SAVONIA
ammattikorkeakoulu

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

AUTOMATISOITU UI-TESTAUS SELENIUMILLA

Työ tehty Visma Enterprise Oy:lle

TEKIJÄ Teemu Luukkanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä Teemu Luukkanen	
Työn nimi Automatisoitu UI-testaus Seleniumilla	
Päiväys 20.12.2022	Sivumäärä 20
Toimeksiantaja Visma Enterprise Oy	

Opinnäytetyön tarkoitus oli tuottaa Seleniumilla automatisoituja UI-testejä Assi-ohjelman Suomessa luotuihin osiin. Tavoitteena oli luoda selkeät, toimivat, toistettavissa ja helposti laajennettavissa olevat Selenium testit UI-elementeille sekä perehtyä automaattisen testauksen hyötyihin ja haittoihin. Opinnäytetyö tehtiin Visma Enterprise Oy:lle.

Opinnäytetyö toteutettiin toiminnallisena kehitystyönä, tarkemmin emansipatorisena toimintakehityksenä. Työssä luotiin Seleniumilla automatisoituja UI-testejä sekä tietokanta sisältöineen. Näiden tarkoitus oli vähentää manuaalista testausta sekä säästää työntekijöiden aikaa ja sitä kautta yrityksen rahaa.

Opinnäytetyön tuloksena syntyi 16 testiä, joilla pystyttiin testaamaan Suomessa tehtyjä Assi-ohjelman muutoksia. Lisäksi luotiin abstraktiluokka TestUtils, joka sisälsi suuren määrän apumetodeja, joita käytettiin lähes jokaisessa testissä. Aikataulullisista syistä toteutettu testimäärä jäi odotettua pienemmäksi. Opinnäytetyö tarjosi kuitenkin hyvän pohjan uusien testien luomiselle ja luotu abstraktiluokka sisälsi lähestulkoon kaikki jatkossakin tarvittavat apumetodit. Luodun tietokannan ansiosta testattava data säilyi identtisenä, jolloin testien toistettavuus pystyttiin takaamaan.

Avainsanat:
UI-testaus, Selenium, testauksen automatisointi

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport	
Degree Programme Master's Degree Programme in Engineering Knowledge Management	
Author Teemu Luukkanen	
Title of Thesis Automated UI-testing with Selenium	
Date 20 December 2022	Pages 20
Client Organisation Visma Enterprise Oy	

The purpose of the thesis was to produce automated UI-tests for the Finnish modifications on Assi-program using Selenium. The goal was to create clear, working, repeatable and easily expandable Selenium tests for UI-elements and learn about the pros and cons of automated testing. The thesis was made for Visma Enterprise Oy.

The thesis was decided to be done as an emancipatory action development. Automated tests and a database with information were created for the thesis. The purpose was to reduce the amount of manual testing and save work hours and money.

As a result of the thesis, 16 tests were created that could test the features made in the Finnish Assi-program. An abstract class called TestUtils was also made. It included many helper methods that were used in almost all tests. The achieved number of tests was lower than originally anticipated because of time issues. The thesis provided a good starting point to build new tests on and the created abstract class included all the necessary helper methods for the near future. The created database ensured that the test data remained identical on every test run and so provided the repeatability for the tests.

Keywords:
UI-testing, Selenium, test automation

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön tilaajana toimi Visma Enterprise Oy. Visma Enterprise tarjoaa asiakkailleen monipuolisesti henkilöstö- ja työvoimanhallintaratkaisuja sekä opetushallinnon sähköisiä palveluita. Visma Enterprisen tuotteisiin kuuluvat koulumaailman Wilma, Primus ja Kurre, henkilöstöhallinnan Saima HR, työvoimahallinnan Numeron ja Aikajana, työajanseurannan Tiima sekä matka- ja kululaskujen hallinnan M2. Assi, jonka käyttöön opinnäytetyö tehtiin, on Wilmasta, Primuksesta ja Kurresta erillinen palvelu, josta on Primukseen rajapinta oppilastietojen hakuun. Se on selainkäyttöinen järjestelmä koulukuraattoreiden ja -psykologien, sekä muiden opiskelu- ja oppilashuollon ammattilaisille. Assiin voidaan kirjata asiakaskertomuksia ja muistiinpanoja, luoda asiakirjoja, sekä koostaa tilastoja turvallisesti ja kootusti yhdessä paikassa. Kirjaukset voivat olla sekä yksilöllisen opiskeluhuollon kirjauksia että yhteisöllisiä tapauksia.

Opinnäytetyön tarkoituksena oli tuottaa Seleniumilla automatisoituja UI-testejä Assin Suomessa luotuihin osiin. Testit toimivat myös jatkosovellusalustana uusille testeille. Ajatus opinnäytetyön aiheesta syntyi Assin tiimin yhteisessä kehityspalaverissa.

Assi on kopio Norjan Vismassa käytössä olevasta Flyt-ohjelmasta, jota kehitetään Norjassa koko ajan. Assiin tuodaan Flytistä noin kuukauden koodikehitys kerrallaan ja tuotu koodi mergetään Assiin. Mergeprosessilla tässä tapauksessa tarkoitetaan sitä, että Norjan Flyt-tiimin tekemä koodi yhdistetään Assi-ohjelmaan. Mergeprosessi aiheuttaa konflikteja, jos samoihin tiedostoihin on tehty muutoksia. Kun konfliktit ovat korjattu, Assin toiminnallisuudet testataan manuaalisesti. Virheitä voi löytyä paljon. Ja kun löydetyt virheet ovat korjattu, tehdään uusi manuaalinen testikierros. Testikierroksia toistetaan, kunnes ei enää löydetä uusia virheitä. Prosessi on paitsi aikaa vievää, myös puuduttavaa. Sen lisäksi Assi on silti jatkuvasti Flytiä kuusi kuukautta kehityksessä jäljessä.

Esitin idean, että voisin opinnäytetyönä luoda automatisoidun testausympäristön. Testausympäristössä testattaisiin tiettyjä Assin toiminnallisuuksia. Näin testikierroksiin menevä aika vähenisi huomattavasti ja resurssit vapautuisivat muuhun työhön ja saisimme Norjan kiinni aikatauluissa. Onnistuessaan opinnäytetyö nopeuttaisi prosessia huomattavasti. Tavoitteena oli luoda selkeät, toimivat ja helposti laajennettavissa olevat Selenium testit UI-elementeille.

Opinnäytetyö on toiminnallinen kehitystyö. Toimintatutkimuksella (Action Research, AR) tarkoitetaan Melrosen (2001) mukaan *”Todellisessa maailmassa tehtävää pienimuotoista interventiota ja kyseisen intervention vaikutusten lähempää tutkimista. Sen avulla pyritään ratkaisemaan erilaisia käytännön ongelmia”.* Toisin sanoen sen avulla yritetään löytää ratkaisu tietyissä tilanteissa havaittuun ongelmaan. Toimintatutkimus voi olla tekninen tutkimus, praktinen tutkimus tai emansipatorinen tutkimus. Teknisessä tutkimuksessa kehittämisprojektin käynnistää joku ulkopuolinen. Praktisessa tutkimuksessa työtekijöitä opastetaan, kuinka tiedostetaan ja muotoillaan sekä uudelleensuunnataan tietoisuuttaan sekä käytäntöä. Emansipatorisessa tutkimuksessa Metsämuurosen (2003, s. 181–182) mukaan, toimija itse pyrkii parantamaan toimintaympäristöään. Toimin itse sisäpiiriläisenä, joten sanoisin tutkimukseni olevan ennen kaikkea emansipatorinen toimintakehitys, jonka tarkoituksena on kehittää tällä hetkellä ei niin toimivaa toimintatapaa paremmaksi.

Toimintasuunnitelmani oli seuraavanlainen. Luodaan UI-testit ja ajetaan ne läpi. Tämän jälkeen reflektoidaan ohjaajille ja tehdään tarvittavia korjauksia, kunnes sekä minä, että opinnäytetyön tilaaja, ollaan tyytyväisiä lopputulokseen.

Tulevia haasteita välttääkseen, konsultoidaan jo etukäteen työkavereita, jotka ovat tehneet Selenium-testauksia ja selvitetään, miten parhaiten opinnäytetyötä voidaan lähestyä. Opiskellaan paljon opetusmateriaaleja Selenium testauksen toteutuksista, sekä Vismalla jo käytössä olevia Selenium testejä. Vastoinkäymiset ennakoidaan varmistamalla riittävä taustatuki. Taustatukena on kaksi kokenutta seniortason ohjelmistokehittäjää sekä mahdollisuus konsultoida Flyt-tiimin osaajia.

UI-elementtien testaus toteutetaan Seleniumilla. Apuna käytetään featurelistaa, josta saadaan käyttöön kaikki testattavat kohdat. Featurelistalla on noin 30 testiä. Testit tulisi toteuttaa niin, että ne voidaan ajaa useaan kertaan ilman, että testi hajoa. Testien pitää myös toimia ja havaita mahdolliset virheet, sekä ilmoittaa näistä virheistä ymmärrettävällä tavalla, jotta ne voidaan korjata.

Opinnäytetyöni käytännön osuus on yrityssalaisuus. Tämän vuoksi työn liitteeksi ei voida laittaa featurelistaa ja tekemääni koodia voidaan avata vain rajatusti. Työstä kuitenkin pyritään kertomaan mahdollisimman kattavasti oikeanlaisen kokonaiskuvan antamiseksi. Kokonaiskuvan hahmottaminen lisää opinnäytetyön luotettavuutta.

2 SANASTO

Angular	Googlen kehittämä ja ylläpitämä TypeScript-pohjainen ohjelmistokehys, jolla luodaan web ohjelmia
Assertointi	Ehto, joka halutaan toteutuvan
ChromeDriver	Seleniumin käyttämä ajuri Chrome selaimen manipulointia varten
Debuggaus	Vian paikallistaminen ja vian korjaus
EDSAC	Electronic Delay Storage Automatic Calculator eli ensimmäisiä tallennetun ohjelman tietokoneita
Featurelista	Ominaisuuslista
Featureswitch	Toiminnallisuuskytkin, jolla saadaan tietty toiminta päälle tai pois
Komponentti	Ohjelmiston tai laitteiston osa
Konflikti	Eriäväisyys olemassa olevan ja tuottavan koodin välillä
Merge	Prosessi, jossa liitetään kaksi eri versiohallinnan haaraa yhteen
Regressio	Uudelleentestaus
Renderöidä	Tiettyjen kappaleiden tai kuvien esittämistä näytöllä
Spesifikaatio	Täsmällinen, vaatimuksia asettava kuvaus
Tietokanta	Tietojärjestelmän osa, jossa tietoja säilytetään
UI-testaus	Käyttöliittymätestaus
Validaattori	Ohjelma, joka tarkastaa koneellisesti dokumentin oikeellisuuden
Verifiointi	Olettaman varmentaminen

3 OHJELMISTOTESTAUS

3.1 Mitä on ohjelmistotestaus?

Mitä ohjelmistotestaus oikeastaan on? Sen historia on pitkä. Ohjelmistotestausta on ollut yhtä kauan, kun on ollut olemassa ohjelmistoja. Vuonna 1949 Brittiläinen tietojenkäsittelijä Maurice Wilkes totesi näin: *"It was one of my journeys between the EDSAC room and the punching equipment that hesitating at the angles of the stairs the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs."* Ohjelmistojen kasvaessa ohjelmistotestauksessa on tapahtunut huomattavaa kehitystä.

Testaus voidaan määritellä eri tavoin. Se voidaan nähdä testauksen prosessina, jossa pyritään löytämään vahvistus sille, että ohjelma täyttää suunnitellun tarkoituksensa (Hertzel, 1993, s. 4). Toinen määritelmä määrittelee testauksen toiminnaksi, jossa testattavaa järjestelmää suoritetaan löytääkseen siitä virheitä (Myers, 1979, s. 6). Wikipedian määritelmä testauksesta on *"Koe eli jonkin ilmiön, ominaisuuden tms. tutkimiseksi tai määrittämiseksi tehtävä käytännön toimenpide, toimenpidesarja tai testi. Kokeita on monenlaisia."* Ohjelmistotestauksen perinteinen määritelmä on yrittää löytää virheitä suorittamalla ohjelmaa tai sen osaa. Toiminnan on oltava suunnitelmallinen, sillä ilman sitä testaus olisi umpimähkäistä satunnaisten syötteiden kokeilua ja tulokset jäisivät kehnoiksi. Testaus on onnistunut, kun sen avulla testattavasta ohjelmasta löydetään virheitä, jotka poikkeavat ohjelman tarkoitetusta toiminnasta. Spesifikaatio on määritelmä ohjelman oletetusta toiminnasta ja sitä tarvitaan, testien tekemiseen ja niiden oikeellisuuden toteamiseen. (Haikala & Märijärvi, 2004, s. 284,287.)

Yksi lähestymistapa on ajatella, että ohjelmistotestaus lähtee olettamasta, että ohjelmasta löytyy virheitä. Silloin testaus nähdään virheiden löytämisen prosessina. Koska oletetaan, että ohjelmassa on virheitä, testaus voidaan todeta epäonnistuneeksi, jos niitä ei löydy testauksella. (Sandler, Badgett & Myers, 2011, s. 6–7.)

Virheettömien ohjelmien tekeminen on lähes mahdotonta. Kymmenessä tuhannessa koodirivissä arvioidaan olevan vähintään yksi virhe. Tämä kuitenkin riippuu ohjelman iästä. Löytämättömien ohjelmavirheiden osuus voi olla jopa 5 prosenttia. Osan näistä selittää ohjelmien suuri syötemäärä. Eikä ohjelman virheellisen kohdan suorittaminen välttämättä aina aiheuta virhettä. Järjestelmässä voi tulla vika, joka johtuu virheellisen kohdan suorittamisesta. Ja toinen toiminto voi korjata virheen itsestään. (Haikala & Mikkonen, 2011, s. 206.)

Ohjelmistotestaus tarkoittaa Kasurisen (2013, s. 10) mukaan työtä, jolla varmistetaan, että ohjelmistotuote on odotuksia vastaava, ja että kaikki ominaisuudet, jotka ovat saatu valmiiksi, varmasti toimivat tarkoituksen mukaisesti. Testaustyön tarkoitus on tunnistaa kohdat, missä ohjelma ei vastaa suunnitelmaa. Näin varmistetaan, että tehtävä tuote on tehty oikein.

Kasurinen (2013, s. 58–59) esittelee käytännön testaustyötä, jossa on aina mukana kolme tahoa: Testaamisesta vastaava esimies, testaajat ja kehittäjät. Testausprojektin aikana tapahtuu kutakuinkin seuraavien periaatteiden mukaisesti.

- Moniammatillisessa yhteistyössä luodaan testaussuunnitelma ja sen pohjalta ensimmäiset testitapaukset, jotta saadaan projekti käynnistettyä ja sovittua mitä tehdään ja kuka tekee mitäkin.
- Kehittäjät luovat komponentteja ja yksikkötestaukset niille.
- Kehittäjät sekä testaajat kirjoittavat uusia testitapauksia, jotka täydentävät olemassa olevaa testisuunnitelmaa ja testitapausvalikoimaa.
- Testaajat kokeilevat komponentteja luoduilla testitapauksilla, ja mikäli testitapaus epäonnistuu, kirjoittavat niistä ilmoituksen ja arvioivat miten vakava vika on kyseessä.
- Saatujen ilmoitusten pohjalta kehittäjät korjaavat järjestelmästä löytyviä ongelmia, ja samalla jatkavat järjestelmän komponenttien rakentamista.
- Sitä mukaan, kun komponentit valmistuvat, niitä integroidaan järjestelmään. Epäonnistuneesta integrointitestien tapauksista kirjoitetaan uusia raportteja.
- Kun kaikki komponentit on rakennettu ja saatu toimimaan yhdessä, siirrytään järjestelmätestaukseen
- Kun järjestelmätestaus on saatu vaiheeseen, jossa järjestelmässä ei enää ole merkittäviä vikoja siirrytään hyväksymistestaukseen.
- Kun asiakas, hyväksyjä tai vastaanottaja on tyytyväinen järjestelmän toimintaan hyväksymistestauksessa, voidaan se todeta valmistuneeksi
- Kun projekti on saatu valmiiksi, koostetaan siitä loppuraportti ja kaikki käytetyt osat arkistoidaan siltä varalta, että tuotetta joudutaan muuttelemaan, korjaamaan tai täydentämään myöhemmin uudessa projektissa.
  (Kasurinen, 2013.)

3.2 Testausmuodot

Erilaisia testausmuotoja on todella useita. Seuraavissa luvuissa käyn läpi ainoastaan työni kannalta olennaisia testausmuotoja.

3.2.1 UI-testaus

UI-testaus eli User Interface testing, tarkoittaa käyttöliittymätestausta. Sillä testataan ohjelman visuaalisten elementtien toiminnallisuutta ja tehokkuutta. Pääpiirteet UI-testaukselle ovat toiminnallisuus, tehokkuus, käytettävyys, vaatimustenmukaisuus ja ohjelman visuaalinen suunnittelu. Näillä varmistetaan, että käyttöliittymä toimii ja ohjelman komponenteissa ei ole häiriöitä. UI-testaus tarkistaa, miten ohjelma käyttäytyy, kun sitä käytetään näppäimistöllä ja hiirellä. (Iyengar, 2021.)

3.2.2 Automaattinen testaus

Kuten testaukselleki, automaattiselle testaukselle löytyy monia erilaisia määritelmiä. Yksikkötestaus, testilähtöinen ohjelmistokehitys, testiskriptien kehittäminen käyttäen apuna skriptauskieliä, suorituskykytestaus sekä toiminnallinen testaus ovat kaikki käypiä termin määritelmiä. Yleisesti ottaen kaikki testit, jotka voidaan tehdä manuaalisesti, esimerkiksi toiminnallisuus-, suorituskyky- tai stressitestit, voidaan automatisoida. (Dustin, Thom & Nernie, 2009, s. 3–4.) Työssäni tarkoitan automaattisella testauksella, tehtyjen testien ajamista tarvittaessa tarkistamaan, että komponentit toimivat toivotulla tavalla.

Vahvuutena automaattisella testauksella on kyky suorittaa väsymättä aikaa vieviä tai paljon toistoja sisältäviä testejä. Näiden manuaalisesti suorittaminen veisi tolkuttoman paljon aikaa (Sandler, Badgett & Myers, 2011, s. 184).

Dustin esittää kirjassaan, että ehdoton automatisoidun testauksen hyöty on toistettavuuden mahdollisuus. Mikäli testi tehdään manuaalisesti löytäen virheen, myöhemmin tilannetta ei enää ehkä pystytä toistamaan. Dustinin mielestä regressiotestaus tulisi tehdä automatisoidusti, sillä muuten se on ikävystyttävää ja aikaa vievää sekä virhealtista. (Dustin, Thom & Nernie, 2009, s. 43–45.) Kaner on epäilevämpi ja uskoo, että regressiotestauksen suurin ongelma on testien vanhentuminen. Pienetkin ohjelman päivitykset voivat rikkoa testit. Tämän takia virheiden määrä lisääntyy, vaikka ohjelman koodi on täysin toimiva. (Kaner, Bach & Pettichord, 2002, s. 108.)

3.2.3 Manuaalinen testaus

Automaattisella testauksella ei pystytä ratkaisemaan kaikkia testaamiseen liittyviä ongelmia. Joissakin tilanteissa manuaalinen testaus on parempi. Manuaalinen testaus vie vähemmän aikaa kuin automaattisen testauksen toteuttaminen. Toisin sanoen, kun on kiireaikataulu, manuaalinen testaus sopii tilanteeseen paremmin. Manuaalista testausta tehdessä voi myös vahingossa löytyä uusia virheitä, koska ihmiset tekevät pieniä variaatioita testejä suorittaessa. Tietokone ei pysty siihen, koska se ei kykene itsenäiseen ajatteluun, vaan suorittaa testit joka kerta täysin samalla tavalla. Tietokoneen virheentunnistuskyky on myös rajoittuneempi verrattuna ihmiseen. Toisin kuin ihminen, se ei huomaa, jos tietokoneesta kuuluu kummallisia ääniä tai jos kuva on epäselvä tai värisevä. (Kaner, Bach & Pettichord, 2002, s. 99.)

Toisena esimerkkinä manuaalisen testauksen eduksi, voidaan mainita tilanne, jossa testauksen kohteena olisi ohjelmiston käyttöliittymä, joka uudistetaan lähiaikoina. Tällaisessa tilanteessa manuaalinen testaus on parempi vaihtoehto, koska käyttöliittymää uudistettaessa, jo olemassa olevat automaattiset testit, eivät enää sen jälkeen todennäköisesti toimisi. Jatkuva ylläpito on automaattisten testit toimivuuden edellytys. Jos niitä ei ylläpidetä, niistä ei ole enää hyötyä. Yleisesti ottaen, lyhyemmällä aikavälillä manuaalinen testaus voittaa automaattisen testauksen. Pidempi käyttö tuo automaattisen testauksen hyödyn esiin. (Selenium, 2021b.)

3.2.4 Mustalaatikkotestaus

Mustalaatikkotestaus on testausmuodoista perinteisin. Mustalaatikkotestauksessa ei tarkastella, mitä koodissa tapahtuu, vaan katsotaan mitä ohjelma tekee, kun sille annetaan erilaisia syötteitä. Menetelmä on yksinkertainen, ja sitä varten tarvitaan ainoastaan jotain toiminnallisuutta suorittava laite. Mustalaatikkotesteillä nähdään helposti, miten järjestelmä reagoi haitallisiin tai vääriin syötteisiin, kuten arvoasteikon ulkopuolelle jääviin arvoihin. Mustalaatikkotestauksessa on tärkeää, että annetuilla syötteillä pystytään todentamaan, että testattava toiminnallisuus toimii oikein kaikissa odotettavissa olevissa tilanteissa. Tämän vuoksi annettavien syötteiden tulisikin sisältää mahdollisimman laajan yhdistelmän kaikista käytössä olevista vaihtoehdoista, aloittaen kaikista tavallisimmista tapauksista. Koska testaaja ei näe mitä laitteen sisällä tapahtuu, hänen tehtäväkseen jää lähinnä varmistaa, että annetut syötteet ovat oikein ja tarkastaa, että syntyvä lopputulos vastaa toivottua lop-

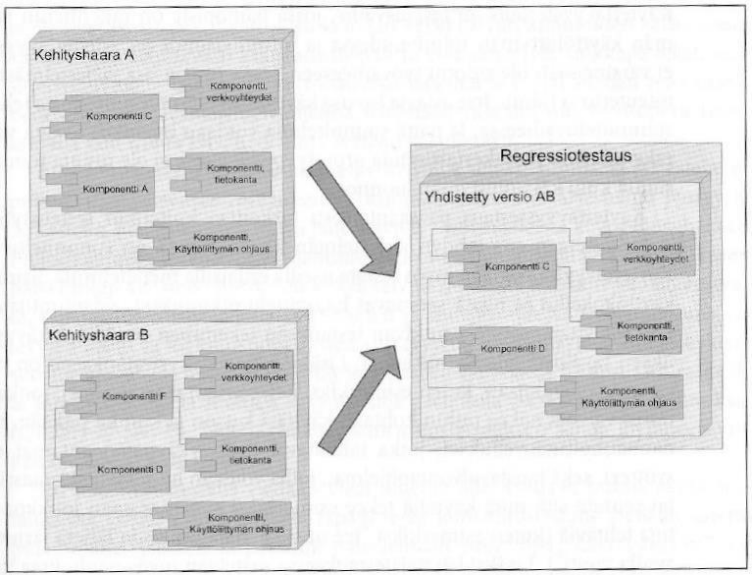
putulosta. (Kasurinen, 2013, s. 64–67; Juvonen, 2018, s. 29.) Käytän työssäni mustalaatikkotestausta, sillä en tarkastele koodin toimivuutta. Oleellista on, että ohjelma toimii käytettäessä ennalta määrätyllä tavalla.

3.2.5 Regressiotestaus

Regressiotestaus on yleistermi, jolla tarkoitetaan uudelleentestaamista, sitä että myös uusi versio toimii oikein. Se ei siis ole varsinaisesti erillinen testausmuoto, kuten yksikkötestaus tai integraatiotestaus eikä menetelmä, kuten mustalaatikkotestaus. Kun toimivan järjestelmän jotain osaa muutetaan, ja muutoksen jälkeen halutaan varmistua siitä, että järjestelmä toimii edelleen oikein, käytetään regressiotestausta. Myös osatavoitteen täyttyessä, voidaan käyttää regressiotestausta varmentamaan, että kaikki toiminnot toimivat edelleen oikein. Myös ne, jotka oli korjattu aikaisemmassa ohjelman versiossa.

Kun jotain ohjelmiston osaa muutetaan, siihen pitää suhtautua kuin se olisi kokonaan uusittu. Suurin osa virheistä löytyy uusista komponenteista tai toiminnoista, jotka käyttävät uusia komponentteja. Kehityshaarojen yhdistämisen jälkeen regressiotestaamalla pystytään todentamaan, että aiemmista osista korjatut virheet eivät esiinny yhdistetyssä versiosta. (Kasurinen, 2013, s. 68–69.)

Regressiotestaus kannattaa, jos testitapauksia toistetaan useamman kerran ja sillä ehdolla, että testitapaukset ovat automatisoitavissa. Kun harjoitetaan jatkuvaa integrointia, on regressiotestaus lähes välttämätöntä. Jatkuvaa integrointia tehdään yleensä ketterissä projekteissa, koska koodia integroidaan ja testataan sitä mukaan, kun toiminnallisuuksia tulee valmiiksi. (Juvonen, 2018, s. 32–33.)



KUVA 1. Kaksi kehityshaaraa yhdistetään versiohallinnassa: regressiotestauksen avulla voidaan varmistaa, että molemmista versioista tuodut ominaisuudet toimivat yhdessä oikein. (Kasurinen, 2013, s. 69.)

3.2.6 Järjestelmätestaus

Järjestelmätestauksella tarkoitetaan kokonaiselle järjestelmälle tehtävää testausta, jossa tarkastellaan toiminnallista kokonaisuutta. Se on yleisnimike, eikä mikään tietty testaustapa. Järjestelmätestaus voi olla esimerkiksi mustalaatikko-mallia, käyttäjätestausta, kuormitustestausta tai tutkivaa testaamista.

Järjestelmätestauksen aikana testaus suoritetaan testiympäristössä, ei varsinaisessa lopullisessa kohdeympäristössä. Lopulliseen kohdeympäristöön siirtymiseen jälkeen puhutaan hyväksymistestauksesta.

Tärkeimpinä eroavaisuuksina järjestelmä- ja hyväksymistestauksen välillä nähdään, että yksittäisistä komponenteista etsitään vielä virheitä järjestelmätestauksen aikana ja toteutettavaan järjestelmään kohdistuvat muutokset ovat yleisiä. Hyväksymistestauksessa järjestelmää ei muuteta enää merkittävästi. Ja painopiste siirtyy vahvasti toiminnallisuuksien todentamiseen. (Kasurinen, 2013, s. 56–57.)

3.3 Täydellisen testauksen mahdottomuus

Täydellinen testaaminen on sitä, että testataan kaikki mahdolliset erilaiset tapaukset ja vaihtoehdot. Testaustoimintaa pitää hallita. On tiedettävä mitä järjestelmässä testataan, ja minkä takia. (Kasurinen, 2013, s. 19–20.)

Käytännössä on mahdotonta saavuttaa täydellinen testikattavuus, riippumatta siitä, mitä testausmenetelmiä projektiin valitaan. Se olisi liian kallista, eikä tarkoituksenmukaista. Päätös siitä, milloin on testattu tarpeeksi, riippuu muun muassa projektin budjetista, siihen käytettävissä olevasta ajasta, tavoitellusta laatutasosta, yrityskulttuurista, testitiimin näkemyksestä sekä regressiotarpeesta. Testikattavuutta voidaan mitata siihen suunnitelluilla työkaluilla, jotka paljastavat, mitkä osat koodista jäävät nykyisillä testitapauksilla testaamatta. Uusien virheiden löytäminen vaikeutuu sitä mukaan, mitä enemmän testataan. (Juvonen, 2018, s. 32–33.)

ISTQB-testaussertifikaatissa (ISTQB 2013), tiivistyy testauksen periaatteet erinomaisesti: *”Ohjelmistotestaukselle on asetettu seitsemän periaatetta. Näiden periaatteiden mukaan ohjelmistojen testaaminen voidaan määritellä ja perustella käytännössä seuraavasti.”*

Periaate 1	Testaus osoittaa vikojen olemassaolon. Tehtävänä on näyttää, että testatussa ohjelmistossa on vikoja ja pienentää todennäkisyyttä siihen, että ohjelmasta löytyy edelleen vikoja.
Periaate 2	Täydellinen testaus on mahdotonta. Testauksen pitäisi perustua riskien kartoittamiseen sekä tehtävän testaustyön priorisointiin.
Periaate 3	Aikainen testaus. Testaus pitää aloittaa jo esituotantovaiheessa. Testauksen tulee kattaa myös vaatimusmäärittelyt ja projektia varten tehdyt suunnitelmat.
Periaate 4	Vikojen kasaatuminen. Testauksen painopiste tulisi sijoittaa niihin moduuleihin ja osiin, joissa tiedetään tai odotetaan olevan eniten vikoja.
Periaate 5	Hyönteismyrkkyparadoksi. Jos testitaupauksia ei missään vaiheessa uusita tai tarkasteta, pädytään tilanteesee, jossa ainoastaan kyseisten testien huomioimat virheet on poistettu järjestelmästä. Tämän vuoksi testaustapauksia tulee päivittää, lisätä ja kehittää projektin edetessä.
Periaate 6	Testaus on tilanneriippuvaista. Testausta tehdään eri tavoilla erilaisissa tilanteissa tai eri projekteissa. Tämän vuoksi testauksen hallinnointimallien tai prosessien pitää mahdollistaa riittävästi liikkumavaraa, jotta erilaiset tilanteet saadaan selvitettyä ilman, että mallia joudutaan muuttamaan.
Periaate 7	Virheettömyyden harhaluulo. Vikojen löytäminen ja korjaaminen ei auta, jos rakennettu järjestelmä on käyttökelvoton. Käytännössä tämä tarkoittaa sitä, että mikään testaaminen, laadunvalvontatyö tai virheiden korjaaminen ei auta, jos tuote on suunniteltu väärin tai ei toteuta kaikkia niitä odotuksia, jotka tuotteelle on asetettu.

Kuva 2. Testauksen periaatteet ISTQB-testaussertifikaatin mukaisesti.

Testaus on muutakin kuin pelkkää ohjelman käyttämistä ja toiminnallisuuksien toteamista. Testauksessa on paljon erilaisia työvaiheita, jotka vaativat erilaisia lähestymistapoja, testausmenetelmiä sekä tiettyjen asioiden tarkastelemista. Tämän kaiken tavoite on löytää ohjelmasta vikoja lähestymällä sitä eri näkökulmista. (Kasurinen, 2013, s. 48–49.)

3.4 Assertointi ja verifiointi

Assertointi ja verifiointi ovat testausohjelman kautta hallittavissa olevia testausmetodeja. Jos ehto ei täyty, assertointi lopettaa testin suorittamisen. Verifiointi taas jatkaa suorittamista, vaikka ehto ei täyttyisikään. Molemmille löytyvät omat käyttötarkoituksensa. Assertointi sopii tilanteeseen, jossa tarkistetaan, sivulla olevan elementin olemassaolo. Esimerkiksi testin alkuun assertioimalla tarkistetaan, onko sivu oikea. Ja sitten verifioimalla tarkistetaan elementin olemassaolo. (Selenium, 2021b.)

3.5 Testauksen suunnittelu

Ohjelmistoprojektin testauksen suunnittelu etenee seuraavasti: ohjelmistoprojekteissa kerätään toteutettavasta järjestelmästä lista vaatimuksia. Tämän jälkeen päätetään, mitä ohjelman pitää pystyä tekemään ja mitä rajoitteita siinä on. Vaatimuslistan pohjalta laaditaan ensimmäiset testitapaukset,

toisin sanoen määritetään, mitä järjestelmän pitää tehdä, kun sitä käytetään määrätyllä tavalla. Tarvittaessa tehdään komponenttien riskikartoitus, jolla selvitetään resurssitarve ja suunnitellaan testausympäristö, jossa tullaan toimimaan. (Kasurinen, 2013, s. 63.)

Työni tulee olemaan dynaamista testausta. *"Dynaamisella testauksella tarkoitetaan testauksen muotoja, jossa testattavaa järjestelmää varsinaisesti käytetään, ja järjestelmän reaktioita annettuihin syötteisiin seurataan"* (Kasurinen, 2013, s. 65).

4 SELENIUM

Jason Huggins rakensi Seleniumin ydinmoduulin ”JavaScriptTestRunner” vuonna 2004 Chicagossa. Sen tarkoituksena oli testata aika- ja kulut-applikaatiota ja siinä käytettiin Python ja Plone - ohjelmointikieliä. Jason esitteli idean muutamille kollegoille. Yksi heistä, Paul Hammant, näki demon ja otti puheeksi Seleniumin ilmaisjakelun ja mahdollisuuden käyttää Seleniumia millä tahansa ohjelmointikielellä. Samoihin aikoihin Beassa Dan Fabulich ja Nelson Sproul loivat Selenium Remote Controlin ja Japanissa Shinya Kasatani kehitti Selenium IDE:n. Chicagossa Haw-bin Chai toi projektiin mukaan X-Pathin, joka helpotti UI-elementtien löytämistä. Simon Stewart yhdisti Selenium Remote Controlin ja Selenium IDE:n yhdeksi toimivaksi ohjelmaksi. Nykyään käytössä on kolme Selenium-vaihtoehtoa, Selenium WebDriver, Selenium IDE sekä Selenium Grid. (Selenium, 2021a.)

Selenium WebDriver käyttää selainta kuten itse sitä käyttäisit, joko paikallisesti tai etänä. Selenium IDE:llä pystytään nauhoittamaan ja toistamaan omia testejä missä tahansa selaimessa. Ja Selenium Grid mahdollistaa WebDriverin ajamisen usealla koneella samaan aikaan lyhentäen testien ajoaikaa radikaalisesti. Seleniumia käytetään pääsääntöisesti web-applikaatioiden testaamiseen, mutta sillä voidaan tehdä muutakin, kuten esimerkiksi automatisoida web-pohjaisia ylläpidollisia tehtäviä. (Selenium, 2021b.) Selenium on jo pitkään ollut yksi suosituimmista testausohjelmista. Ja Norjan testit olivat tehty Seleniumilla. Näiden vuoksi päädyin tekemään opinnäytetyöni testit myös Seleniumilla, tarkemmin Selenium WebDriverilla.

5 OPINNÄYTETYÖN KÄYTÄNNÖN OSUUS

Testien tekemistä varten luotiin uusi tyhjä tietokanta, johon luotiin testien vaatima tietosisältö, kuten käyttäjätunnukset, joita tarvitaan sisäänkirjautumiseen. Näitä tietosisältöjä käytetään testien ajamiseen. Testattavat ominaisuudet määräytyivät featurelistalta, joka sisälsi 30 opinnäytetyön tilaajan valitsemaa ominaisuutta. Jokainen kirjoittamani testi testasi yhtä listan ominaisuutta. Jokaisella ominaisuudella on järjestelmänvalvojan sivulta löytyvä oma feature-switch, jolla kyseinen ominaisuus saadaan päälle tai pois päältä. Suurin osa testeistä ovat yrityssalaisuuksia, joten niistä en voi työssäni kertoa. Tässä kuitenkin muutama ominaisuus, joiden raportointiin sain luvan opinnäytetyön tilaajalta:

- Y-tunnuksen validointi
- Henkilötunnuksen validointi
- Kuntakoodin validointi
- Kaksikielisyyden tarkistaminen

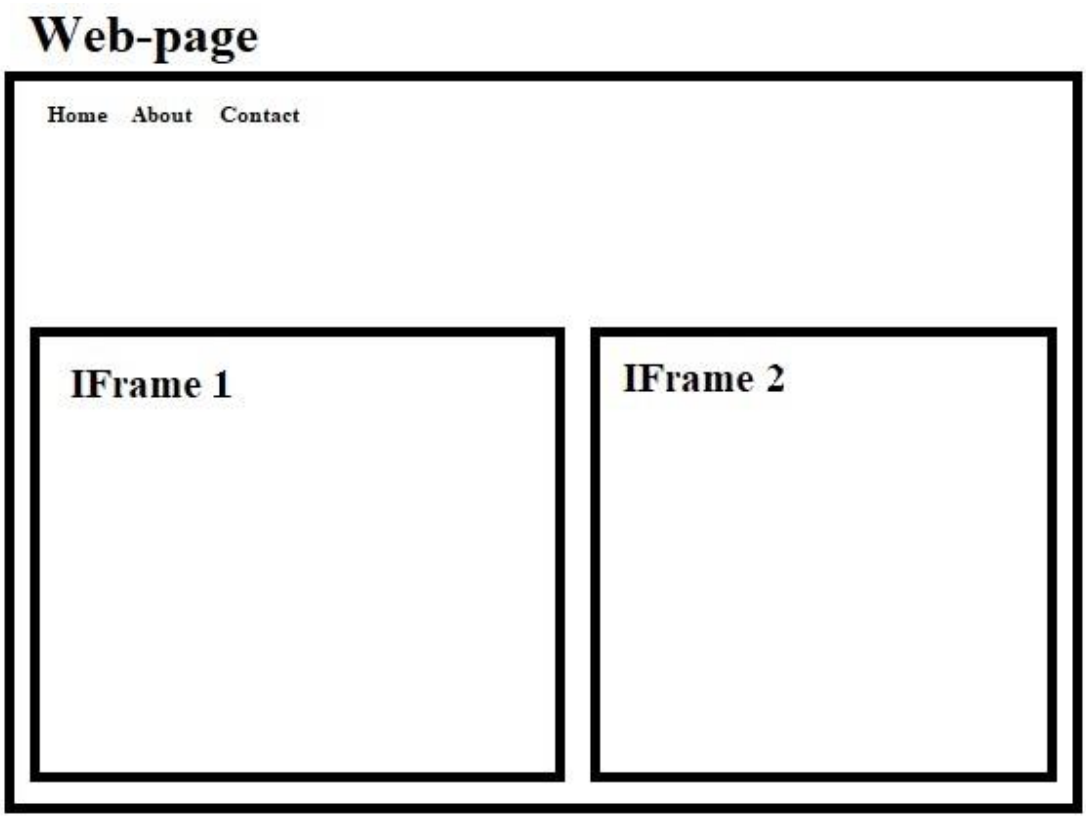
Validoinneissa testi tarkistaa, että syötetty arvo on oikeassa muodossa ja että ohjelma antaa virheviestin, jos väärässä muodossa oleva arvo syötetään. Ohjelma ei anna käyttäjän edetä, ennen kuin arvo on oikeassa muodossa. Validointitestejä on helppo ja nopea luoda lisää luomieni testien pohjalta. Kaksikielisyyden tarkistamisessa testi tarkistaa, että sivu kääntyy nappia painamalla suomesta ruotsiksi ja takaisin, ja että oletuskielenä on suomi.

5.1 Abstraktiluokka TestUtils

Uudelleenkirjoituksen ja luettavuuden takia, luotiin abstraktiluokka nimeltä TestUtils, joka sisältää paljon apumetodeja. TestUtils:ssa otetaan käyttöön testien ajamista varten tarvittava ChromeDriver, joka mahdollistaa sen, että testit voivat käyttää Chromeselainta. ChromeDriverin käyttöönotto tässä vaiheessa on olennaista, koska sen vuoksi jatkossa jokaisessa testissä ei tarvitse määrittää sitä uudelleen, vaan se alustetaan automaattisesti TestUtils-luokan kautta. Tärkein luomistani metodeista on sisäänkirjautuminen. Metodi menee ohjelman sivulle, valikoi testien ajon kannalta oikean käyttäjän ja muut sisäänkirjautumiseen tarvittavat tiedot, sekä kirjautuu ohjelmaan sisään. Tämän jälkeen sisäänkirjautumismetodi tarkistaa, että aloitussivu latautuu näkyville. Vasta sen tapahduttua, metodia kutsunut testi alkaa testata määrättyä ominaisuutta.

Odotusmetodeja luotiin useita. Odotusmetodit nimensä mukaisesti odottavat, että haluttu elementti tulee näkyviin, on käytettävissä, tai ei ole käytettävissä. Jos metodi ei onnistu tehtävässään, eli esimerkiksi elementti ei tule näkyviin, niin testi epäonnistuu ja testin ajajalle annetaan virheviesti.

Assin ohjelmistokehikko koostuu sekä vanhasta, että uudesta Angularista. Koska Assia kehitetään jatkuvasti, vanhan Angularin päivittäminen uuteen tapahtuu osa kerrallaan. Tämä tarkoittaa sitä, että osa sivusta voi olla uudella Angularilla tehty ja toinen osa vanhalla. Tämä osoittautui haasteelliseksi, koska vanhalla Angularilla luodut osat ovat aina IFrame-elementin (ks. kuva 3) sisällä, mikä vuorostaan tarkoittaa sitä, ettei testistä pääse suoraan Iframe-elementin sisällä oleviin tietoihin käsiksi. Kaikki vanhalla Angularilla toteutettu sisältö on tulostettu IFrame:n sisään.



KUVA 3. Esimerkki internetsivusta, jossa on kaksi IFramea vierekkäin ja kummassakin IFramessa on omat sisällöt

Ongelma ratkaistiin luomalla apumetodit, joiden avulla pystytään vaihtamaan tarvittaessa IFrame-elementin sisälle tai ulos siitä (ks. kuva 4). Jälkeenpäin tämä vaikuttaa selkeältä ratkaistulta, mutta asian oivaltaminen ja ratkaisun löytäminen vei minulta kohtuuttoman paljon aikaa. Kuvassa 4 on kyseisten apumetodien koodi.

```
public void SwitchToFrameByAngular(EnumAngular enumAngular, IWebDriver _driver)
{
    if (enumAngular == EnumAngular.NewAngular)
    {
        _driver.SwitchTo().DefaultContent();
    }
    else
    {
        SwitchToDefaultFrame(_driver);
    }
}

1 reference | TeemuLuukkanen, 3 days ago | 1 author, 2 changes
public void SwitchToDefaultFrame(IWebDriver _driver)
{
    _driver.SwitchTo().DefaultContent();
    _driver.SwitchTo().Frame(_driver.FindElement(By.XPath("//iframe[@data-sel-id='app-frame']")));
}
```

KUVA 4. Kaksi metodia IFrame-elementtejä varten

TestUtils-luokkaan luotiin yhteensä 16 metodia. TestUtils-luokan tarkoitus on helpottaa ja nopeuttaa uusien testien kirjoittamista, sillä sen sisältämät metodit toistuvat lähes jokaisessa testissä. Ja luokan metodeja käyttämällä vältytään itsensä toistamiselta testejä tehdessä. Näin koodi on siistimpää ja luettavampaa.

5.2 XPath-sijaintipolku

HTML-elementtien löytämiseksi sivulta, käytettiin XPath-sijaintipolkua (ks. kuva 5). Tyylejä elementtien löytämiseksi on useita, mutta XPathiin päädyttiin, koska se vaikutti luotettavimmalta tavalta ja sitä käytettiin Norjassa tehdyissä testeissä. W3schools (2022) mukaan, *"XPath on lyhenne sanoista XML Path Language. Se on ei-XML-pohjainen kieli XML-dokumenttien osien osoittamiseen ja XML-dokumentin rakenteeseen perustuvan tiedon luontiin. XPath-kieli perustuu XML-dokumentin puumuotoiseen esitystapaan ja antaa siten mahdollisuuden poimia eri osia dokumentista tietyillä valintakriteereillä."*

```
2 references | TeemuLuukkanen, 8 days ago | 1 author, 1 change
protected By Kieli => By.XPath("//div[@class='navbar-desktop__menu-right']/div[@class='navbar-desktop__element navbar-desktop__element--right']");
1 reference | TeemuLuukkanen, 5 days ago | 1 author, 1 change
protected By Seuraava => By.XPath("//button[text()='Seuraava']");
2 references | TeemuLuukkanen, 5 days ago | 1 author, 1 change
protected By Hetu => By.XPath("//input[@name='birthNumber']");
2 references | TeemuLuukkanen, 5 days ago | 1 author, 1 change
protected By Salasana => By.XPath("//input[@name='alternativeLoginPassword']");
1 reference | 0 changes | 0 authors, 0 changes
protected By Asiakas => By.XPath("//fl-dropdown[@name='orgs']/div/button");
1 reference | 0 changes | 0 authors, 0 changes
protected By User => By.XPath("//fl-dropdown[@name='employees']/div/button");
```

KUVA 5. XPathilla pystytään tekemään todella monimutkaisiakin elementtien paikannuspolkuja

Otetaan kuvan 5 koodista Hetu XPath tarkempaan tarkasteluun. Xpath-polussa etsitään input elementtiä, jolla on name-parametrityyppi ja tämän sisältämä arvo on 'birthNumber'. Kun Hetulla on XPath-polku, voidaan testissä apumetodia kutsumalla tarkistaa muun muassa seuraavat asiat. Näkyykö kyseinen elementti sivulla? Onko kyseisen kentän sisältämä arvo oikein? Ja onko kyseinen elementti interaktiivinen?

Suurin työ oli luoda apumetodit. Kun ne oli luotu, testien kirjoittaminen oli helppoa ja nopeaa. Suunnitellusta aikataulusta jäätiin sen verran jälkeen, että päätettiin supistaa testien määrää kolmestakymmenestä kuuteentoista.

6 POHDINTA

Aluksi todettakoon, että vaikka tein paljon taustaselvityksiä etukäteen, rajasin aiheen tarkasti ja varmistin, että minulla on taustatuki käytössä – yllätyksiä ja vastoinkäymisiä tuli ja reilusti. Jälkeenpäin ajatellen, minun ei olisi pitänyt tehdä opinnäytetyön käytännön osuutta kesäaikaan, sillä vaikka itselläni oli silloin hyvin aikaa työn tekemiseen, taustatukeni olivat lomailemassa ja sain taistella vastoinkäymisten kanssa yksinäni.

Minulle annettiin kolme kuukautta aikaa tehdä työ. Ensimmäisen kuukauden opiskelin Seleniumia ja selvitin tietotaustaa. Yritin myös muokata Flytin koodia testeihini sopivaksi. Tämä kuitenkin osoittautui virheeksi. Flytin koodi oli erittäin monimutkaista ja monimuotoista sekä osin vanhentunutta. Puolessa välissä projektin aikajanaa päätimme yhteispalaverissa, että parasta olisi unohtaa tämä lähestymistapa ja aloittaa alusta luomalla itse koodi kokonaisuudessaan.

Palaverissa selvisi myös, että taustatukiohjelmoijat eivät olleet ehtineet perehtyä Seleniumiin. Projekti jatkui niin, että toinen heistä otti asiakseen perehtyä Seleniumiin paremmin, jotta pystyisi auttamaan minua. Tavoitin onnekseni pitkäaikaisten yritysten jälkeen Norjasta yhden Seleniumtestien kehittäjistä, joka oli palannut lomiltaan. Sain häneltä ratkaisevan neuvon työssäni etenemiseen. Olin juuri päässyt vauhtiin, kun käyttäjätunnukseni hävisivät tietokoneen uudelleenkäynnistämisen yhteydessä ilman mitään näkyvää syytä. Seuraava viikko vierähtikin IT-tuen kanssa. Ongelma paikallistettiin domain-tasolle, eikä sitä valitettavasti saatu korjattua. Toisin sanoen aloitin työni kolmannen kerran lähes alusta. Siinä vaiheessa aikaa oli jäljellä kaksi viikkoa. Pidimme yhteispalaverin, jossa tarkensimme odotuksia ja tavoitteita. Yritin pyytää lisäaikaa projektin loppuunsaattamiseksi, mutta se ei valitettavasti onnistunut. Onneksi loppuaika sujui ongelmitta. Ja sain luotua tarvittavat testit, joita oli lopulta 16.

En voi sanoa olevani tyytyväinen opinnäytetyöhöni. Jos saisin aloittaa alusta, tekisin monen asian eri tavalla. En yrittäisi muokata Flytin koodeja vaan tekisin oman projektin. Ajoittaisin työn tekemisen johonkin muuhun ajankohtaan kuin kesälomakauteen. Tekisin myös ensin käytännön työn ja vasta jälkeenpäin kiireettä teoriaosuuden. Koen, että jos olisin saanut lisäaikaa loppuvaiheessa, olisin päässyt alkuperäisiin tavoitteisiin. Nyt tavoitteista kuitenkin jouduttiin tinkimään ja työssä jäi toteuttamatta osa alun perin halutuista toiminnoista. Opin kuitenkin todella paljon. Niin Selenium-testauksesta kuin projektinhallinnastakin. Koodaustaitoni paranivat huomattavasti ja opin hahmottamaan paremmin kokonaisuutta sekä löytämään virheitä koodissa. Opin näkemään testauksen tuomat hyödyt ja ymmärtämään täydellisen testauksen mahdottomuuden sekä rajaamaan testauksen olennaisiin sekä kriittisiin ominaisuuksiin.

Työtäni pystytään jatkojalostamaan tiimissä, sillä varsinkin tekemäni TestUtils-luokka sisältää suuren määrän välttämättömiä apumetodeja ja toimii erinomaisena pohjana uusille testeille. Jatkokehityksenä tietokanta populoitaisiin koodin kautta. Näin varmistettaisiin, että jokaisella testiajolla tietokannasta löytyvät aina saman tiedot. Tämän lisäksi tästä työstä pois jääneet testit toteutettaisiin. Testikattavuutta voitaisiin laajentaa koskemaan jokaista ohjelman ominaisuutta.

LÄHDELUETTELO

Dustin, E.;Thom, G.;& Nernie, G. (2009). *Implementing automated software testing.* Boston: Pearson Education Inc.

Haikala, I.;& Mikkonen, T. (2011). *Ohjelmistotuotannon käytännöt.* Hämeenlinna: Talentum media oy.

Haikala, I.;& Märijärvi, J. (2004). *Ohjelmistotuotanto.* Helsinki: Talentum.

Hertzel, B. (1993). *The complete guide to software testing.* A Wiley-QED Publication.

Iyengar, P. (28. 12 2021). *User Interface Testing: A Complete Guide*. (Headspin inc) Haettu 29. 10 2022 osoitteesta Headspin: https://www.headspin.io/blog/ui-testing-a-complete-guide-with-checklists-and-examples

Juvonen, R. (2018). *Ohjelmistoprojektin sudenkuopat ja miten ne vältetään.* Helsinki: Books on Demand.

Kaner, C.;Bach, J.;& Pettichord, B. (2002). *Lessons Learned in Software Testing.* New York: John Wiley & Sons Inc.

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

Melrose, M. (2001). Maximizing the rigor of action research: why would you want to? How could you? *Field Methods, 13(2)*, 160-180. doi:https://doi.org/10.1177/1525822X0101300203

Metsämuuronen, J. (2003). *Tutkimuksen tekemisen perusteet ihmistieteissä.* Jyväskylä: Gummerus Kirjapaino Oy.

Myers, G. (1979). *The art of software testing.* New York: A wiley-interscience publication.

Sandler, C.;Badgett, T.;& Myers, G. (2011). *The Art of Software Testing.* New Jersey: John Wiley & Sons Inc.

Selenium. (2021a). *The story starts in 2004*. (Software Freedom Conservancy) Haettu 29. 10 2022 osoitteesta Selenium: https://www.selenium.dev/history/

Selenium. (2021b). *Overview of Test Automation*. (Selenium) Haettu 29. 10 2022 osoitteesta Selenium: https://www.selenium.dev/documentation/test_practices/overview/#to-automate-or-not-to-automate

W3schools. (2022). *What is XPath*. (Refsnes Data) Haettu 29. 10 2022 osoitteesta W3schools: https://www.w3schools.com/xml/xpath_intro.asp