelmien sijaan ohjelmistoprojekti elää vaatimusten mukaan, ajatuksena että ohjelmistoprojekti on dynaaminen ja jatkuva oppimisprosessi, missä ei voida etukäteen tietää tarkasti, miten tuote kehittyy, vaan sitä kehitetään vaatimusten mukana. (Vuori, 2010)

2.1.3 DevOps-toimintamalli

DevOps on lyhenne sanoista development ja operations. DevOps-toimintamalli on suosittu ohjelmistokehitysprosesseissa. Se tarkoittaa tuotehallintaa ja sen avulla pyritään automatisoimaan ohjelmistokehitykseen kuuluvat prosessit, kuten testaaminen ja ylläpito. Tarkoituksena on liittää yhdelle alustalle kaikki ohjelmistokehitykseen liittyvät osapuolet, kuten asiakkaat, kehittäjät, ohjelmistontuottaja, laadunvarmistus ja palveluiden ylläpito. DevOpsin avulla kaikki nämä ohjelmistokehityksen osa-alueet ovat helposti hallittavissa ja kaikkien projektin jäsenten nähtävillä. Se mahdollistaa nopeampaa kehitystä, koska kehittäjät voivat DevOps-ympäristön avulla vastata asiakkaiden tarpeisiin nopeammin. Pilvipalvelut ovat tärkeä osa DevOpsia ja DevOps-alustan käyttö vaatii jatkuvaa ylläpitoa, jotta toiminta on kannattavaa. Siihen kuuluu paljon erilaisia teknologioita sekä työkaluja, ja ympäristöt ovat myös integroitavissa moniin ohjelmistokehitysprosesseissa käytettäviin muihin alustoihin. DevOps-ympäristössä pystytään hallitsemaan myös testausprojektia sujuvasti kehityksen rinnalla ja sen kautta voi dokumentoida virheet ja parannusehdotukset muiden osapuolten nähtäväksi. (Abildskov, n.d.)

2.2 Ohjelmistotestauksen vaiheet

Luukkaisen (2021) mukaan ”täysin kattava testaaminen on mahdotonta ja testaus on joka tapauksessa työlästä, eli onkin tärkeää löytää kohtuullisen kokoinen testitapausten joukko, jonka avulla on kuitenkin mahdollista löytää mahdollisimman suuri määrä virheitä”.

Tästä syystä ohjelmistotestaukseen kuuluu eri vaiheita, joita kutsutaan testaustasoiksi, koska niissä testataan ohjelmistoa eri tasoilla. Yleisimmin käytetyt testaustasot ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. Testaus jaetaan näihin eri testaustasoihin riippuen testauksen kohteesta. (Kasurinen, 2013)

2.2.1 Yksikkötestaus

Yksikkötestaus tarkoittaa testauksen suorittamista yksikkötasolla, eli testataan pieniä ohjelman osia, joita kutsutaan moduuleiksi. Yleensä ohjelmoija tekee yksikkötestausta samalla kun kehittää ohjelmistoa. Hän testaa ja varmistaa jokaisen tekemänsä funktion ja koodin pätkän toiminnan. Testauksen kohteena siis ovat yksittäiset metodit ja luokat, ja yksikkötestauksen tavoitteena on ohjelmiston sisäisen laadun varmistus ja kontrollointi. Ohjelmistokehityksessä on tärkeää, että ohjelmistokoodin sisäinen laatu on tasokas, sillä huonolaatuisen koodin korjaaminen on työlästä ja jatkokehitys hankalaa. Näin ollen ohjelmiston kehitys voi hidastua ja kehitystyö tuntuu vähemmän mielekkäältä. On myös todettu, että ohjelmiston virheitä on kustannustehokkaampaa paikallistaa yksikkötestausvaiheessa kuin myöhemmissä testausvaiheissa. (Luukkainen, 2021)

2.2.2 Integraatiotestaus

Integraatiotestauksessa testataan monen eri toisiinsa integroidun moduulin toimintaa yhdessä. Kun yksikkötestauksessa testatut luokat ja metodit on integroitu kokonaisuudeksi, integraatiotestaus on järkevä aloittaa. Testejä suunnitellessa täytyy ottaa huomioon, kuinka vahvoja integraatioita moduulien välillä on. Jos integroitavien moduulien välillä on vahva kytkentä ja dataa liikkuu paljon, testausta täytyy tehdä enemmän. Luukkainen (2021) kertoo, että ”integraatiotestauksessa painopiste on ohjelman komponenttien välisten rajapintojen toimivuuden tutkimisessa sekä komponenttien yhdessä tuottaman toiminnallisuuden oikeellisuuden varmistamisessa.”

Integraatiotestaus sopii hyvin pienempien systeemien testaamiseen, ja raja integraatiotestauksen ja yksikkötestauksen välillä on häilyvä. Usein ajatellaan, että yksikkötestaus muuttuu integraatiotestaukseksi heti kun mukaan tulee yksiköiden välinen rajapinta. (Luukkainen, 2021)

2.2.3 Järjestelmätestaus

Järjestelmätestauksessa testataan, että ohjelmisto on kehitetty vaatimusmäärittelyjen mukaisesti, ja koko järjestelmä testataan huolellisesti kokonaisuutena. Useasti järjestelmätestauksen osana

testataan myös, kuinka ohjelmisto tai palvelin kestää kuormitusta. Usein myös käyttäjätestaus kuuluu järjestelmätestaukseen, jolloin testataan sovellusta perustuen sen käyttöskenaarioihin. Järjestelmätestaus on ainoa testauksen vaihe, jossa testataan sekä toiminnallisia, että ei-toiminnallisia ominaisuuksia. Järjestelmätestaukselle on olemassa monia muotoja kuten käytettävyystestaus, suorituskykytestaus ja tietoturvatestaus. Järjestelmätestaus suoritetaan yleensä integraatiotestauksen jälkeen. (Luukkainen, 2021)

2.2.4 Hyväksymistestaus

Hyväksymistestaus tarkoittaa koko testausprosessin viimeistä osaa. Se tehdään viimeisenä ennen järjestelmän käyttöönottoa. Hyväksymistestauksessa simuloidaan loppukäyttäjän toimintoja ja varmistetaan ohjelmiston toiminnallisuus. Hyväksymistestaus todentaa, että ohjelmisto toimii oikein. Tässä vaiheessa ohjelmistosta ei pitäisi löytyä enää virheitä. Hyväksymistestaus suoritetaan mahdollisimman valmiissa toimintaympäristössä oikealla datalla. Hyväksymistestauksen voi suorittaa myös asiakas, joka testauksen jälkeen päättää, täyttääkö tuote kaikki vaatimukset. (Luukkainen, 2021)

2.3 Suunnitelmallinen testaus

Testaus aloitetaan ohjelmistokehityksessä mahdollisimman aikaisessa vaiheessa, mieluiten jo samalla, kun suunnitellaan itse ohjelmistoa. Hyvin suunnitellun testauksen päämäärä on, että testaus on ketterästi liitetty mukaan jokaiseen ohjelmiston kehitysvaiheeseen. Usein testauksen resurssit ovat rajalliset, joten joudutaan tekemään paljon valintoja testattavien toimintojen välillä, tai kuinka kattavasti testejä suoritetaan. Hyvä testaussuunnitelma ja priorisointi auttaa käyttämään rajalliset resurssit tehokkaasti. Kun aletaan toteuttamaan suunnitelmallista testausprosessia, laaditaan testisuunnitelma, joka sisältää testausstrategian ja määrittelee testausprosessin ja sen kulun. (Ohjelmistotestauksen perusteet, versio 1.0, n.d., ss.20-22)

2.3.1 Testisuunnitelma

Testisuunnitelma on yksityiskohtainen asiakirja, jossa kuvataan testistrategia, testauksen tavoitteet, laajuus ja aikataulu, sekä testien suoritukset ja testaukseen tarvittavat resurssit. Testisuunnitelma auttaa määrittämään tarvittavat resurssit ja raamit testaukselle. (Hamilton,2021)

Testisuunnitelmalla on testaukselle monenlaista hyötyjä ja se auttaa sekä testaajia että ulkopuolisia ihmisiä ymmärtämään paremmin testauksen yksityiskohtia. (Hamilton,2021)

Ohjelmistohankkeen alussa testisuunnitelma voi olla pelkkä luonnos, missä ei kuvata paljoa yksityiskohtia, mutta projektin edetessä esiin tulee enemmän testaamisen tarpeita ja tietoa saataville, jolloin testisuunnitelmaa tulee laajentaa ja kasvattaa kehityksen mukana. Testisuunnitelma siis elää jatkuvasti ja seuraa tuotetta koko sen elinkaaren ajan. (Software Testing Fundamentals,2020)

2.3.2 Testitapaukset

Testitapausten suunnittelu on yleensä testisuunnitelman viimeinen vaihe. Testitapauksissa määritetään suoritettavat toiminnot testiskenaarion vaatimusten täyttymisten varmistamiseksi. Testitapauksia suunnitellessa tulee välttää turhia ja päällekkäisiä testitapauksia ja testitapauksia valitessa täytyy pyrkiä priorisoimaan ne siten, että valitaan eniten virheitä tuottavat tapaukset testattavaksi ensin. (Ohjelmistotestauksen perusteet, versio 1.0,n.d.,ss.24)

Testitapausta määriteltäessä pyritään kuvaamaan testitapauksen syötteet, askeleet ja lopputulos mahdollisimman yksinkertaisesti ja selkeästi. Testitapaus kuvataan sanallisesti askel askeleelta, kuten myös jokaisen askeleen haluttu lopputulos. Testitapaukselle tulee myös antaa yksilöllinen tunniste, nimetä testitapauksen laatija sekä päivämäärä, milloin tapaus on luotu. Testitapauksen suorittamisen jälkeen täytyy kirjata ylös, kuka sen on suorittanut ja milloin ja määritellä ympäristö, jossa testitapaus on suoritettu. Suorittamisen jälkeen määritellään myös testin lopputulos (hyväksytty tai hylätty), jos yksikin askel testitapauksessa ei vastaa testissä kuvattua, haluttua lopputulosta, tapaus on hylätty. (Ohjelmistotestauksen perusteet, versio 1.0,n.d.,ss.24)

2.4 Testitapausten automatisointi

Nykyajan ohjelmistokehityksessä on erittäin yleistä, että resursseja ja kustannuksia vaativa ohjelmistotestaus halutaan automatisoida. Hyvin pienissä projekteissa automatisointi ei välttämättä kannata, mutta isommissa ja jatkuvissa projekteissa testauksen automatisoinnilla voi säästää paljon aikaa ja kustannuksia. Testausautomaation mahdollisuus kannattaisi ottaa jo varhaisessa kehitysvaiheessa huomioon, vaikka automaatiotestauksen rakentamisen voi aloittaa missä tahansa ohjelmiston kehitysvaiheessa. Varhaisessa vaiheessa aloittaessa se tuo vähemmän työtä ohjelmistoprojektiin. Kun ohjelmisto on otettu käyttöön, siihen tulee lähes aina jotain myöhäisempiä muutoksia, joiden testaamisen automatisointi tuo suurta lisäarvoa projektille. Kehitystyö voi tukeutua siihen, että hyvin dokumentoitujen automaatiotestien varmistamana ohjelmisto toimii lisättyjen toimintojenkin jälkeen niin kuin pitää. ”Automatisoimalla ohjelmistotestausta päästään huomattavaan tehokkuuteen ohjelmiston laadunvarmistuksessa” (ATR Soft Oy,n.d)

Testitapausten automatisoinnilla tarkoitetaan manuaalisen testauksen simuloimista, eli manuaalinen testaus suoritetaan koneellisesti. Ennen testauksen automatisoimista täytyy olla valmiina yksityiskohtainen testisuunnitelma ja vaativuusmäärittely ohjelmistolle, jossa kuvataan tarkasti testauksesta haluttu lopputulos. (Pohjolainen, 2003).

Testiautomaatio päihittää ihmisen suorittaman testauksen silloin kun tapahtumaa pitää toistaa useaan kertaan. Säännöllisesti toistuvan tapahtuman testaamisessa ihminen on kehno, koska me tylsistymme nopeasti saman toistamiseen. Kun ihmiseltä puuttuu motivaatio, niin tekemiemme virheiden määrä lähtee nousuun niin testauksessa kuin koodauksessakin. Kone ei tylsisty koskaan ja tekee saman asian väsymättä aina uudestaan ja uudestaan, tarvittaessa erittäin nopeasti. (Testauksen osaamisyhteisö, n.d.)

2.4.1 Testitapausten automatisointi, hyödyt

Automatisoiduilla testitapauksilla yrityksen on mahdollista saada suuria hyötyjä. Yritys voi säästää jopa kuukausien ihmistyöt automatisoinnin avulla, kunhan automatisoidut testit on suunniteltu

huolellisesti ja systemaattisesti. Automatisoiduilla testeillä on suuri merkitys myös kehitystyöhön. Tärkeintä on, että kehittäjä saa palautetta tekemisistään nopeasti. (Mensio,2005)

Testitapausten automatisoinnilla pystytään huomattavasti vaikuttamaan testaamiseen kuluvaan aikaan ja budjettiin. Ohjelmistoa pystytään kehittämään nopeammin, koska testaukseen ei tarvitse enää kuluttaa niin paljoa aikaa, myös työmäärä pienenee. Työmäärä pienenee, koska testejä voidaan suorittaa monia yhtaikaa ja valvomatta. Automatisoidun testauksen avulla löytyneet virheet on helppo toistaa ja näin on nopeampi jäljittää virheiden juurisyyt. Testitapausten automatisointi myös keventää huomattavasti testaajan työtaakkaa, ja on motivoivampaa keskittyä työhön, jos päivä ei mene testausrutiineja toistaessa. Automatisoinnin avulla testausta on myös helpompi suunnata ohjelmiston kriittisille osille, ja näin eniten virheitä aiheuttamat ohjelmiston osat tulee testattua tehokkaimmin. (Pyhäjärvi & Pöyhönen, 2014)

2.4.2 Testitapausten automatisointi, haasteet

Testitapausten automatisointiin liittyy myös paljon haasteita. Parasta olisi, jos automatisoinnista vastaisi kokenut ohjelmistokehittäjä, joka tuntee ohjelmiston. Automatisoitujen testien luominen ei yleensä ole ongelmallista, suurimpana haasteena on tehdä prosessi tehokkaasti ja tuottavasti. Haasteena on yleensä automatisoitavan kohteen ymmärtäminen ja halutun testaustuloksen hahmottaminen. Tärkeintä on päättää mitä automatisoidaan, joten ohjelmiston avaintoiminnot tulee määritellä ennen automatisointia riittävän tarkasti, jotta toiminnoille voi luoda tehokkaan automatisoinnin. Testitapauksia automatisoidessa on tunnettava hyvin ohjelmiston logiikka ja rakenne. Testausautomaatio vaatii paljon työtä ja suunnittelua, ja tämän vuoksi suurena haasteena on usein myös yrityksen rajalliset resurssit. (Gharhai, 2020)

2.4.3 Testitapausten automatisointi, työkalut

Nykyään on paljon yrityksiä, jotka ovat keskittyneet testauspalvelujen tarjoamiseen, joten testaus on mahdollista ulkoistaa kokonaan suhteellisen helposti. Päädyttäessä ratkaisuun testaamisen hoitamisesta ilman ulkoisia palveluita kannattaa testausteknologia valita huolellisesti. (Pohjolainen, 2003. ss. 23-34)

Testityökalun hankinnasta päättäminen kannattaa tehdä ohjelmistokehitysprosessin varhaisessa vaiheessa. Työkalua valitessa tulee ottaa huomioon työkalun integroiminen käytettyyn kehitysympäristöön ja sen tuomat mahdollisuudet ja monipuolisuus. Myös kustannuksiin tulee varautua ennalta. Huomioonotettavia kustannuksia ovat mm. työntekijöiden kouluttamisesta ja valitun työkalun käyttöönotosta aiheutuvat kustannukset. Täytyy varmistaa, että valittu teknologia on yhteensopiva mahdollisimman monen kehityksessä käytetyn käyttöjärjestelmän, ohjelmointikielen tai muiden piirteiden kanssa. Testausprosessi kannattaa analysoida tarkkaan ennen teknologian valintaa, jotta saadaan määriteltyä tarkasti mitä uudelta työkalulta vaaditaan. (Pohjolainen, 2003. ss. 23-34)

Yksi suosituista testiautomaatiotyökaluista on Selenium, joka on testausautomaatiotyökalu verkkopalveluille. Selenium mahdollistaa selaimen automatisoinnin ilman erillisiä testauskomentokieliä. Selenium tarjoaa paljon erilaisia laajennuksia ja mahdollisuuksia selainpohjaiselle testaamiselle. Selenium mahdollistaa simuloinnin käyttäjän ja selaimen vuorovaikutuksesta ja se on yhteensopiva useiden automaatiotestausteknologioiden kanssa. Yksi suosituista Seleniumin kanssa käytetyistä teknologioista on Robot Framework. (Selenium, n.d.)

2.4.4 Robot Framework

Robot Framework on avoimen lähdekoodin testaustyökalu, jota voidaan käyttää sekä testiautomaatiossa että prosessiautomaatiossa. Robot Framework säätiö tukee Robot Frameworkin kehitystä ja on voittoa tavoittelematon järjestö, johon kuuluu useita suuria yrityksiä. Säätiö takaa, että Robot Framework pysyy ajan tasalla ja että ohjelmistoa päivitetään jatkuvasti. Robot Framework on integroitavissa useimpien ohjelmistojen kanssa ja siihen kuuluu helppo syntaksi, jota käytetään avainsanoilla. Se on Python-pohjainen ohjelmisto, ja se on kehitetty erityisesti ohjelmistojen hyväksymistestaukseen (ADD), käyttäytymisperusteiseen ohjelmistokehitykseen (BDD), sekä robottiprosessiautomaatioon (RPA). (Robot Framework, n.d.)

Robot Frameworkilla voidaan käytännössä kirjoittaa avainsanoihin pohjautuen testiskripti, jonka suoritettua saadaan HTML-muotoinen tulosraportti ja lokitiedosto. Ennen skriptien kirjoittamista tulee kuitenkin tehdä tarvittavat asennukset, sekä määritellä testauksessa tarvittavat kirjastot ja

muut asetukset. Robot Frameworkin asennus on yksinkertainen, ja siihen löytyy kattavat ohjeet internetistä. Robot Frameworkin voi asentaa Windowsin komentokehotteen kautta, tai sitä voi käyttää jollain muulla alustalla. Yksi suosituista alustoista on JupyterLab. (Robot Framework, n.d.)

3 Ohjelmistot taloushallinnolle

Nykypäivänä yritysten taloushallinto on suurelta osalta siirtynyt digitaaliseen muotoon niin, että kaikki yrityksen taloushallintoon liittyvät asiat voidaan hoitaa sähköisessä muodossa taloushallinto-ohjelmistossa. Taloushallinnon sähköistyminen ja automatisoituminen on ollut suuri edistysaskel ja helpottaa huomattavasti taloushallintoon kuuluvia toimintoja. Monet ohjelmistoalan tekijät ovat myös huomanneet tämän mahdollisuutena, ja alkaneet kehittää yhä parempia järjestelmiä taloushallinnon hoitamista varten. (Taloushallintoliitto, 2018)

Yrityksen taloushallinnolla tarkoitetaan eri toimintoja, jotka kokonaisuutena muodostavat yritykselle taloushallinnon. Yrityksen taloushallinnon toimintoja ovat mm. kirjanpito, osto- ja myyntilaskujen käsittely, palkanlaskenta, maksuliikenne, viranomaisilmoitusten lähetys, arkistointi ja sisäinen laskenta. Hyvä taloushallinto-ohjelmisto mahdollistaa näiden kaikkien toimintojen suorittamisen sähköisesti ja yleensä ohjelmistossa on myös lisäominaisuuksia, kuten esimerkiksi verkkolaskutus, sähköinen arkistointi ja automatisoidut tiliöinnit. Taloushallinto-ohjelmiston tärkein ominaisuus on luotettavuus, sillä yrittäjälle yrityksen taloushallinnon järjestäminen on äärimmäisen tärkeä ja tarkka asia yrityksen liiketoiminnan sujuvuuden kannalta. Yrittäjät arvostavat luotettavia, helposti käytettäviä, selkeitä ja saavutettavia toimintoja sekä käytön aikana saatavaa nopeaa tukea tarpeen vaatiessa. Suomessa on monta toimijaa, jotka tarjoavat sähköisiä taloushallintojärjestelmiä, ja ohjelmistoja kehitetään jatkuvasti. (Taloushallintoliitto, 2018)

Tässä opinnäytetyössä tullaan perehtymään erityisesti Talenomin pienyrittäjille suunnattuun järjestelmään, jossa erikoisuutena verrattuna muihin järjestelmiin on mm. sen selkeys, helppokäyttöisyys ja yrittäjän mahdollisuus avata oma maksutili, joka mahdollistaa sen, ettei yrittäjän tarvitse avata pankkiin erikseen kallista yritystiliä, vaan kaikki taloushallintoasiat hoituvat tämän TiliJaska- ohjelmiston kautta. (TiliJaska, n.d)

TiliJaska ohjelmisto on jo tuotantovaiheessa ja asiakkaiden aktiivisessa käytössä. On tärkeää, että kaikki toiminnot ovat luotettavia ja suuria virheitä ei ohjelmakoodiin pääse päivitysten myötä. TiliJaskaa parannellaan myös aktiivisesti ja uusia yrittäjien arkea helpottavia toimintoja lisätään. Päivityksiä tulee usein, joten on tärkeää saada ohjelmistontestaus systemaattiseksi ja luotettavaan

tilaan, jotta suurilta virheiltä vältytään. Kuten aiemmin olen todennut, testaaminen ei voi olla virheetöntä, mutta tärkeintä on, että avaintoiminnot ovat moitteettomia ja jokaisessa päivityksessä mennään eteenpäin, kohti täydellistä, automatisoitua ja helppoa taloushallintoa.

4 Testausprojektin kehittämisen aloittaminen TiliJaska-sovellukselle

Projektin tarkoituksena oli luoda systemaattinen testausstrategia, sisältäen automaatiotestauksen TiliJaska-ohjelmistolle. Projekti aloitettiin määrittelemällä ja dokumentoimalla testausstrategia ja resurssit.

Tämän jälkeen siirryttiin luomaan testitapauksia eniten kriittiselle osalle ohjelmistoa, mistä oli löydetty eniten bugeja. Koko sivuston kattavat testitapaukset määriteltiin aiemmin kirjoitettujen käyttäjätapausten perusteella, ja koska valmis suunnitelma oli jo olemassa, testaaminenkin muuttui systemaattisemmaksi ja pystyttiin toteuttamaan järjestelmällisemmin sekä raportoimaan testien tuloksista paremmin.

Projektityössä edettiin käyttäen Scrum-projektimallia ja vastuualeeni oli testitapausten suunnittelu sekä automaatiotestien kehittäminen ja liittäminen ohjelmistohankkeen DevOps-ympäristöön.

Projektia kehitettiin Azure DevOps-ympäristössä. Sen kautta jokainen ohjelmistokehittäjä voi käyttää automatisoituja testitapauksia apuna kehitystyössään ja näin ollen pystyy testaamaan nopeasti omat ohjelmistoon tekemänsä muutokset. Tämä osoittautui erittäin tehokkaaksi ratkaisuksi. Ohjelmistokehittäjille on tärkeä apu saada työstään palautetta nopeasti, aikaa vievän manuaalisen testauksen sijaan. Projektin tarkoituksena on saada TiliJaska-ohjelmisto vielä entistäkin käyttäjäystävällisemmäksi ja toimivammaksi, jotta asiakkaat pystyvät hoitamaan kirjanpidon ja muut talousasiat helposti ja sujuvasti.

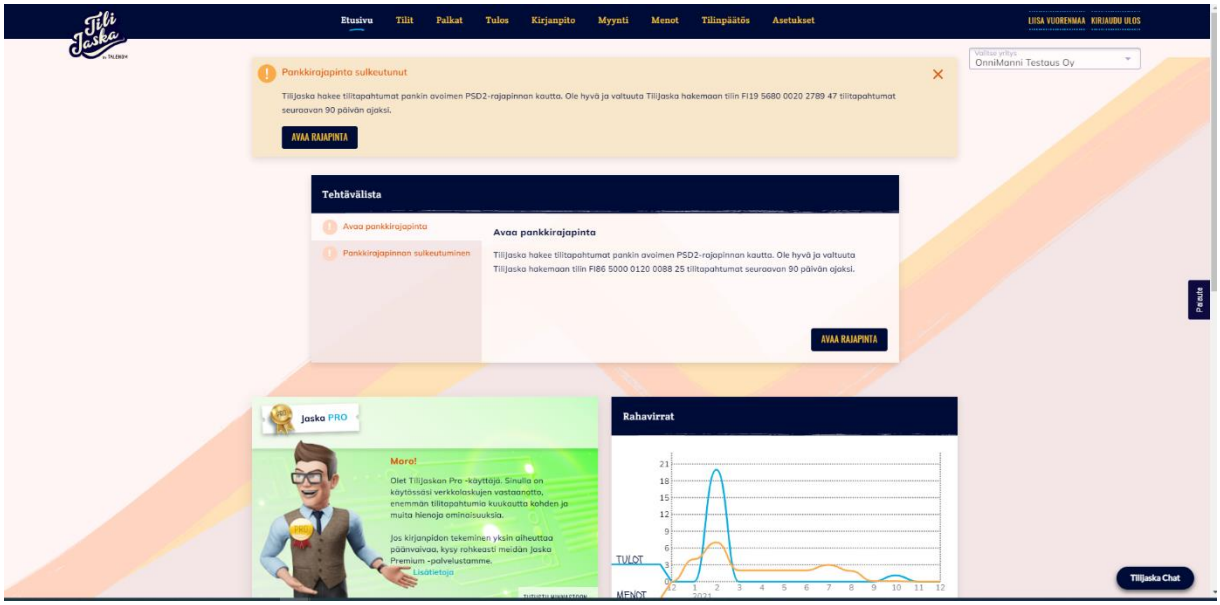
4.1 TiliJaska-sovellus

TiliJaska-sovellus on pienyrittäjille suunnattu taloushallinto-ohjelmisto, jossa yritys voi hoitaa sujuvasti kaikki yrityksen taloushallintoon liittyvät asiat. TiliJaskassa yrittäjän on myös mahdollista aktivoida itselleen oma yritystili ja näin kaikki yrityksen talousasiat hoituvat helposti ja yrittäjä voi keskittyä vain yrittämiseen. Jokainen yritystiliasiakas saa virtuaalisen ja halutessaan myös fyysisen maksukortin käyttöönsä. TiliJaskassa kirjanpidon on tarkoitus olla mahdollisimman helppoa ja käyttäjäystävällistä. Prosesseja pyritään automatisoimaan mahdollisimman pitkälle ja näin

helpotetaan yrittäjän työtä. TiliJaskassa on eri asiakastasoja. Asiakastaso riippuu siitä, kuinka monta tilitapahtumaa yrityksellä on kuukaudessa ja haluaako yrittäjä hoitaa kirjanpitonsa itse ohjelmistossa. Yrittäjän on mahdollista myös ottaa itselleen Premium-asiakkuus, jolloin hänen ei tarvitse tehdä muuta kirjanpitonsa eteen, kuin liittää tilitapahtumiinsa tositteet ja Talenomin kirjanpitäjä huolehtii yrityksen kirjanpidosta.

TiliJaskassa avainpalveluja ovat mm. automatisoidut tiliöinnit, yritystili, myynti- ja ostolaskutus, verkkolaskutus, palkanlaskenta, yhteys omaan pankkitiliin, virtuaalinen/fyysinen maksukortti, tulos/tase laskelmat sekä asiakaspalvelu-chat. (TiliJaska, n.d.)

Kuva 1 TiliJaskan etusivu



4.2 Sopivat testausmenetelmät

Taloushallinto-ohjelmiston testauksessa täytyy ottaa huomioon paljon erilaisia asioita. TiliJaskassa täytyi tarkasti miettiä testaamista ja testausmenetelmiä. Testaamiseen tarvittiin testiyritys, jolla kirjautua palveluun ja oli myös jotain toimintoja, joihin oli vaikea keksiä testausmenetelmiä. Sellainen toiminto oli esimerkiksi TiliJaskaan rekisteröityminen sillä rekisteröitymiseen tarvittiin oma yritys tai toiminimi y-tunnuksella ja vahva kirjautuminen. Kaikilla ei ole omaa yritystä, joten näin ollen kaikki eivät voi rekisteröityä. Rekisteröitymisen testaamiseen täytyi suunnitella muita

tapoja. Toinen hankalasti testattava toiminto oli verkkolaskujen lähettämisen aktivointi. Koska testiyrityksenä oli erään projektissa työskentelevän henkilön oikea yritys, pystyttiin lähettämään verkkolaskuja hänen yrityksellään toiselle projektiin kuuluvalle henkilölle. Verkkolaskujen lähetystä testattaessa täytyi olla äärimmäisen tarkka, että Y-tunnus oli oikein, jotta lasku menivät oikeaan osoitteeseen, eikä kenellekään tuntemattomalle. Verkkolaskutuksen aktivoinnin testaaminen oli myös hankalaa, koska verkkolaskutuksen pystyi aktivoimaan vain kerran yhdelle Y-tunnukselle, joten testaamisessa käytettiin generoituja y-tunnuksia. Aivan loppuun asti tätä toimintoa ei pystytty testaamaan. Oli myös tärkeää vielä todentaa, ettei generaattorilla luotu Y-tunnus ollut oikeasti yhdenkään yrityksen Y-tunnus, tämä pystyttiin todentamaan YTJ-verkkopalvelun kautta. Testausprosessin edetessä vastaan tuli paljon tämänkaltaisia ongelmia ja olikin tärkeää, että oli tarkasti määritelty, mitä kannattaa automatisoida ensin ja miten automatisoidaan.

5 Testitapausten automatisointi TiliJaska-ohjelmistolle

Seuraavissa alaluvuissa kerron TiliJaskaan luotujen testitapausten automatisoinnista ja sen haasteista, sekä huomioon otettavista asioista taloushallinto-ohjelmiston testauksen automatisointia suunnitellessa.

5.1 Testisuunnitelman laatiminen ja käyttö testitapausten automatisoinnissa

Testausprosessin alkaessa määriteltiin ensimmäisenä testisuunnitelma, jonka tarkoitus oli kulkea ja laajentua prosessin mukana. Tärkeää oli priorisoida mitä testataan ja miten testataan, mitä testitapauksia automatisoidaan ja mitä testataan vain manuaalisesti. TiliJaskassa testaamisen haasteena oli käyttäjätapausten puuttuminen, joten ennen testitapausten suunnittelua, koko ohjelmiston toiminnot tuli dokumentoida ja linkittää niihin käyttäjätapaukset. TiliJaska on hyvin laaja ohjelmisto, joten käyttäjätapauksia jo valmiina oleville toiminnoille tuli runsaasti. Käyttäjätapausten laatiminen aloitettiin ensimmäisenä myyntilasku-sivuille, koska se on yrittäjille tärkein toiminto. Sivun käyttäjätapausten valmistuttua oli aika aloittaa luomaan niihin perustuen testitapauksia, jotka todentavat käyttäjätapausten vaatimusten täyttymisen. Kaikki projektin käyttäjätapaukset dokumentoitiin testausprojektimme Teams-tiedostoihin ja Azure DevOps-ympäristöön, missä käyttäjätapauksia on helppo linkittää testitapauksiin ja toiminnallisuuksiin.

Testitapaukset luotiin Given/When/Then-kaavan mukaan, joka on yleisesti käytetty tapa dokumentoida selkeitä ja ymmärrettäviä, hyväksymistestauksessa käytettyjä testitapauksia. Tässä kaavassa on olennaisena ideana jakaa testiskenaario kolmeen osaan: Given kuvastaa tilaa, missä aloitetaan testin suorittaminen, sitä voisi myös kutsua testin ennakkoehdoksi. When kuvastaa toimintaa, mikä pitää suorittaa testitapauksessa. Lopuksi “Then” kuvastaa lopputulosta, missä kuvaillaan muutokset, joita odotetaan toiminnan tulokseksi. Seuraavaksi kirjoitan esimerkin havainnollistamaan testitapausten kaavaa. Ajatellaan, että meidän pitäisi kirjoittaa testitapaus siirtymisestä TiliJaskan internet-sivuille.

Seuraava esimerkki on kirjoitettu englanniksi, koska organisaatiomme on kansainvälinen ja näin ollen kaikki testitapaukset on kirjoitettu englanniksi ja näin esimerkki havainnollistaa mahdollisimman hyvin tehtyä työtä.

Esimerkiksi sivustolle siirtymisestä voidaan luoda seuraavan kaltainen testitapaus:

- **GIVEN** that we have Google Chrome open
- **WHEN** we input www.TiliJaska.fi to the searchbox and we press enter
- **THEN** the homepage of TiliJaska opens and we are able to log in.

Testitapausten automatisointi oli myös selkeä suorittaa Given/When/Then-kaavalla luotujen testitapausten mukaan. Tarkoituksena oli toiminto toiminnolta priorisoida ja valita TiliJaskan avaintoimintoja automatisoitaviksi. Automatisoinnin priorisoinnissa keskityttiin ensin toimintoihin, jotka vaativat toistuvaa ja jatkuvaa testausta ja näin ollen manuaalisesti testattuina veivät eniten aikaa. Tämän kaltaisten TiliJaskan perustoimintojen automatisointi oli yksinkertainen toteuttaa ja näin ollen automatisoinnista saadaan tuottavaa mahdollisimman nopeasti.

5.2 Testausteknologian valinta

Testausteknologiaksi valittiin Robot Framework, koska se on erityisesti suunniteltu hyväksymistestaukseen, mutta se on myös erittäin monipuolinen ja laaja kokonaisuus, joten testausta on mahdollista laajentaa myöhemmin sisältämään myös muita testistrategioita käyttäen samaa teknologiaa. Teknologian valintaan vaikutti myös integroimisen helppous ja monipuolisuus. Robot Framework on tunnettu ohjelmisto ja on integroitavissa helposti moniin kehitysympäristöihin. Robot Framework on myös hyvin tuettu ja sitä päivitetään aktiivisesti. Monet alan johtavat yritykset käyttävät sitä ohjelmistokehityksessä.

Ohjelmistossa käytetään Python-ohjelmointikieltä ja siihen on helppo saada laajennuksia Pythonilla, Javalla tai muilla ohjelmointikielillä toteutetuilla kirjastoilla. Robot Frameworkin käyttö on myös ilmaista, joten sitä käytettäessä ei tarvitse huolehtia lisenssikuluista.

5.3 Automatisoitujen testien luominen

Testitapausten dokumentoinnin valmistuttua aloitettiin luomaan valituille tapauksille automatisointia. Tapausten automatisointi sinällään ei ollut vaikeaa, vaan niin kuin teorialuvuissa kävimme läpi, eniten työtä on vaatinut tapausten suunnittelu ja priorisointi. Käytännössä aluksi mietittiin, mitkä kirjastot ovat tarpeellisia tapauksen automatisointiin, ja liitettiin ne sitten skriptitiedostoon. Tärkeää oli myös, että muuttujille suunniteltiin kuvaavat nimet ja koodi oli selkeää, jotta kuka vain muukin projektin jäsen saattoi sitä ymmärtää.

Automatisoinnin edetessä huomattiin, kuinka äärettömän tärkeä hyvin dokumentoitu testitapaus oli. Ohjelmiston toiminnon logiikkaa ei tarvinnut enää alkaa uudestaan miettiä samalla, kun luotiin automatisointia. Työ aloitettiin järjestelmällisesti, priorisointien mukaan ja samalla tuli ilmi paljon uusia asioita automatisoinnista ja Robot Frameworkin teknologiasta.

Robot Framework -testitapausten luomiseen käytetään valmiita avainsanoja, jotka saadaan koodiin sisällytetyistä kirjastoista. Selenium-kirjasto sisältää selaimessa tapahtuvaan testaamiseen tarvittavia avainsanoja ja se oli yksi kirjastoista mitä käytettiin tässä projektissa testitapausten automatisointiin. Mahdollista oli luoda automatisoidut testitapaukset Given/ When/Then-kaavan mukaan myös Robot Frameworkissa.

Testiskriptiä luodessa pystyin upottamaan kaavan koodiin ennen skriptin avainsanaa ja tällöin Robot Framework ohittaa sen. Näin pystyin luomaan testiskriptejä täydellisesti, testisuunnitelmaan dokumentoitujen testitapausten mukaisesti.

Kuvassa 2 on esitettynä avainsanapohjaisten testiskriptien luontia Robot Framework ympäristössä. Kuvassa 3, on esitettynä TiliJaskan tehtävälista, johon ensimmäisen kuvan avainsanat liittyvät.

Kuva 2 Avainsanat

```
***Test case***
Check that the "Tehtävälista" works on the dashboard
   Given we are logged in to TiliJaska
   when we checking dashboard contains "Tehtävälista"
   Then we can click each button on tasklist and it shows your undone tasks and direct you to the right page
```

Kuva 3 Tehtävälista TiliJaskassa

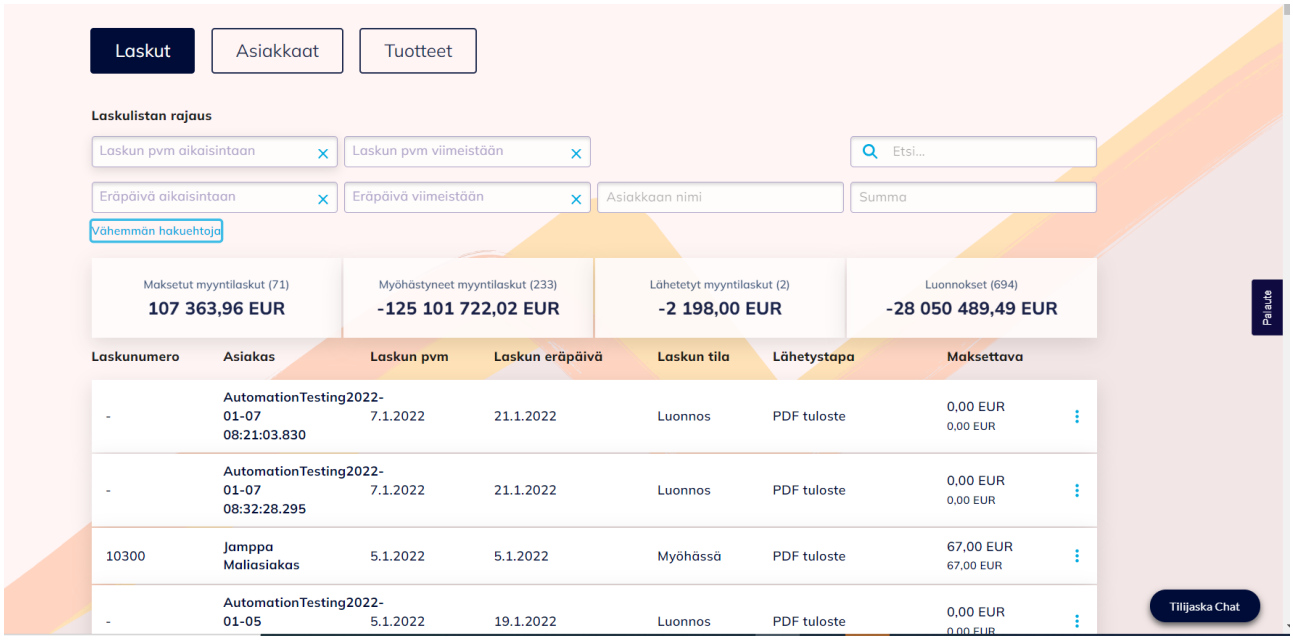
Tärkeimpiä testien automatisointeja oli myyntilaskusivun testitapausten automatisointi. Projektin aikana saatiin osa myyntilaskusivujen testitapauksista automatisoituja. Myyntilaskusivuille automatisoitavia tapauksia on suunniteltu yhteensä 17 kappaletta ja kaikkien automatisointi vie aikaa.

Tärkein valmiiksi saatettu automatisointi oli myyntilaskun luominen yritysasiakkaalle. Tässä tapauksessa luodaan myyntilasku ja tarkistetaan, että laskun lopputuloksessa näkyvät tiedot ovat samat mitä on syötetty laskun teko vaiheessa. Laskulta tarkistetaan myös, että lopullinen hinta vastaa laskulle määriteltyä ALV-verokantaa.

Toinen tärkeä automatisoitu tapaus oli laskujen hakutoiminto. Saamme automaattisesti tarkistettua, että laskulistaa pystyy lajittelemaan, sekä laskuja pystyy hakemaan eri kriteereiden mukaan listalta.

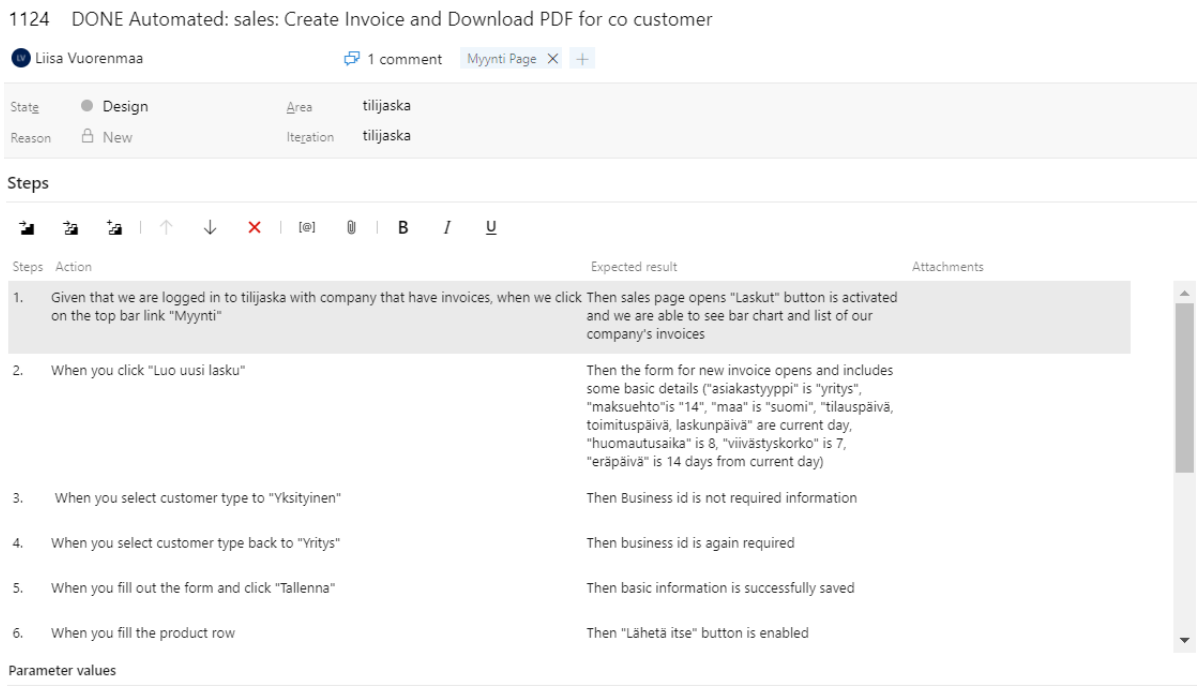
Kuvassa 4 on myyntilasku osion etusivu. Kuvassa näkyy myyntilaskujen hakutoiminnot, joiden testaus saatiin automatisoitua projektin myötä.

Kuva 4 Myyntilaskusivu



Kuva 5 on kuvankaappaus Azure DevOpsista. Kuvassa 5 näkyy testisuunnittelma, jonka pohjalta luotiin automatisoidut testitapaukset myyntilaskun luomiseksi. Kuvassa 6 näkyy kuvan 5 suunnitelman pohjalta luodut avainsanat. Kuvassa 7 näkyy testiskriptejä, joita avainsanat sisältävät.

Kuva 5 Testisuunnitelma DevOps ympäristössä



Kuva 6 Testisuunnitelman mukaisesti luodut avainsanat

```
3]: ***settings***
    Resource  create_and_search_invoices.resource.ipynb

]: ***Test cases***
   Create Invoice and Download PDF for co customer

      Given that we are logged in to tilijaska with company that have invoices
      When we click on the top bar link "Myynti"
      Then sales page opens "Laskut" button is activated and we are able to see bar chart and list of our company's invoices
      When we click "Luo uusi lasku"
      Then the form for new invoice opens and includes some default details
      When we select customer type to "Yksityinen"
      Then business id is not required information
      When you select customer type back to "Yritys"
      Then business id is again required
      When you fill out the form and click "Tallenna"
      Then basic information is successfully saved
      When you fill the product row
      Then "Lähetä itse" button is enabled and price is correct compared to used sales account
      When you click "Esikatsele PDF"
      Then PDF invoice is downloaded to your computer and includes information that you entered and your company's contact information

   Log | Report
```

Kuva 7 Avainsanojen testiskriptit

```
Given that we are logged in to tilijaska with company that have invoices
   Open Browser    ${url}    chrome


   sleep   5s
   input text   xpath://*[@id="username"]   ${username}
   input text   xpath://*[@id="password"]   ${password}
   Click button   xpath://*[@id="login"]

   sleep   10s

   wait until page contains   Tehtävälista
   wait until page contains   Rahavirrat
   wait until page contains   Avainluvut
   wait until page contains   Pankkitili
   sleep   6s


   #Check you are log in with testing company and switch it if necessary
   ${currentCompany}=   get value   class:MuiInputBase-input
   Run keyword if   '${currentCompany}'!= 'OnniManni Testaus Oy'   switch to onnimanni
    sleep   6s

   ${currentCompany2}=   get value   class:MuiInputBase-input

   #Check company is onnimanni
   should be equal   ${currentCompany2}   OnniManni Testaus Oy
   set global variable   ${currentCompany2}

   #get business id an vat id from settings page for later use
   click element   CSS:#topbar-item-settings
   sleep   5s
   wait until page contains   Asetukset
   #Get text from right side of information card on the settings page
   ${infoCardTextRight}=   get text   CSS:div.MuiGrid-grid-sm-12:nth-child(2)
   ${currentBusinessId}=   get line   ${infoCardTextRight}   1
   ${getMyVatId}=   split string   ${currentBusinessId}   -
   ${currentVatId}=   set variable   FI${getMyVatId}
   set global variable   ${currentBusinessId}
   set global variable   ${currentVatId}
   log   ${currentBusinessId}
   log   ${currentVatId}
When we click on the top bar link "Myynti"

   Click link   xpath://*[@id="topbar-item-sales"]
    sleep   4s

Then sales page opens "Laskut" button is activated and we are able to see bar chart and list of our company's invoices

   page should contain  Myyntilaskut

   #Chart element on the sales page
   page should contain element   CSS:.MuiContainer-maxWidthLg

   sleep   10s
```

5.4 Taloushallinto-ohjelmiston testitapausten automatisoinnin haasteet

Testitapauksien luomisen ja teknologian valinnan jälkeen täytyi miettiä seuraavanlaisia kysymyksiä: Mitkä tapauksista kannattaa automatisoida? Minkä tapauksen automatisoinneista olisi eniten hyötyä, jotta testaus olisi tuloksellista? Mitä automatisoidaan ensin ja miksi?

Projektitiimimme oli yhtä mieltä siitä, että TiliJaska-ohjelmiston tärkein ominaisuus asiakkaille kirjanpidon lisäksi oli myyntilaskutus. Ohjelmiston kirjanpito-ominaisuuksissa ei ole havaittu virheitä tai saatu negatiivista asiakaspalautetta lukuun ottamatta muutamia pieniä ei-toiminnallisia bugeja, jotka eivät vaikuttaneet mitenkään kirjanpito-ominaisuuden käytettävyyteen, joten prioriteettilistan korkeimpana oli ohjelmiston myyntilaskutusosio.

Myyntilaskutusosiosta oli tullut asiakaspalautetta jonkin verran. Palaute koski lähinnä ohjelmiston jäämistä jumiin tai sitä, että ohjelmisto ohjasi asiakkaan virhesivulle jonkin tietyn toiminnon jälkeen. Koska näistä asioista tuli palautetta, niin näihin asioihin pureuduttiin myös testauksessa ensin, jotta asiakastyytyväisyys saatiin pidettyä korkealla ja käytettävyys hyvänä. Kuten aiemmin totesimme, niin kaikkea ei kannata automatisoida, mutta myyntilaskutus olisi yksi tärkeä osio, jonka perustoiminnot oli tarkoitus saada automatisoitua, mukaan lukematta verkkolaskutuksen aktivointia ja lähetystä.

Voidaan siis todeta, että suurin haaste taloushallinto-ohjelmiston automatisoinnissa oli valinta, mitä automatisoidaan. Myös maksutilin palveluita, kuten rahansiirtoa olisi hankala automatisoida samalla tekniikalla kuin muuta sivustoa, joten päädyttiin tulokseen, että parasta olisi automatisoida rahaliikenteeseen liittyvät asiat käyttämällä request-kirjastoa. Request-kirjasto on erityinen kirjasto, jonka avulla voi lähettää HTTP-pyyntöjä helposti ilman, että tarvitsee manuaalisesti lisätä merkkijonoja.

6 Projektin lopputulos ja pohdinta

Projektin lopputuloksena syntyi kattava dokumentointi ohjelmiston toiminnoille sisältäen kaikki toiminnot ja niihin liittyvät käyttäjätapaukset, joita tietysti vieläkin lisätään ohjelmiston koko ajan kasvaessa. Sain luotua ohjelmistolle tarkasti dokumentoidut testitapaukset linkitettyinä DevOps-ympäristössä käyttäjätapauksiin. Kaikki dokumentaatio on koottu Azure DevOps-ympäristöön, jossa ne ovat jokaiselle ohjelmiston kehittäjälle helposti näkyvissä ja käytettävissä. Projektissa syntyi myös testiautomaatio osalle TiliJaska-ohjelmiston avaintoiminnoista, ja testiautomaation rakennus jatkuu edelleen tämän opinnäytetyön jälkeen tarkoituksena kattaa testistrategian mukaisesti kaikki ohjelmiston osat, joiden testaus on päätetty automatisoida. Myöhemmin testiautomaatiota on tarkoitus laajentaa myös ohjelmiston muihin osiin, käyttäen erilaisia testausmenetelmiä.

Tärkeimpiä projektin aikana valmistuneita automatisoituja testitapauksia oli TiliJaskaan kirjautuminen, etusivun toiminnallisuuksien testaaminen, myyntilaskun luonti- ja hakutoiminnon testaaminen.

Testausprojektin edetessä minun käsitykseni testaajan työnkuvasta on laajentunut ja olen myös käsittänyt paremmin, kuinka suuri prosessi systemaattinen testaus ohjelmistokehityksessä voi olla. Projektin aikana on tullut vahvasti esille, miten hyödyllistä tämän taloushallinto-ohjelmiston testitapausten automatisointi on ja kuinka automatisoinnilla voi säästää paljon vaivaa ja aikaa, varsinkin toistuvissa ja jatkuvissa toimenpiteissä.

Lopputuloksena toteaisin, että taloushallinto-ohjelmiston testaus on hyödyllistä automatisoida, kunhan muistetaan olla olettamatta, että automatisoimalla löydettäisiin ohjelmiston kaikki virheet. On parempi ajatella niin, että automatisointi vain todentaa mitkä käyttäjäskenaariot toimivat testitapaukseen asetettujen vaatimusten mukaan. Testauksen automatisoinnin hyödyt yritykselle ovat ilmeiset. Vaikka itse prosessi tuottaa kuluja ja vie resursseja, se maksaa itsensä takaisin. Mitä aikaisemmin ohjelmistossa oleva virhe löytyy, sitä kustannustehokkaampaa ja helpompaa sen korjaaminen on. Testausprojekti tulee edetessään ja kasvaessaan hyödyttämään

toimeksiantajayritystäni yhä enemmän ja enemmän mm. säästämällä aikaa ja resursseja, ja mahdollistamalla kehittäjille jatkuvan ja nopean palautteen saannin työstään.

Projekti toimeksiantajalleni onnistui kaiken kaikkiaan hyvin. Projekti jatkuu kuitenkin edelleen ja testitapaukset kasvavat kasvamistaan, ja hiomme ja päivitämme myös aktiivisesti vanhoja testitapauksia ja testiautomaatiota. Olen vakituisessa työsuhteessa toimeksiantajayrityksessäni ja työtehtäväni on viedä tätä testausprojektia eteenpäin.

Olen saanut projektista hyvää palautetta ja projektitiimimme kanssa mietimme aktiivisesti, miten voisimme vielä parantaa ja tehostaa projektia. Testausprojektissa parasta on ollut sen monipuolisuus ja laajuus. On myös innostavaa huomata, kuinka automaation kautta pystyn tehostamaan omaa työtäni merkitsevästi ja todentamaan virheitä ohjelmistossa.

Esimieheni mukaan projekti on ollut liiketaloudellisesti hyödyllinen ja on ollut yritykselle erittäin tarpeellinen. Olen myös toiminut hyvässä hengessä ja aktiivisesti muiden tiimin jäsenten kanssa. Tiimissä käytetään paljon myös englannin kieltä, minkä käytössä olen esimieheni mukaan kehittynyt paljon jo projektin aikana. Tultuani harjoittelijaksi tiimiin, yritys on ottanut muitakin Hämeen Ammattikorkeakoulun opiskelijoita harjoitteluun.

7 Yhteenveto

Tässä opinnäytetyössä kuvaamani testausprojektin myötä olen löytänyt vastaukset tutkimuskysymyksiini. Projektin kuluessa oli erittäin selkeää ja tuli konkreettisesti esille, että taloushallinto-ohjelmiston testaaminen on mahdollista ja kannattavaa yritykselle. Testaamisen automatisoinnista on yritykselle monia hyötyjä ja vaikka taloushallinto-ohjelmiston kanssa toimiessa tulee olla erittäin tarkka asiakastietojen ja rahaliikenteen vuoksi, automatisoitu testaaminen voidaan toteuttaa.

Projektin edetessä olen oppinut paljon testaamisesta ja testauksen automatisoinnista ja minulle on myös avautunut, kuinka laaja ja tärkeä osa testaaminen on ohjelmistokehitystä. Olen opetellut myös uusia teknisiä taitoja liittyen automatisointiin ja oppinut käyttämään minulle ennestään tuntemattomia kirjastoja Robot Frameworkissa.

Myöskään Azure DevOps-ympäristö ei ollut minulle ennestään tuttu, mutta tämän projektin myötä olen tutustunut siihen ja oppinut paljon uusia hyödyllisiä asioita myös projektin hallinnasta DevOps-ympäristössä.

Toisin kuin aiemmin opiskeluni aikana, jolloin kiinnostukseni kohdistui eniten automatisointiin ja backend-ohjelmointiin, nyt opinnäytetyöni myötä olen kiinnostunut ja innostunut erittäin paljon laadun valvonnasta, DevOps-ympäristöjen hallinnasta ja ohjelmistojen välisistä integroinneista. Olen oppinut ymmärtämään paremmin suurempaa ohjelmistokokonaisuutta ja opin työni ohella koko ajan lisää ohjelmistokehityksestä. Jatkuva oppiminen on minulle erittäin innostavaa ja minulla on erittäin suuri motivaatio jatkaa tämän opinnäytetyöni myötä itseni kehittämistä ohjelmistotestauksen ja automatisoinnin asiantuntijaksi.

LÄHTEET

Mili, A. & Tchier, F. (2015). Software Testing: Concepts and Operators. John Wiley & Sons. https://books.google.fi/books?hl=fi&lr=&id=cSpzCQAAQBAJ&oi=fnd&pg=PR3&dq=software+testing+techniques&ots=GMZAO3Uyhm&sig=ZLO0jUnttkdrz_BI7vqoivc2nQk&redir_esc=y#v=onepage&q=software%20testing%20techniques&f=false

ATR Soft Oy. Testausautomaatio tehostaa laadunvarmistusta. Haettu 31.10.2021. https://www.atrsoft.com/2018/02/12/testausautomaatio-tehostaa-laadunvarmistusta/

Abildskov. J. Mitä on DevOps. Haettu 19.01.2022. https://www.eficode.com/fi/blog/mita-on-devops

Singh, S. K. & Singh, A. (2019). Psychology of Testing. https://books.google.fi/books?hl=en&lr=&id=HdKwDwAAQBAJ&oi=fnd&pg=PT3&dq=+Software+testing&ots=79fGl_sCc0&sig=-CgS17F4SzNKNSjcUvxeYLP_I6s&redir_esc=y#v=onepage&q=Software%20testing&f=false

Gelperin, Hezel. 1988. The growth of software testing. s.687 –695. Haettu 31.10.2021. https://www.researchgate.net/publication/234808293_The_growth_of_software_testing

Ghahrai, A. Päivitetty 11.3.2020. Problems with Test Automation and Modern QA. Haettu 31.10.2021. https://devqa.io/problems-test-automation-modern-qa/

Hamilton, T. 8.10.2021. TEST PLAN: What is, How to Create (with Example). https://www.guru99.com/what-everybody-ought-to-know-about-test-planing.html

Kasurinen, J. (2013). Ohjelmistotestauksen käsikirja. Docendo, Jyväskylä: Saarijärven Offset Oy.

Matti Luukkainen, Ohjelmistotuotanto avoin yliopisto. 2021. Haettu 31.10.2021. https://ohjelmistotuotanto-hy-avoin.github.io/osa3/

Pyhäjärvi, M. & Pöyhönen, E. 2014. Testauksen Automatisointi. (Haettu 13.12.2021). https://slideplayer.fi/slide/2723180/

Vuori, M. 6.10.2010. Ketterä testaus ja testaus ketterässä ohjelmistokehityksessä. Haettu 31.10.2021. https://www.mattivuori.net/julkaisuluettelo/liitteet/kettera_testaus.pdf

Ohjelmistotestauksen perusteet, versio 1.0. s. 20-22. (Haettu 31.10.2021) https://docplayer.fi/2677930Ohjelmistotestauksen-perusteet-versio-1-0.html

Mensio, O. IBM Rational Software. 2005. Testauksen automatisointi on haasteellista mutta palkitsevaa. http://www.pcuf.fi/sytyke/lehti/kirj/st20051/ST051-08A.pdf

Pohjolainen, P. 2003. Ohjelmistotestauksen automatisointi s. 23-34. Haettu 1.12.2021. https://docplayer.fi/3477676-Ohjelmiston-testauksen-automatisointi.html

Pohjolainen, P. 2003. Ohjelmiston testauksen automatisointi. Hakupäivä 27.10.2021, http://www.cs.uku.fi/tutkimus/Teho/PenttiPohjolainen_Gradu.pdf

Robot Framework. Haettu 13.12. https://robotframework.org/

Selenium. Selenium – Web Browser Automation. Haettu 1.12.2021. https://www.selenium.dev/documentation/

Software Testing Fundamentals. (2020). Test Plan. Haettu 21.10.2021 osoitteesta. https://softwaretestingfundamentals.com/test-plan/

Softwaretest professonials. 2021. Testing-automation. http://www.softwaretestpro.com

Taloushallintoliitto. Tilitoimiston asiakkaalle. Haettu 13.12.2021. https://taloushallintoliitto.fi/tilitoimistoasiointi/tilitoimiston-palvelut

Testauksen osaamisyhteisö (TestausOSY). Testiautomaatio on kuormitustestauksen kaveri. https://www.sytyke.org/wp-content/uploads/2019/01/LT-Vol1Ed2.pdf

TiliJaska, (Haettu 13.122021). www.tilijaska.fi

Kautto, T. 21.11.1996. Ohjelmistotestaus ja siinä käytettävät työkalut. Haettu 1.12.2021. http://www.mit.jyu.fi/opiskelu/seminaarit/ohjelmistotekniikka/testaus/

YP. Kirjanpito-ohjelma. Haettu 13.12.2021. https://yrityksen-perustaminen.net/kirjanpito-ohjelma/

Liite 1: Aineistonhallintasuunnitelma

Opinnäytetyössäni käytettävä tutkimusmateriaali on hankittu, lähdeluettelossani viittaamistani lähteistä, toimeksiantaja yrityksen tietovarastoista ja omien kokemusteni kautta.

Käytän yritykseltä saamaani materiaalia, tutkimuskysymyksiin vastatessani ja suorittaessani työni käytännön osuutta.

Tutkimuskysymyksieni vastaus perustuu osittain yrityksen kautta saamaani tekniseen materiaaliin ja osittain asiakaspalaute materiaaliin, jota käsittelen anonyymisti.

Materiaalia säilytän teoria osuuden tutkimusaineiston osalta omalla tietokoneellani, yritykseltä saamieni tietojen osalta työssä käyttämälläni tietokoneellani, jota säilytän yrityksen tiloissa.

Aineistoa säilytetään opinnäytetyön tekemisen aikana kahdella tietokoneella, joissa on molemmissa virus suojaus ja salasana suojaus.

Opinnäytetyö julkaistaan myöhemmin digitaalisessa opinnäytetyön Theseus- palvelussa.

Opinnäytetyön käytännön osuudessa aikaansaamani materiaali jää yrityksen käyttöön, missä sitä voidaan jakaa yrityksen omissa kanavissa. Toimeksiantaja yrityksellä, Talenom Software Oy:lla on opinnäytetyöhön omistus-, käyttö- ja tekijänoikeus työn valmistuttua.

Opinnäytetyöni valmistuttua, säilytän omalla koneellani olevaa tutkimus materiaalia yhden vuoden, jonka jälkeen hävitän sen poistamalla kaikki tiedostot asianmukaisesti.