KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutus

Anniina Aaltonen

VERKKOKAUPAN REGRESSIOTESTAUKSEN AUTOMATISOINTI

Opinnäytetyö
Syyskuu 2019

Karelia AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Syyskuu 2019
Tietojenkäsittelyn koulutusohjelma

Tikkarinne 9
80200 JOENSUU
+358 13 260 600 (vaihde)

Tekijä
Anniina Aaltonen

Nimeke
Verkkokaupan regressiotestauksen automatisointi

Toimeksiantaja
Stockmann Oyj Abp

Tiivistelmä

Tässä opinnäytetyössä toteutettiin automaatiotestejä verkkokaupan manuaalisen regressiotestaamisen rinnalle. Toimeksiantajana toimi Stockmann Oyj Abp, joka on vähittäiskauppaa harjoittava suomalainen pörssiyhtiö. Tavoitteena oli perehtyä testausautomaatiotyökalu Selenium IDE:en sekä suunnitella ja luoda automaatiotestit kyseisellä työkalulla.

Opinnäytetyössä käytiin läpi yleisesti ohjelmistotestausta, testauksen eri tasoja ja vaiheita sekä perehdyttiin ketterään ohjelmistokehitykseen. Lisäksi teoriaosuudessa käsiteltiin testausautomaatiota ja toiminnalliseen osuuteen valittua testaustyökalua.

Työn toiminnallisen osuuden automaatiotestit saatiin tavoitteiden mukaisesti toimiviksi ja valmiiksi. Tuloksena syntyi yhdeksän verkkokauppatilauksen testauspaketti, jolla pystytään testaamaan yhdeksän eri verkkopankkia. Testaustulosten perusteella voidaan päätellä, että automaatiotestejä ajettaessa saadaan keskimääräisesti aina nopeampia tuloksia kuin manuaalisesti tehdyissä testeissä. Tulevaisuudessa toimeksiantajan on helpompi lähteä laajentamaan testausautomaatiota muihin verkkokaupan regressiotestauksen osiin, kun ensimmäinen osio on valmis.

Kieli
suomi

Sivuja 34

Asiasanat
ohjelmistotestaus, automaatiotestaus, regressiotestaus, ketterä kehitys

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
September 2019
Business Information Technology

Tikkarinne 9
80200 JOENSUU
FINLAND
+ 358 13 260 600 (switchboard)

Author
Anniina Aaltonen

Title
Automated Regression Testing for an Online Store

Commissioned by
Stockmann Oyj Abp

Abstract

The purpose of this thesis was to implement automated regression testing for an online store. This thesis was commissioned by Stockmann Oyj Abp, which is a Finnish retailer public company. The goal was to orientate oneself with the testing automation tool Selenium IDE and design and create automation testings with that testing tool.

This thesis provides general information about software testing, different levels and phases of testing and agile development. The theoretical part also includes facts about automation testing and selection of the testing tool for the action-oriented part.

Automation tests for the action-oriented part were succesfully completed. As a result, there is a testing package of nine online store orders, which can test nine different online banks. Based on the results, it can be concluded that running automation tests produces, on average, faster results than manual tests. In the future, it will be easier for the commissioner to continue extending test automation to other regression testing parts when the first section is complete.

Language

Finnish

Pages 34

Keywords

software testing, automation testing, regression testing, agile development

Sisältö

1 Johdanto

Ohjelmistotestaus on tarkkaa työtä ja tärkeä osa ohjelmistokehityksessä. Tärkeää on myös, että testausta ei suoriteta vain yhdellä tasolla. Sitä tulisi tehdä useilla eri tasoilla, jotta voidaan varmistua siitä, että kehitettävä verkkosivu tai sovellus on laadukas. Testauksen automatisointi on myös yksi hyvä tapa kitkeä inhimillisiä virheitä sekä sillä voi säästää työaikaa manuaaliselta testaamiselta.

Tämän opinnäytetyön tavoitteena on automatisoida osa toimeksiantajan verkkokaupan regressiotestauksesta ja automaation avulla vähentää testaajien manuaalista työtä. Toimeksiantajana tässä opinnäytetyössä toimii Stockmann Oyj Abp, joka oli myös kirjoittajan työnantajana opinnäytetyön tekohetkellä.

Stockmann on vähittäiskauppaa harjoittava suomalainen pörssiyhtiö, joka on perustettu vuonna 1862. Tällä hetkellä Suomessa on kuusi tavarataloa sekä Baltiassa kaksi tavarataloa. Lisäksi yhtenä myyntikanavana toimii verkkokauppa Stockmann.com. Stockmannin verkkokauppa on alkujaan avattu vuonna 2010. Vuonna 2018 Stockmann käynnisti digitaalisen kiihdytyshankkeen, jossa panostetaan digiosaamisen vahvistamiseen. Tavoitteena on verkkokaupan kasvattaminen ja monikanavaisuuden vahvistaminen.

Verkkokaupan testaajat ovat tehneet tähän saakka kaikki verkkokaupan regressiotestaukset manuaalisesti alusta loppuun asti itse. Tavoitteena on tehdä valmis testauspaketti automaatiotesteihin tarkoitetulla testaustyökalu Selenium IDE:llä, josta hyötyisi regressiotestauksia tekevä verkkokauppatiimi.

Opinnäytetyön teoriaosuudessa käsitellään ohjelmistotestausta ja mitä se yleisesti on. Lisäksi työssä kerrotaan hieman testausautomaatiosta ja yleisellä tasolla ohjelmointivirheistä. Ohjelmistotestauksen lisäksi opinnäytetyössä käsitellään yleisesti ohjelmistokehityksen ketteriä menetelmiä, sekä Scrum- ja Kanban menetelmiä, joita toimeksiantaja käyttää.

Valitsin opinnäytetyön aiheeksi verkkokauppatestauksen automatisoinnin, sillä olen saanut työelämässä kokemusta ainoastaan manuaalisesta testaamisesta. Erityisesti automatisointi kiinnostaa ja haluan päästä kokeilemaan, pystyisikö omia työtehtäviä automatisoimaan ja samalla auttamaan myös muita testaajia.

2 Ohjelmistotestaus

2.1 Yleisesti testauksesta

Ohjelmistotestaus on työtä, jonka avulla varmistetaan, että tekeillä olevasta ohjelmistotuotteesta tulee toivotun kaltainen. Kaikki tuotteeseen valmiiksi saadut ominaisuudet varmistetaan, että ne toimivat niin kuin on tarkoituskin. Se on yksi kokonaisuus, joka kuuluu ohjelmistotuotannon piiriin ja Jussi Pekka Kasurinen määrittelee sen näin: ”varmistetaan että tehdään oikeaa tuotetta ja että tuote on tehty oikein.” (Kasurinen 2013, luku 1.)

Testauksella halutaan tarkastaa, että se mitä on saatu valmiiksi, vastaa myös sitä, mitä on ollut tarkoitus tehdä. Lisäksi sillä halutaan tunnistaa ne tuotoksen kohdat, jotka ovat poikenneet suunnitelmista. (Kasurinen 2013, luku 1.)

Testaaja tekee työssään monia erilaisia asioita ja työvaiheesta riippuen testaaja voi päätyä tekemään dokumentaatioita, kirjoittamaan koodia tai pitämään koekäyttäjien haastatteluja. Tästä johtuen testaus on kokonaisuutena paljon laajempi asia kuin esimerkiksi ohjelmointityö tai tekninen kirjoittaminen. (Kasurinen 2013, luku 1.)

Haikalan ja Märijärven (2004, 284) mukaan testaus on normaalissa puhekielessä melkein mitä tahansa kokeilemista. Erilaisia ohjelmistoja testatessa perinteinen määrittely testaukselle on suunnitelma etsiä virheitä suorittamalla ohjelmistoa tai sen osaa. Ohjelmistojen testauksen määritelmässä keskeiset asiat ovat suunnitelmallisuus ja etsintä. Umpimähkäinen testaaminen kokeilemalla ja usein vielä

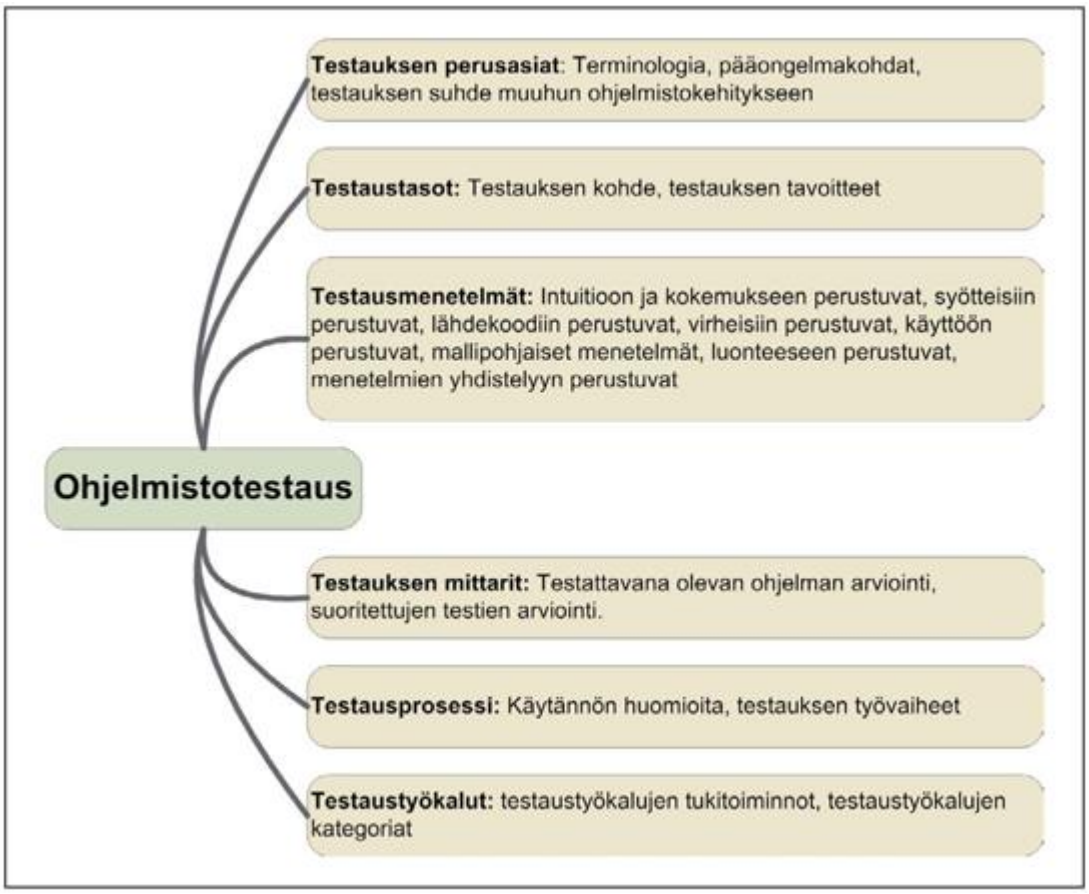
ohjelmiston tekijän toimesta on monesti se, mitä tehdään. Tällöin ohjelman toimivuus ja sen todentaminen on etusijalla, eikä itse virheiden löytäminen. (Haikala & Märijärvi 2004, 284.)

Testauksessa työtuntien ja testitapausten määrä ei välttämättä tarkoita, että testaus olisi tehokasta. Parempia tuloksia voi saada pienellä, hyvin suunnitelluilla testitapauksilla ja muutaman tunnin testauksella kuin esimerkiksi useiden päivien umpimähkäisellä kokeilulla. (Haikala & Märijärvi 2004, 284.) Nykyään testaus määritellään monesti vähän laajemmin laadun mittaamisen näkökulmasta niin, että testauksen katsotaan käsittävän kaikki tavat, joilla tähdätään mittaamaan ja parantamaan ohjelman laatua (Craig & Jaskiel 2002, 25).

2.2 Testaamisen osa-alueet

Tässä luvussa esitellään perusajatuksia siitä, mitä testaajan pitää osata, miten testaus tapahtuu ja kuinka se jakautuu erilaisille tasoille (Kasurinen 2013, luku 3). Testaustoiminta voidaan jakaa useaan eri osa-alueeseen (Kasurinen 2013, luku 3). Yksi malli on IEEE SWEBOK (Software Engineering Body of Knowledge), josta löytyy teknisiä raportteja, joita käytetään ohjelmistoalan ammattilaisten sertifioinneissa (Kasurinen 2013, luku 3; Suomen Standardisoimisliitto SFS ry 2019). SWEBOK-mallissa osa-alueet jaetaan kuuteen eri pääkategoriaan ja 19 alakategoriaan. Nämä kategoriat kattavat kaikki toiminnot, joita testaajan tulisi ymmärtää ja jotka ovat ohjelmistotestauksessa keskeisiä asioita tekemiselle. (Kasurinen 2013, luku 3.)

Nämä kuusi pääkategoriaa ovat seuraavat: ohjelmistotestauksen peruskäsitteet, testauksen tasot, testausmenetelmät, testauksen mittarit, testiprosessi ja testaustyökalut (kuva 1). Ohjelmistotestauksen peruskäsitteisiin kuuluu testaukseen liittyvää terminologiaa, testauksen pääongelmakohdat sekä testauksen suhde muuhun ohjelmistokehitykseen (Kasurinen 2013, luku 3).



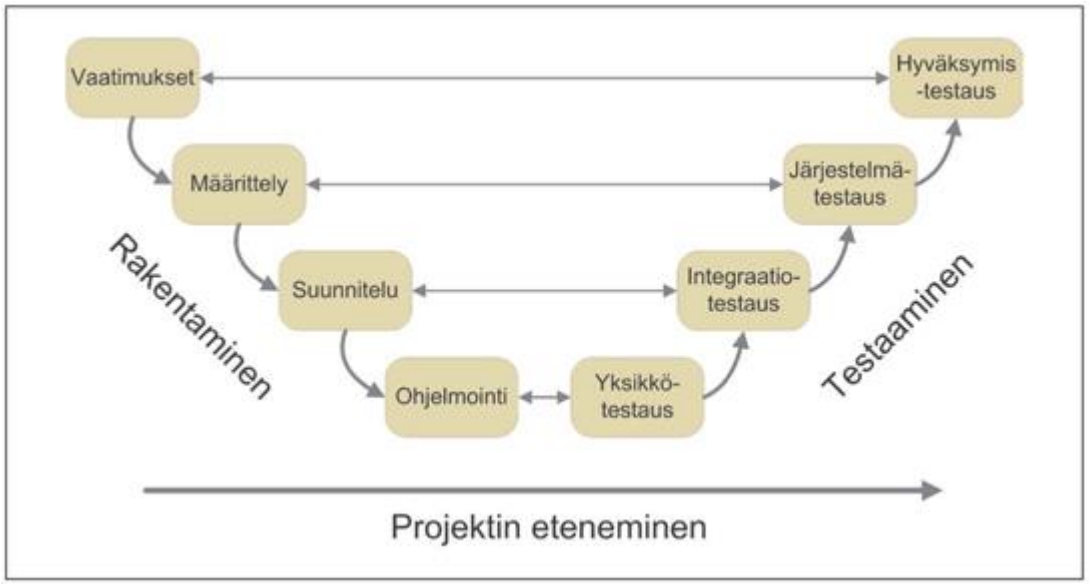
Kuva 1. SWEBOK-mallin kuusi pääkategoriaa. (Kasurinen 2013).

2.3 Testaamisen tasot

SWEBOK-mallin testaustasojen keskeiset osaamisalueet ovat testauksen kohde sekä testauksen tavoitteet. Testauksen kohteet tarkoittavat testattavia osakokonaisuuksia ohjelmista, jolloin testaus voi kohdistua muun muassa yksittäiseen moduuliin, moduulien yhdistämistä kokeilevaan integraatiotestaukseen tai koko järjestelmän kattavaan järjestelmätestaukseen. Testauksen tavoitteet käsittelevät sitä, millaisia erilaisia testaustapoja on olemassa. (Kasurinen 2013, luku 3.)

Testaustasoja kuvaamaan kehitettiin 1990-luvulla myös V-malli, jonka avulla voi visualisoida ohjelmistotuotannon vaiheiden ja eritasoisen testauksen suhdetta toisiinsa (Smart Education 2019a). Testauksessa tämä malli määrittelee tekniselle testaamiselle kolme päävaihetta: moduulitestaus (sisältää myös yksikkötestauksen), integraatiotestaus sekä järjestelmätestaus. V-mallissa testauksen suunnittelu tapahtuu, siis testaustasoa vastaavalla suunnittelutasolla (kuva 2).

Testausvaiheet johtavat lopulta hyväksymistestaukseen. (Kasurinen 2013, luku 3; Haikala & Märijärvi 2004, 288.)



Kuva 2. V-malli. (Kasurinen 2013).

2.3.1 Yksikkötestaus

Yksi ohjelmistotestauksen pääelementeistä on yksikkötestaus, jonka ideana on testata ohjelmiston yksittäistä moduulia, ohjelmaa, oliota, funktiota, komponenttia tai jotain muuta ohjelmiston osaa (Lui & Chan 2008, 271; Kasurinen 2013, luku 3; International Software Testing Qualifications Board 2010, 23). Tarkoituksena on etsiä vikoja ja validoida, että jokainen ohjelmiston yksikkö toimii suunnitellusti (Kasurinen 2013, luku 3; International Software Testing Qualifications Board 2010, 23).

Yksikkötesti on ensimmäinen alimman tason testausvaihe, jossa tarkastellaan tyypillisesti suoraan testattavaa koodia ja se voidaan suorittaa erillään muusta järjestelmästä. Usein testaajana toimii koodin kirjoittanut ohjelmoija ja jos jokin testeistä epäonnistuu, niin testaaja pystyy korjaamaan löydetyt viat nopeasti, ilman että havaintoja hallitaan muodollisesti. (Kasurinen 2013, luku 3; International Software Testing Qualifications Board 2010, 23; Loveland, Miller, Prewitt & Shannon 2005, 29.)

Usein ketteriä menetelmiä käyttäessä yksikkötestaukset ovat tärkeässä roolissa, sillä koontiversioita julkaistaan tiheästi. Onkin tavallista, että ketteriä menetelmiä käyttäessä hyödynnetään testivetoista kehitysmallia, jolloin yksikön ohjelmointi aloitetaan kirjoittamalla sille testit ja koodaamalla itse yksikkö vasta tämän jälkeen. (Smart Education 2019b.)

Stockmannin verkkokauppakehityksessä käytetään hyvin joustavasti ketteriä menetelmiä, yhdistellen Scrum-menetelmää sekä Kanban-menetelmää. Näistä menetelmistä kerrotaan tarkemmin luvussa Ketterät menetelmät. Yksikkötestaus kuuluu pääosin ulkoistetulle kehittäjätiimille, jotka ovat kirjoittaneet ja koodanneet testit heidän testiympäristöönsä. Stockmannin omat testaajat tekevät toisinaan yksikkötestejä testiympäristössä, mutta pääosin yksikkötestaus kuuluu ulkoistetulle kehittäjätiimille.

2.3.2 Integraatiotestaus

Yksikkötestauksen jälkeen on seuraavana työvaiheena integraatiotestaus, jossa järjestelmän eri osia aloitetaan sovittamaan yhteen ja tarkoituksena on saada lopulta koko järjestelmä toimimaan yhtenä kokonaisuutena (Kasurinen 2013, luku 3).

Testauksessa pyritään löytämään ne virheet, joita ei tullut esiin yksikkötestauksessa. Testeissä testataan useiden eri komponenttien yhteistoimintaa, suoritetaan tiettyjä suorituspolkuja, jotka hyödyntävät useita eri yksiköitä tai laajempia komponentteja ja lopuksi arvioidaan testien tuloksia. (Smart Education 2019b.) Näiden testien avulla varmistetaan, että yksiköt ja komponentit toimivat keskenään oikein ja siirtävät rajapintojensa avulla oikeaa tietoa oikeaan paikkaan (Sommerville 2007, 541).

Integraatiotestauksessa suoritetaan testitapauksia, jotka ovat yksikkötestaukseen verrattuna laajempia, mutta eivät kuitenkaan kata vielä täysin koko järjestelmän testausta. Testitapauksina voi olla esimerkiksi eri moduulien välinen viestinvaihto tai samaa tietokantaa käyttävien moduulien yhteistoiminnan

varmistaminen. Lähtökohtana on todistaa, kun toimivaan kokonaisuuteen lisätään aina uusi kokonainen osa lisää, niin tarkastuksien jälkeen se toimii edelleen moitteettomasti. (Kasurinen 2013, luku 3.)

2.3.3 Järjestelmätestaus

Järjestelmätestauksessa on testattavana koko ohjelmisto tai järjestelmä, ja sen ideana on testata kahden tai useamman komponentin toiminnallisuutta. Päämääränä on varmistaa, että komponentit ovat yhteensopivia keskenään ja että ohjelmisto vastaa asetettuja vaatimuksia. Jotta tähän tulokseen päädytään, täytyy testaajaksi pyrkiä valitsemaan sellainen henkilö, joka ei ole ollut mukana kehitystyössä aikaisemmin. (Haikala & Märijärvi 2004, 290; Sommerville 2011, 219.)

Järjestelmätestaus suoritetaan testiympäristössä, eikä varsinaisessa lopullisessa tuotantoympäristössä. Siinä vaiheessa, kun järjestelmä siirretään tuotantoympäristöön, aletaan puhumaan hyväksymistestauksesta. Suurin ero järjestelmätestauksen ja hyväksymistestauksen välillä on, että järjestelmätestauksessa etsitään virheitä edelleen myös yksittäisistä komponenteista, kun taas hyväksymistestauksessa painopiste siirtyy vahvasti toiminnallisuuksien todentamiseen. (Kasurinen 2013, luku 3.)

Järjestelmätestaukseen voi sisältyä tarvittaessa myös erillinen kenttätestaus ja/tai hyväksymistestaus. Näiden lisäksi järjestelmätestauksessa voidaan myös testata järjestelmän ei-toiminnallisia ominaisuuksia, muun muassa miten hyvin järjestelmä selviää oletetusta tai sitä suuremmasta kuormasta (kuormitustestit), kuinka hyvin järjestelmän suunnitelleet käyttäjät saavat hyödynnettyä toteutusta toiminnassaan (käytettävyystestit), kuinka hyvin järjestelmä toipuu virhetilanteista tai kuinka pitkään järjestelmä toimii itsenäisesti ilman ongelmia (luotettavuustestit), sekä onnistuuko järjestelmän asentaminen odotusten mukaisesti (asennustestit). (Haikala & Mikkonen 2011, 208.)

2.3.4 Hyväksymistestaus

Hyväksymistestaus on aiemmin esitellyn V-mallin viimeinen testauksen taso, mutta muiden yleisten testausmallien mukaan voidaan vielä hyväksymistestauksenkin jälkeen suorittaa esimerkiksi laajoja järjestelmäintegrointitestejä. Testauksen tavoitteena on osoittaa, että ohjelmisto tai järjestelmä on riittävän luotettava, korkealaatuinen ja kyvykäs täyttämään annetut vaatimukset. Lisäksi testauksessa pyritään arvioimaan järjestelmän valmiutta käyttöönottoon ja käyttöön. (International Software Testing Qualifications Board 2010, 26; Kasurinen 2013, luku 3.)

Testaajana toimii usein asiakas, joka tarkastaa ja hyväksyy tuotteen liiketoiminnan edustajien näkökulmasta. Kun hyväksymistestaukset on suoritettu onnistuneesti, siirtyy ohjelmisto asiakkaan omaisuudeksi tai ainakin sen kehityksenaikainen huolto- ja korjausvelvoite lakkaa olemasta voimassa. (International Software Testing Qualifications Board 2010, 26; Kasurinen 2013, luku 3.)

2.4 Regressiotestaus

Regressiotestaus ei ole erillinen testauksen muoto, vaan se on yleistermi uudelleentestaamiselle. Tärkein ominaisuus siinä on todentaa, että tehtyjen korjauksien jälkeen järjestelmä toimii edelleen oikein ja mitään uutta ei ole mennyt rikki. Se ei ole siis mikään yksittäinen testauksen työtaso, kuten esimerkiksi yksikkötestaus tai integraatiotestaus ovat, vaan se on yleinen nimitys kaikelle testaustyölle, jolla varmistetaan, että uusin versio toimii oikein. (Kasurinen 2013, luku 4.) Sitä voidaan suorittaa kaikilla testausten tasoilla ja testejä ajetaan useita kertoja ja ne yleensä kehittyvät aika hitaasti (International Software Testing Qualifications Board 2010, 28).

Aina kun uusi moduuli liitetään osaksi integraatiotestausta, ohjelmisto muuttuu erilaiseksi. Tällaiset muutokset voivat aiheuttaa ongelmia toiminnallisuuksiin, jotka toimivat aikaisemmin virheettömästi. (Pressman 2010, 462.) Regressiotestauksen ideana on, että suoritetaan samoja testejä uudelleen, jolloin pyritään var-

mistamaan, että tehdyt muutokset ei aiheuta ongelmia aiemmin toteutettujen elementtien toiminnassa. Testausta tehdään joko manuaalisesti, automatisoidusti, tai molempien yhdistelmänä. (Kasurinen 2013, luku 4; Smart Education 2019b.)

Regressiotestaustavat voidaan jakaa kolmeen erilaiseen ryhmään: testit, jotka hyödyntävät ohjelman kaikkia jo toteutettuja ominaisuuksia, testit sellaisille toiminnoille, joihin muutokset vaikuttavat todennäköisimmin, sekä testit, jotka koskevat vain muutoksien alla olevia komponentteja. Näiden kolmen ryhmän avulla saadaan parannettua testien luotettavuutta ja tehokkuutta sekä hyvät valmiudet ja luottamus kehityksen jatkamiseen. (Pressman 2010, 462.)

Usein isoille komponenteille tai ohjelmistoille on suunniteltu todella laajat regressiotestaukset. Jokin pieni muutos ohjelmiston yhteen osaan saattaa saada aikaan virhetilanteen ihan eri moduulissa, kuin mihin muutos tehtiin. Usein kehittäjät eivät tätä halua uskoa, vaikka todisteita olisi tarjolla. Näiden ongelmien löytämiseksi apuna käytetäänkin juuri regressiotestausta. (Ammann & Offutt 2008, 215.)

Samojen ominaisuuksien testaaminen uudelleen ja uudelleen on usein tylsää ja pitkäveteistä testaajalle. Tämän vuoksi testausten automatisointi kokonaan tai jonkun tietyn osion osalta voisi olla ratkaisu, joka helpottaa testaajien työtä. (Lui & Chan 2008, 23.)

Tämän opinnäytetyön toimeksiantajana toimivan Stockmannin verkkokauppatiimin työnkuvaan kuuluu regressiotestausten suorittaminen aina, kun uusia versioita verkkokaupasta tuodaan QA-ympäristöön. QA-ympäristö on testiympäristö, joka vastaa tuotantoympäristöä ja se tulee sanoista Quality Assurance eli suomennettuna laadunvarmistus. Käytän tässä opinnäytetyössä termiä QA-ympäristö, sillä sitä käyttää myös toimeksiantaja sekä automaatiotestit tehdään siihen ympäristöön. Verkkokauppatiimi testaa verkkokauppaa ja sen toimivuutta pääasiassa QA-ympäristössä sekä tuotantoympäristössä.

Tapana on ollut, että noin 1-5 henkilöä suorittaa regressiotestaukset ja varmistaa, että verkkokauppa toimii perusominaisuuksiltaan oikein, eikä mitään ole mennyt rikki. Testattaviin ominaisuuksiin kuuluvat muun muassa nämä:

- Verkkokauppatilausten tekeminen, varmistetaan että tilaukset menevät läpi ja *Kiitos tilauksestasi* -sähköposti tulee perille.
- Tilauksia tehdessä varmistetaan, että eri maksutavat ja toimitustavat toimivat.
- Kategoriat ja tuotteet näkyvät ja toimivat kaupassa.
- Sisäänkirjautuminen ja uloskirjautuminen, sekä rekisteröinti verkkokauppaan onnistuu.
- Verkkokaupan haku toimii ja tuottaa oikeita tuloksia.

Näiden lisäksi Stockmannin regressiotestauksiin kuuluu myös paljon muita toiminnallisuuksia ja kokonaisuuksia. Mikäli testaajat havaitsevat mahdollisia virheitä, he kirjoittavat tarkan selostuksen bugista, kuinka se saadaan toistettua ja laittavat mahdollisuuksien mukaan kuvia virhetilanteesta. Tämän jälkeen bugi raportoidaan kehittäjätiimille, jolloin he pystyvät aloittamaan mahdolliset korjaustoimenpiteet.

2.5 Testausautomaatio

Testausautomaatiolla tarkoitetaan testaustoiminnan yhtä muotoa, jossa ohjelman testaamista varten rakennetaan automaatiotyökaluja testien suorittamista varten. Tavoitteena on muodostaa joukko toistuvasti tehtäviä testitapauksia ja rakentaa erillinen laite niiden nopeaa tarkastamista varten. Automaation rakentamisessa pohjimmainen ajatus on vapauttaa testaajia muihin työtehtäviin. (Kasurinen 2013, luku 4.)

Testausautomaatio herättää monenlaisia ajatuksia niin kehittäjien kuin testaajienkin parissa. Testaajalle automaatio voi tarkoittaa käyttöliittymätestien nauhoittamista ja sen toistamista apuohjelman avulla, kun taas kehittäjä voi mieltää sen yksikkötestien ajamisena. Kun puhutaan testausautomaatiosta, niin sen voi liittää esimerkiksi osaksi rasitus-, luotettavuus-, turvallisuus- tai suorituskykytestausta. (Dustin & Garrett & Gauf 2009, Luku 1.) Näiden lisäksi testausautomaatioon voidaan liittää hyvin vahvasti regressiotestaus, sillä se on tyypillisin kohde automaatiotestaukselle. Regressiotestauksessa varmistetaan järjestelmän eheys ja testataan ettei uudet toiminnallisuudet ole aiheuttaneet häiriöitä aiemmin tehtyjen osien toimivuuteen. Automatisoimalla regressiotestauksen osia, voidaan vanhoja ominaisuuksia testata minuuteissa. Manuaalista testaamista automatisointi ei

kuitenkaan poista koskaan kokonaan, mutta kehitystyön edetessä luodut automatisoidut testit, voidaan aina jatkossa suorittaa automaatiotesteillä manuaalisen testaamisen sijaan. (ATR Soft 2018.)

Stockmannin verkkokauppakehitys on jatkuvaa, jolloin myös manuaalista regressiotestaamista on vähintään kerran kuukaudessa. Automatisoimalla osia regressiotestauksesta voi toimeksiantaja tulevaisuudessa mahdollisesti säästää kuluissa sekä parantaa testausprosesseja.

Sellaisessa tilanteessa, jossa tuotteesta tehdään esimerkiksi joka yö uusi käännös (daily build), ei testaajien ajankäytön kannalta ole järkevää käyttää aikaa joka päivä samojen perustestien tekemiseen. Toisaalta automaatiotestit voivat olla myös tarkastuksia, jossa esimerkiksi tietokone jätetään yön ajaksi tekemään testejä ja aamulla kehittäjät voivat aloittaa työpäivän käymällä läpi tarkastuksien tuloksia ja korjata mahdollisesti löytyneet ongelmat. (Kasurinen 2013, luku 4.)

2.6 Ohjelmointivirheet (bugit)

Testauksen yhtenä ideana on jäljittää ja korjata virheet. Sen avulla on mahdollista osoittaa, että ohjelmassa on virheitä. (Haikala & Mikkonen 2011, 205.) Ohjelmointivirhe tai toiselta nimeltään bugi on siis ohjelmiston lähdekoodissa oleva virhe (Tech Target 2007). Virheiden vakavuudet voivat vaihdella, sillä virhe voi olla esimerkiksi käyttäjää ärsyttävä kosmeettinen yksityiskohta tai sitten sellainen, joka estää jonkin järjestelmän käyttämiseen kokonaan (Haikala & Mikkonen 2011, 206).

On arvioitu, että virheitä on ohjelmistoissa yleensä noin yksi muutamaa kymmentä ohjelmariviä kohden. Käytössä olleissa vanhemmissa ohjelmistoissa on arvioitu olevan noin yksi virhe tuhatta ohjelmariviä kohden. Lisäksi on myös tehty arvioita, joiden mukaan noin viisi prosenttia ohjelmavirheistä on sellaisia, joita ei koskaan edes löydetä, sillä vaikka ohjelmistoa käytetään, ei se välttämättä johda virhetilanteisiin. (Haikala & Mikkonen 2011, 206.)

3 Toimeksiantajan ketterät menetelmät

3.1 Scrum

Ketteriä menetelmiä kutsuttiin alkujaan kevyiksi menetelmiksi, sillä niitä oltiin kehitelty 1990-luvulla perinteisen ohjelmistokehitysten rinnalle. Tekniikan kehittyessä ja ohjelmistoprojektien monimutkaisuuden kasvaessa oli syntynyt tarve kevyemmille ohjelmistokehitysmenetelmille. Näiden kevyiden menetelmien tarkoituksena oli mukautua paremmin muutoksiin, sillä perinteistä ohjelmistokehitystä tukevat mallit eivät kyenneet reagoimaan muuttuviin vaatimuksiin riittävän tehokkaasti. (Terimaa 2019, 14–16.)

Scrum on ohjelmistokehityksen ketteristä menetelmistä tunnetuin sekä yksi suosituimmista. Se on saanut nimensä Rugby pelin strategiasta, jossa pelistä poistunut pallo saadaan takaisin peliin tiimityön avulla. Scrumin luonteeseen kuuluu jatkuva prosessien kehittyminen. (Juvonen 2018, 18–19; Terimaa 2019, 17.)

3.1.1 Roolit

Juvosen (2018, 19) mukaan Scrum-menetelmässä on kolme roolia: tuotteen omistaja, Scrum-mestari ja tiimin jäsen, kun taas Teerimaan (2019, 18) mukaan Scrumissa on olemassa kuusi erilaista roolia. Nämä roolit ovat tuotteen omistaja, Scrum-mestari, tiimin jäsen, asiakas, johto ja käyttäjä. Stockmannin verkkokauppakehityksessä käytetään Teerimaan mainitsemia rooleja.

Tuotteen omistajan tulee edustaa asiakasta ja yksi hänen tehtävistään on tehdä tuotteen työlista. Työlistaa täytyy ylläpitää ja asettaa toteutettaville ominaisuuksille prioriteetit. Tuotteen omistajalla täytyy olla kykyä tehdä tuotetta koskevia päätöksiä, sillä ne ovat perusedellytyksenä tässä roolissa toimimiselle ja hän on se, joka viime kädessä päättää kehitysjonossa olevien tehtävien priorisoinnista. (Juvonen 2018, 19; Terimaa 2019, 18.)

Scrum-mestarin tehtävänä on tuntea Scrum-prosessin käytänteet ja varmistaa, että niitä noudatetaan ja tiimi toimii koko ajan parhaiden edellytysten mukaisesti. Usein Scrum-mestaria luullaan virheellisesti projektipäälliköksi tai esimieheksi, mutta sitä Scrum-mestari ei ole. Hän on se, joka kommunikoi sekä asiakkaan että kehitystiimin kanssa, ja varmistaa prosessin tuottavuuden. (Juvonen 2018, 19; Terimaa 2019, 18.)

Scrum tiimin avulla ohjelmistotuote saadaan toteutettua. Heillä on valmiudet toimia haluamallaan tavallaan, valitsemalla itse työlistalta tehtäviä, mutta kuitenkin niin, että huomioivat tehtävien prioriteetit. Lisäksi tiimi osallistuu myös tehtävien työmäärän arviointiin sekä tuotteen kehitysjonon arviointiin ja muokkaamiseen. (Juvonen 2018, 19; Terimaa 2019, 18.) Juvonen (2018, 19–20) kertoo myös, että Scrum-tiimin ideana on välttää nimettyjä roolijakoja (esim. koodaaja, ohjelmistoarkkitehti), vaan kaikki pääsisivät halutessaan tekemään kaikkea. Todellisuudessa usein käy kuitenkin niin, että tietynlaiset tehtävät menevät niille, jotka ne parhaitan osaa tehdä.

Teerimaan (2019, 18) kirjoittamiin rooleihin kuuluu vielä lisäksi asiakas, jonka tehtävänä on osallistua luomaan tuotteen kehitysjonoa silloin, kun uutta tuotetta aletaan kehittämään tai vanhaa parantamaan. Johto on Scrum-rooleista se, joka määrittelee projektissa käytettävät standardit ja käytänteet. Projektia koskevissa päätöksissä, johdolla on aina viimeinen päätäntävalta ja he osallistuvat myös projektin tavoitteiden ja vaatimusten määrittelyyn. Viimeisenä roolina on käyttäjä, jonka tehtävänä on osallistua sprinttien suunnitteluun ja arviointiin.

3.1.2 Prosessi

Scrumissa ideana on koostaa 1–4 viikon mittaisia sprinttejä eli työkokonaisuuksia. Tarkoituksena on saada toteutettua jokaisessa sprintissä jotain valmista. Projektin toteutusvaiheen alussa ei välttämättä tarvitse tietää montako sprinttiä tarvitaan, että tuote on valmis. Sprinttejä suoritetaan yksi kerrallaan, ja niin pitkään kuin on tarve tai rahaa riittää. (Juvonen 2018, 20.)

Jos projektille halutaan asettaa aika- tai kuluraja, tällöin voidaan sprinttien määrä lukita ennalta. Näissä tapauksissa tuote tehdään niin valmiiksi kuin ehditään keskittyen tärkeimpiin ominaisuuksiin. Scrum-prosessin tarkoituksena on reagoida liiketoimintaympäristön tapahtumiin, jolloin jokaisen uuden sprintin alussa voidaan suorittaa tehtävien priorisointi uudelleen. (Juvonen 2018, 20.)

3.1.3 Sprintit

Jokainen sprintti alkaa suunnittelulla, jossa työlista käsitellään tuotteen omistajan kanssa. Tuotteen omistajan tulee olla valmistautunut suunnitteluun, priorisoida työlista etukäteen ja työlistan tulee olla ajan tasalla. Suunnitteluvaiheessa mukana ovat myös Scrum-mestari sekä tiimi. Kun suunnittelussa on päästy siihen vaiheeseen, että on valittu riittävästi toiminnallisuuksia sprintissa toteutettavaksi, aloitetaan tuottamaan sprintin työlista. Työlistaa tuotettaessa toiminnallisuuksia pilkotaan pienemmiksi tehtäviksi, maksimikokoa yhdelle tehtävälle voidaan pitää karkeasti kahta työpäivää ja tätä suuremmat tulisi pilkkoa pienemmiksi tehtäviksi. Yksittäisten tehtävien kokoja suunnitellessa keskustelu saattaa edetä tekniseksi, jolloin tuotteen omistajan ei tarvitse olla välttämättä paikalla. Kuitenkin suunnittelun lopuksi tuotteen omistajalta tarvitaan vielä hyväksyntä sprintin sisällöksi. (Juvonen 2018, 21.)

Kun sprintti on suunniteltu, sen jälkeen on vuorossa työtehtävien suorittamista. Tiimin jäsenet valitsevat itselleen sprintin työlistalta tehtävän, jota alkavat työstämään. Suoritetut tehtävät merkitään tehdyksi, kun ne on saatu valmiiksi, ja tämän jälkeen tiimiläinen valitsee työlistalta uuden tehtävän. Joskus käy niin, että kaikkea suunniteltua ei saada yhdessä sprintissä tehtyä. Sprinttiä ei kuitenkaan lähdetä venyttämään, vaan ne mitä ehditään, niin toteutetaan. (Juvonen 2018, 22.)

Scrumiin kuuluu päiväpalaverit, joiden ideana on seurata sprintin aikaista tilannetta. Päiväpalaverin ohjeellinen kesto on 15 minuuttia ja se järjestetään yleensä aina samaan aikaan joka päivä. Scrum-mestari kuuntelee kaikkien tiimiläisten vastauksen näihin kolmeen kysymykseen:

1. Mitä olet tehnyt sitten edellisen palaverin?
2. Mitä aiot tehdä seuraavaksi?

3. Mitkä ongelmat tällä hetkellä haittaavat työskentelyäsi?

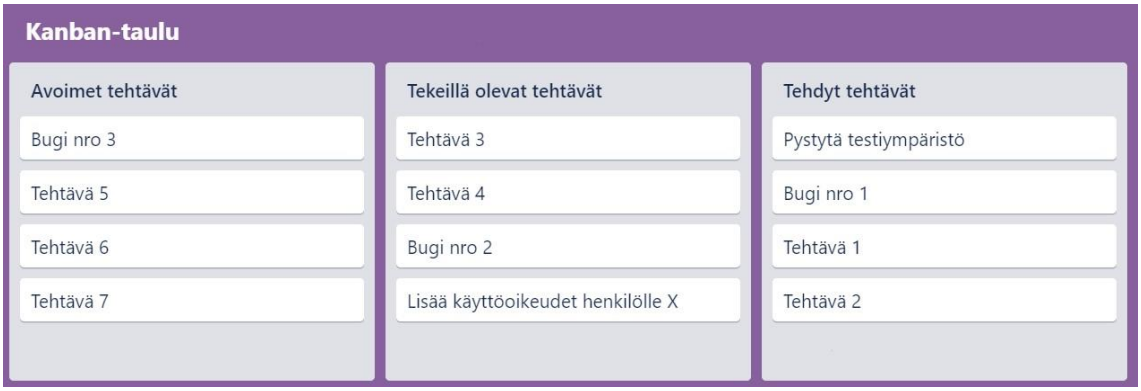
Scrum-mestarin tehtävänä on erityisesti ratkoa kolmannen kysymyksen yhteydessä tulleisiin ongelmiin. Päiväpalaverin tarkoituksena on myös mahdollistaa tiedonjako tehokkaasti ja nopeasti kaikille tiimin jäsenille. (Juvonen 2018, 22.)

Scrum prosessin viimeinen osio on sprintin jälkitarkastelu. Sprintin lopussa pidetään usein demotilaisuus, jossa esitellään sprintissä koodaamalla tehtyjä toiminnallisuuksia. Demoon voidaan kutsua projektin tuotoksista kiinnostuneita tahoja, mutta toisinaan demon voi pitää myös vain pelkästään tuotteen omistajalle. Demotilaisuuksien avulla vaatimuksia voidaan tarkentaa, selvitellä mahdollisia väärinymmärryksiä sekä nähdään mitä on tehty ja mitä voidaan tarvittaessa muuttaa. Lopuksi tiimi arvioi omaa työskentelyään, mikä meni hyvin, mikä taas heikommin ja mitä voidaan parantaa? (Juvonen 2018, 22–23.)

3.2 Kanban

Kanban on japanilainen sana, joka tarkoittaa kylttiä tai taulua. Ohjelmistoprojektissa se on simppeli menetelmä, jonka avulla on tarkoitus seurata helposti tiimin tai yksittäisen henkilön työjonoa. Kanban-tauluun kuuluu kolme eri saraketta (kuva 3):

- avoimet tehtävät
- tekeillä olevat tehtävät
- tehdyt tehtävät



Kuva 3. Kuvitteellinen esimerkki Kanban-taulusta (Trello 2019).

Kanban-taulu voidaan pystyttää esimerkiksi tiimihuoneen seinälle tai taulu voidaan luoda sovellukseen tai verkkopalveluun. Ideana on kiinnittää tauluun tehtäviä tarralapuilla. Valmiiksi tehdyt tehtävät poistuvat taululta sovittujen sääntöjen mukaan. (Juvonen 2018, 24.)

4 Verkkokauppatestauksen automatisointi

4.1 Testauksen alkuasetelmat

Toimeksiantajana tässä opinnäytetyössä toimii Stockmann Oyj Abp, joka on opinnäytetyön tekohetkellä myös kirjoittajan työnantajana. Projektin tavoitteena on ottaa käyttöön Stockmann.com-verkkokaupan manuaalisen regressiotestauksen rinnalle automaatiotestejä, jotka toteutetaan selaimeen asennettavalla Selenium IDE -työkalulla. Testauksen automatisoinnilla pyritään helpottamaan manuaalisen testaamisen parissa työskentelevien työntekijöiden työtaakkaa.

Testaustyökalun käyttöönottoprojekti koskee ICT-osaston verkkokauppatiimiä, jossa työskentelee alle kymmenen henkilöä. Verkkokaupan testaus on vain yksi osa tiimin työtehtävistä. Muita tiimin työtehtäviä ovat muun muassa verkkokaupan tuki- ja häiriönhallinta sekä muihin yrityksen projekteihin osallistuminen.

Testaustyökalun valintaprosessissa, toimeksiantajan kanssa käydyn yhteisen keskustelun tuloksena valittiin Selenium IDE. Toinen vaihtoehto olisi ollut iMacros, mutta tarkempien selvitysten jälkeen Selenium IDE vaikutti verkkokauppatiimille paremmalta vaihtoehdolta. Merkittävimmät tekijät testaustyökalun valinnassa olivat sen helppo käyttöönotto ja alhaiset käyttökustannukset. Lisäksi Selenium IDE on saanut kehuja testaustyökaluja käsittelevissä artikkeleissa, muun muassa Applitoolsin tekstissä (Applitools 2019).

Stockmannin verkkokauppaa kehitetään jatkuvasti ketteriä menetelmiä hyödyntäen, jolloin uusia versioita kaupasta tulee noin 1–3 kertaa kuukaudessa. Jokaisen uuden version jälkeen tehdään regressiotestaukset, jonka avulla varmistetaan, että verkkokauppa toimii normaalisti ja virheitä ei löydy. Stockmann käyttää

verkkokauppa kehityksessään Scrum- ja Kanban-menetelmiä ketterästi yhdistellen.

Tässä projektissa automatisoidaan yhdeksän erilaista verkkokauppatilausta. Jokaisen testitilauksen maksutapana on eri verkkopankki, jolloin saadaan testattua kaikki tällä hetkellä QA-ympäristössä saatavilla olevat verkkopankit. Näitä erilaisia verkkopankkeja on käytössä yhteensä yhdeksän, jotka muodostavat kyseiset yhdeksän testitapausta.

Tarkoituksena on tehdä verkkokauppatilausten testauspaketti, jossa testit ajamalla saadaan testattua ja todennettua, että tilaukset menevät läpi ja verkkopankit toimivat. Valituissa testitapauksissa otetaan huomioon myös se, että nämä testitapaukset ovat mahdollisimman muuttumattomia, jolloin automatisointi ei mene hukkaan. Tämän hetkisen tiedon mukaan verkkokaupan ostosprosessia ei olla muuttamassa radikaalisti, sillä iso muutos ostoskorin toimintaan tehtiin viimeksi vuonna 2018.

Luin ja Chanin (2008, 23) mukaan samojen ominaisuuksien testaaminen uudelleen ja uudelleen voi olla usein tylsää, ja tästä johtuen tässä opinnäytetyössä keskitytäänkin nimenomaan regressiotestauksen automatisointiin. Automatisoimalla alkuun ensin yhden regressiotestauksen osion eli verkkopankit, saadaan mahdollisesti helpotettua testaajien työtä, vähennettyä samanlaisten ominaisuuksien manuaalista testaamista sekä tulevaisuudessa automaatiotestejä on helpompi lähteä laajentamaan.

4.2 Testitapaukset

Tässä projektissa automatisoidaan verkkokauppatilauksia QA-ympäristössä, jotka on aikaisemmin tehty testaajan toimesta täysin manuaalisesti alusta loppuun. Testaaja on tehnyt jokaisen klikkauksen ja vaiheen itse.

Verkkokauppatilauksen tekemisessä on noin kymmenen erilaista vaihetta, jotka ovat:

- sivuston avaaminen
- sisäänkirjautuminen (tarvittaessa)

- tuotteen valinta
- tuotteen lisääminen ostoskoriin
- siirtyminen ostoskoriin
- omien tietojen täyttäminen (kirjautuneella asiakkaalla nämä ovat valmiina)
- toimitustavan valinta
- maksutavan valinta
- maksaminen
- paluu maksupalveluntarjoajalta takaisin verkkokauppaan.

Yhden manuaalisesti tehdyn testitilauksen tekemiseen menee aikaa noin 1,5–4 minuuttia riippuen testaajan nopeudesta, onko testaajalla valittu valmiiksi testituotteet ja onko maksamiseen tarvittavat tunnukset esillä. Näiden lisäksi tilausten tekemiseen menevään aikaan voivat vaikuttaa myös muut keskeytykset, kuten sähköposti- ja pikaviestintäilmoitukset tai työyhteisön sisältä tulevat keskeytykset ja kysymykset.

Valitsin automaatioprojektin testitapauksiksi verkkopankit, sillä regressiotestauksessa on tärkeää testata koko tilausprosessia ja tarkastaa, että maksutavat toimivat. Automaatiotesteillä on helppo todeta, mikäli testit epäonnistuvat, sillä uusia tilauksia ei tällöin kulje läpi verkkokaupasta ja testityökalu ilmoittaa virhetilanteesta näytöllä.

Tällainen yhdeksän verkkokauppatilauksen testauspaketti helpottaa testaajia huomattavasti, sillä aikaa vievä klikkailu vähenee huomattavasti. Olemassa olevista regressiotestitapauksista tilauksen tekemisessä on eniten vaiheita, jolloin ne ovat myös kaikista työläimpiä testata manuaalisesti. Lisäksi myös muilta ICT-osaston tiimeiltä saattaa tulla pyyntöjä verkkokauppatilauksista. Tällaisten automaatiotestien avulla pystyttäisiin myös auttamaan tarvittaessa muita tiimejä pienellä vaivalla ja muodostamaan testitilauksia nopeasti.

4.3 Testaustyökalu Selenium

4.3.1 Selenium WebDriver

Selenium on erikoistunut testauksen automatisointiin ja siltä löytyy muutamia erilaisia työkaluja. Esittelen niistä kaksi, Selenium WebDriverin sekä Selenium IDE:n, jota käytän tässä projektissa testien automatisointiin.

Selenium WebDriver on testausohjelmisto, jonka avulla voidaan automatisoida testejä eri selaimille ja eri alustoilla (Selenium 2019). Pääasiassa se on tarkoitettu web-sovellusten testaamiseen ja sen automatisointiin. Se tukee muun muassa Windows-, Linux- ja MacOS-käyttöjärjestelmiä, ohjelmointikielistä ainakin Java, C#, Ruby, Python ja Javascriptiä, sekä selaimista Chrome, Firefox, Safari, Internet Explorer ja Microsoft Edge. (Ranorex 2019.)

4.3.2 Selenium IDE

Selenium IDE on avoimen lähdekoodin testaustyökalu, jolla voi luoda, nauhoittaa ja toistaa testausskriptejä. Nauhoitettuja skriptejä voi muokata myös jälkeen päin. Työkalusta löytyy omat versiot Firefox sekä Chrome-selaimille, ja niiden asennus on käyttäjälle yksinkertaista, eikä se vaadi mitään erillisiä lisäasetuksia. (Selenium IDE 2019.)

Lisäosan vahvuuksia ovat sen nopea käyttöönotto, helppokäyttöisyys ja testien nopea ajaminen. Selenium IDE:n käyttämiseen ei tarvitse osata koodata, sillä pääosin se vain tallentaa tekemisesi selaimella ja toistaa sen tarvittaessa. (AltexSoft 2018.) Yksinkertainen käyttöliittymä on yksi työkalun hyvistä puolista, sillä se on rakennettu niin, että sitä osaa käyttää myös sellaiset testaajat, jotka ovat vasta aloittelijoita (Yarn 2016).

4.4 Työkalun käyttöönotto ja ohjeet asennukseen

Selenium IDE testaustyökalun käyttöönotto on hyvin nopea ja helppo prosessi. Ensimmäisenä siirrytään Seleniumin omille sivuille osoitteeseen https://www.seleniumhq.org/selenium-ide/, josta pääsee valitsemaan, haluaako työkalun ladata Google Chromen vai Mozilla Firefoxin selaimeen (kuva 4).

Kuva 4. Selenium IDE testaustyökalun asennus Chrome-selaimeen (Selenium IDE 2019).

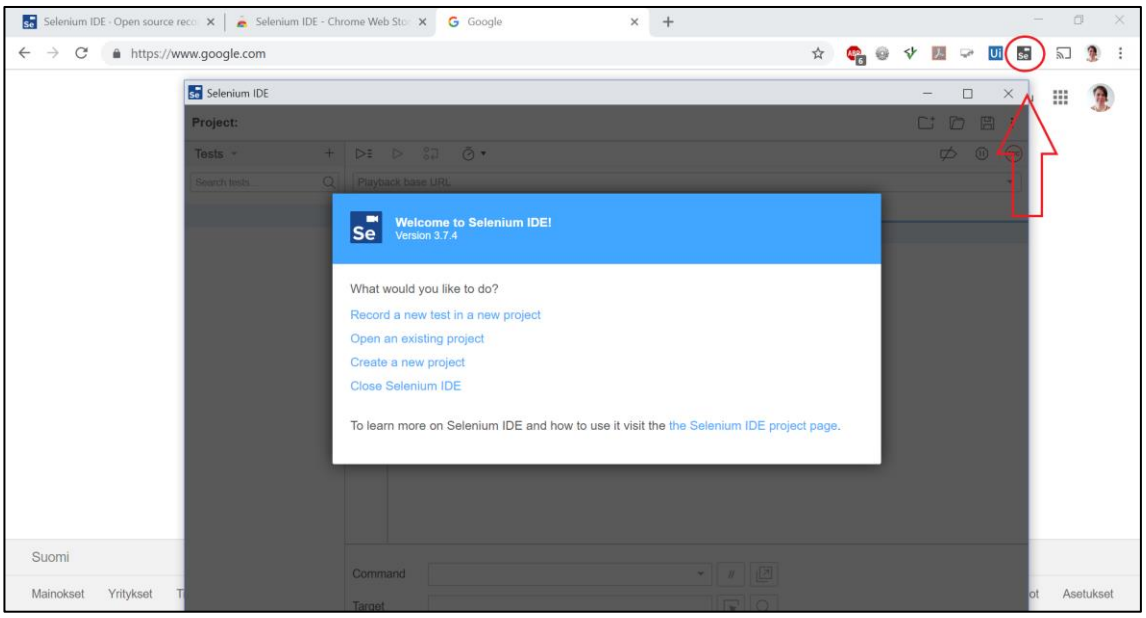
Tämän opinnäytetyön projekti toteutetaan Chrome-selaimella, joten loput asennusohjeet pätevät Chromeen asennettavan lisäosan asennukseen. Chrome Download -nappia painamalla (kuva 4) sivusto siirtyy Googlen omaan Chrome Web Storeen, josta lisäosan saa asennettua painamalla Lisää Chromeen-nappia (kuva 5).

Kuva 5. Chrome Web Storen näkymä, asennettaessa lisäosaa selaimeen (Chrome web store 2019).

Tämän jälkeen selaimeen aukeaa ponnahdusikkuna, jossa kysytään, että lisätäänkö Selenium IDE. Lisää laajennus-nappia painamalla Selenium IDE asentuu selaimeen (kuva 6).

Kuva 6. Selenium IDEn asennus lähes valmis (Chrome web store 2019).

Näiden vaiheiden jälkeen lisäosa on asennettu ja se on käyttövalmis. Työkalun saa avattua selaimen oikeasta kulmasta painamalla Seleniumin merkkiä, jossa on kirjaimet: Se (kuva 7).



Kuva 7. Selenium IDE asennettuna selaimeen ja avattuna ensimmäistä kertaa. Punainen nuoli ja ympyröity merkintä näyttävät mistä työkalun saa päälle.

4.5 Automaatiotestien luominen

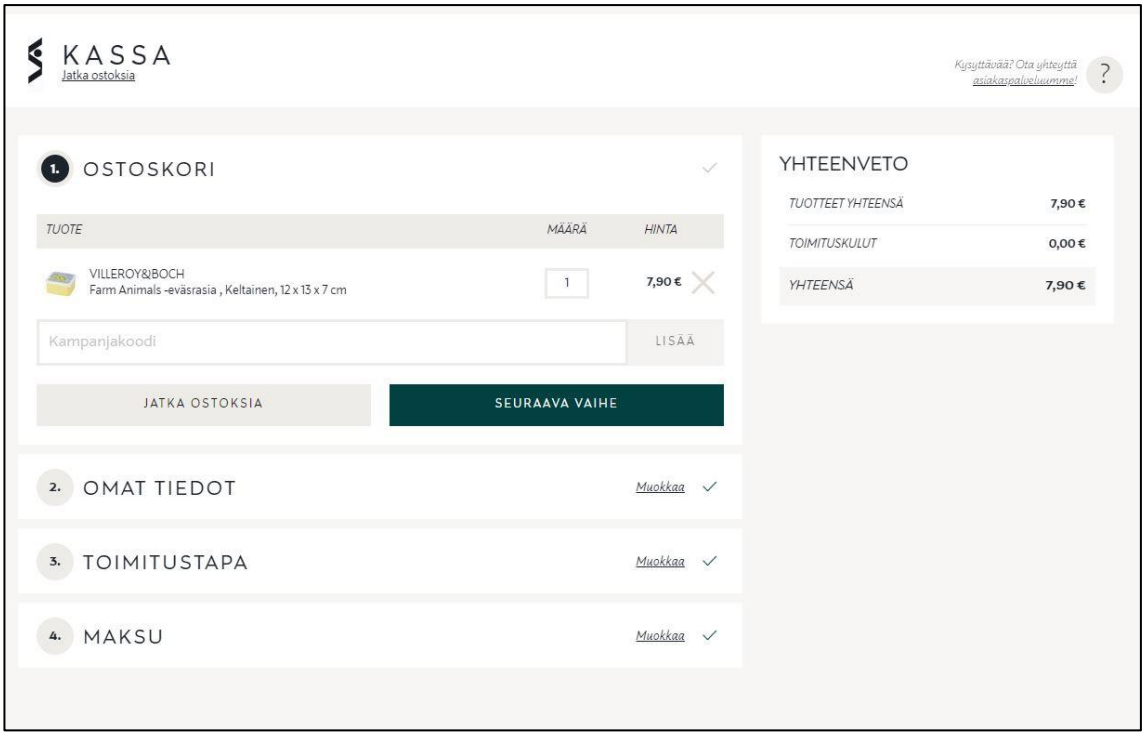
Ennen varsinaisten automaatiotestien luomista, tein alkuvalmisteluja testitapausten nauhoittamista varten. Valitsin QA-ympäristön verkkokaupasta yhden testituotteen, jota käytän alustavasti kaikissa testitilauksissa ja lisäsin tuotteelle riittävästi saatavuutta, jotta tuote ei lopu kesken testaamisen. Myöhemmässä vaiheessa testauksiin voi ottaa mukaan lisää erilaisia testituotteita. Tämän lisäksi rekisteröidyin verkkokauppaan ja loin uuden testiasiakkaan. Tällä testiasiakkaalla suoritan kaikki testitilaukset. Lopuksi otin esille kaikkien yhdeksän eri verkkopankkien testitunnukset, jotta maksaminen sujuu jouhevasti.

Käyttämäni testaustyökalu on Selenium IDE. Se nauhoittaa käyttäjän liikkeet ja painallukset, ja niiden perusteella pystyy luomaan erilaisia testitapauksia. Aluksi lähdin kokeilemaan, miten työkalu toimii, sillä aikaisempaa kokemusta kyseisestä työkalusta ei juurikaan ollut.

Ensimmäisenä määritettiin verkkokaupan URL-osoite, jonka jälkeen pystyi aloittamaan nauhoituksen. Aluksi kokeilin selata verkkokauppaa ja katsoa, että millaisia komentoja työkalu nauhoittaa. Lisäksi kokeilin muokata niitä, sekä luoda komentoja ilman nauhoitusta. Kun olin saanut selkeän kuvan työkalun toiminnoista, lähdin nauhoittamaan ensimmäistä verkkokauppatilausta.

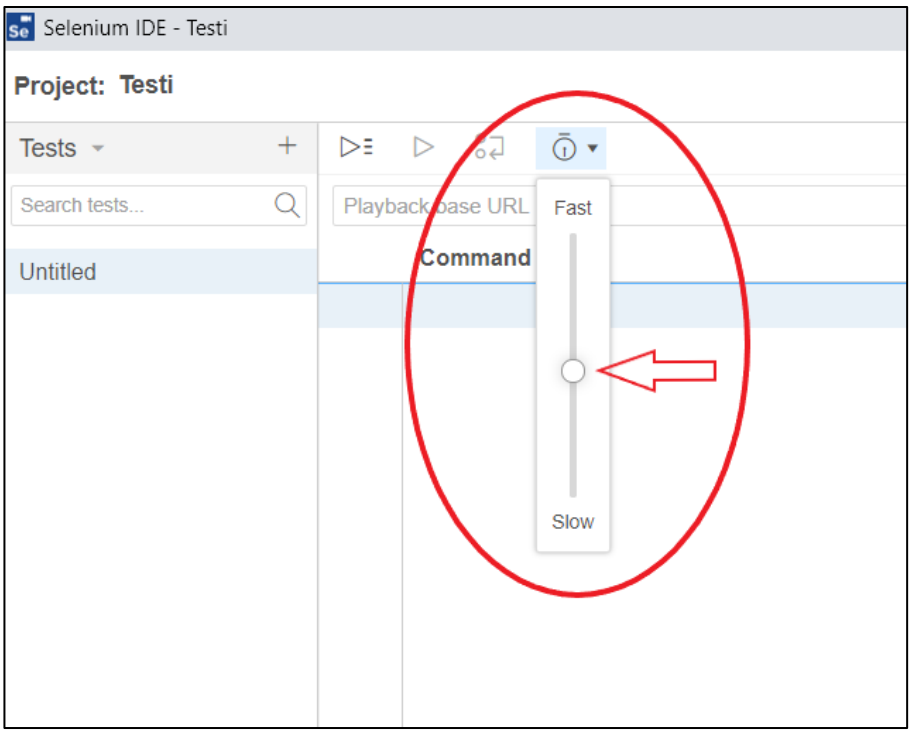
Heti ensimmäisen testitilauksen tekovaiheessa törmäsin haasteisiin. Kun sain nauhoitettua tilauksen valmiiksi ja yritin toistaa testiä, testitapaus jäi jumiin ja tuli ilmoitus virheestä. Aluksi oli vaikea ymmärtää, mikä oli pielessä, sillä testitapaus vaikutti olevan oikein, mutta työkalu toimi liian nopeasti.

Selenium IDE oli asettanut automaattisesti testien ajonopeudeksi kaikista nopeimman. Ongelma oli siis testien ajonopeudessa, sillä verkkokauppa ei pysynyt Selenium IDEn perässä. Verkkokaupan hitaimmat kohdat löytyvät kassan eri vaiheista, joita on yhteensä 4 kappaletta: Ostoskori, omat tiedot, toimitustapa ja maksu (kuva 8). Hitauden pystyy todentamaan silmämääräisesti, sillä kaupan kassan vaiheet latautuvat auki hitaammin. Joskus käyttäjä joutuu odottamaan muutamia sekunteja, että osio aukeaa ja pystyy etenemään seuraavaan kassan vaiheeseen.



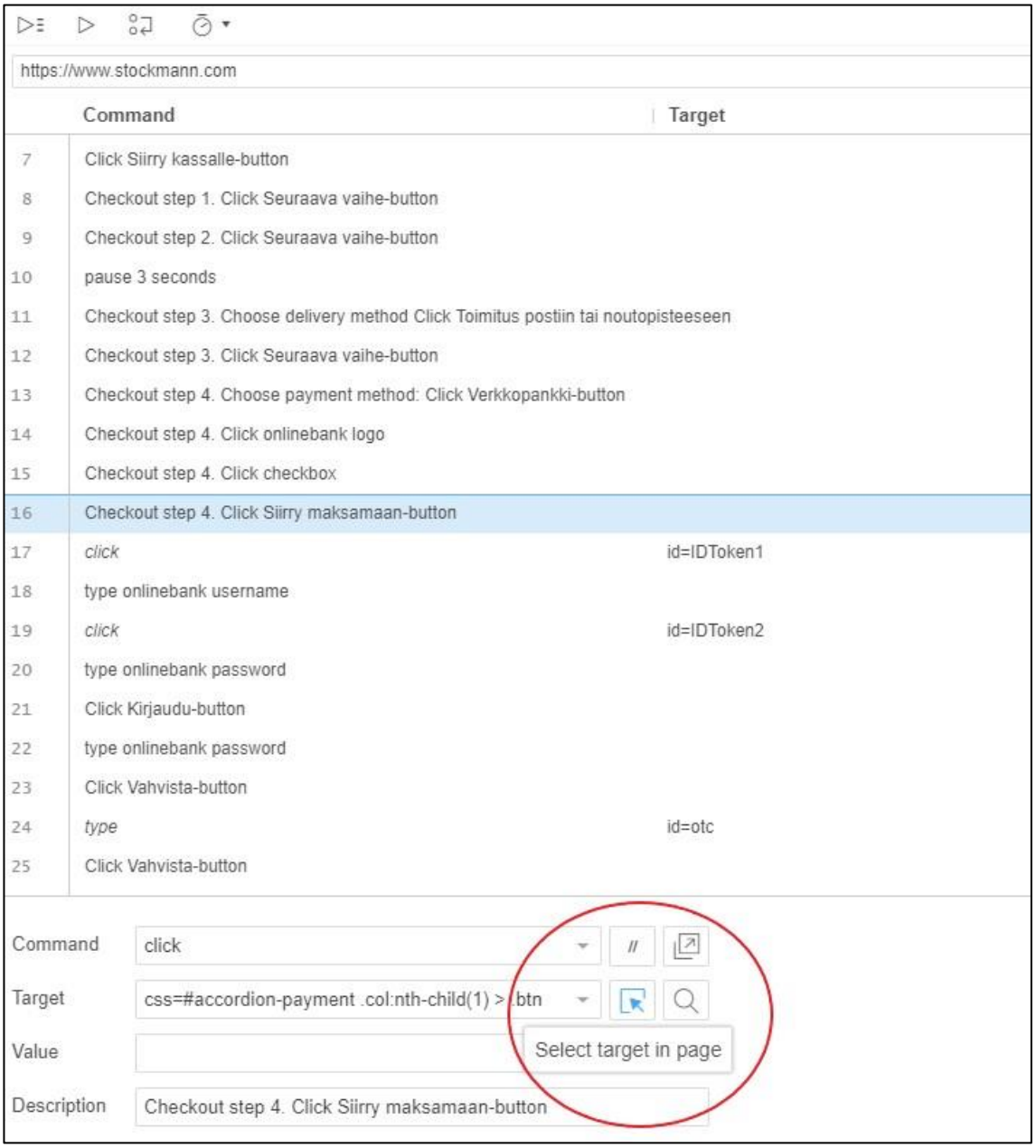
Kuva 8. Stockmann verkkokaupan (QA-ympäristö) kassan neljä eri vaihetta: 1. Ostoskori, 2. Omat tiedot, 3. Toimitustapa ja 4. Maksu.

Vaihdoin testien ajonopeuden nopeimmasta puolet hitaammalle (kuva 9), jonka jälkeen sain ensimmäisen kerran yhden testitilauksen onnistuneesti läpi.



Kuva 9. Selenium IDEn testien ajonopeuden säädin säädetty puoleen väliin mittaria.

Kun ensimmäinen verkkokauppatilaus onnistui, oli helppo lähteä toteuttamaan seuraavia. Kopioin ensimmäiseksi tehdyn onnistuneen testitapauksen ja lähdin muokkaamaan ainoastaan maksutapaa. Aluksi kokeilin nauhoittaa uudestaan loppupään vaiheet, mutta huomasin, että helpompi tapa oli muuttaa testikomentojen kohteita eli targetteja, pelkästään klikkaamalla ja vaihtamalla uudet oikeat kohteet tilalle (kuva 10).



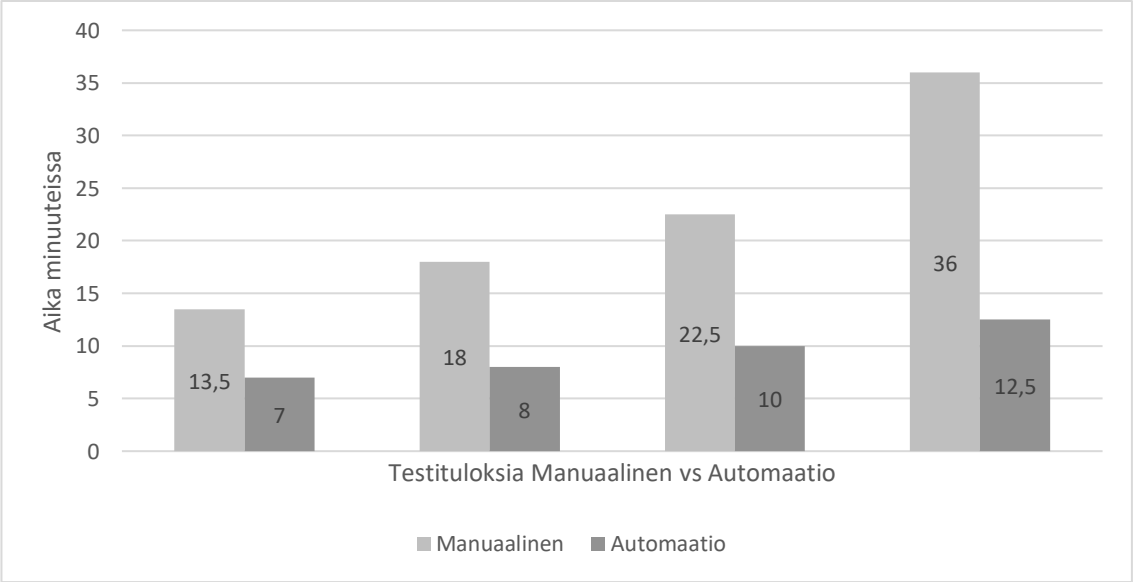
Kuva 10. Selenium IDEn testikomentojen kohteiden vaihtamisen voi suorittaa valitsemalla verkkosivulta oikean osan käyttämällä *Select target in page*-toimintoa.

4.6 Tulokset

Testitulokset osoittavat, että automaatiotestit toimivat pääosin virheettömästi. Automaatiotestejä saatiin ajettua useita kymmeniä kertoja ja näin ollen testituloksia on helpompi tarkastella ja vertailla manuaaliseen testaamiseen. Selenium IDE:llä ajetuissa automaatiotesteissä tuli muistaa aina asettaa testauksen ajonopeus riittävän alhaiseksi, jotta testit eivät menneet virhetilanteeseen. Muutamia kertoja automaatiotestejä ajaessa, yksi tai kaksi testitapausta jäivät virhetilanteeseen, jolloin testituloksia ei voinut verrata keskenään. Usein syynä oli liian nopea testien ajonopeus ja näistä virheistä päästiin eroon, kun ajonopeutta vähennettiin.

Ensimmäisellä kerralla, kun valmis testauspaketti ajettiin läpi, aikaa kului yhdeksän verkkokauppatilauksen muodostamiseen 12 minuuttia ja 36 sekuntia. Tämän tuloksen jälkeen testejä vielä muokattiin ja paranneltiin lisää. Parannusten jälkeen testitulokset ovat olleet noin 7-8 minuuttia, riippuen testaustyökaluun säädetystä ajonopeudesta. Keskimääräisesti yhden tilauksen tekemiseen Selenium IDE:llä menee noin yksi minuutti, mutta tämän hetkisten testitulosten perusteella lukua pystyisi alentamaan, jopa 50 sekuntiin.

Kuviossa 1 esitetään manuaalisesti tehtyjä testituloksia sekä automaatiotyökalulla tehtyjä testituloksia. Vaalean harmaalla pohjalla näkyy manuaalisesti tehtyjen tilausten tuloksia ja tumman harmaalla automaatiotestien tuloksia.



Kuvio 1. Testituloksia manuaalisesti tehdyistä ja automaatiotyökalu Selenium IDE:llä tehdyistä testeistä.

Kuviossa 1 manuaaliset tulokset on laskettu sen mukaan, kuinka kauan testaajalla keskimääräisesti menee aikaa yhden testitilauksen tekemiseen. Ensimmäisessä manuaalisessa palkissa on laskettu, että yhden tilauksen tekemiseen menisi aikaa 1,5 minuuttia, jolloin kokonaisuutena yhdeksän tilauksen tekemiseen menisi aikaa 13,5 minuuttia. Toisessa manuaalisessa palkissa tilauksen tekemiseen menisi aikaa 2 minuuttia per tilaus, jolloin kokonaisaika olisi 18 minuuttia. Kolmannessa palkissa on laskettu yhden tilauksen tekoajaksi 2,5 minuuttia ja näin ollen kokonaiskesto olisi 22,5 minuuttia. Viimeisessä manuaalisessa palkissa yhden tilauksen tekemiseen on laskettu menevän aikaa 4 minuuttia, jolloin kokonaiskesto olisi 36 minuuttia.

Automaatiotestien tulokset on saatu ajamalla testejä Selenium IDE:llä ja pyöristämällä lukuja helpommin tarkasteltavaan muotoon. Kuviossa 1 tumman harmaalla pohjalla olevat palkit esittävät automaatiotestien tuloksia. Paras tulos mitä tähän mennessä on saatu, on ollut ajaltaan 7 minuuttia ja 4 sekuntia. Kuviossa 1 se on pyöristetty lähimpään minuuttiin, joten ensimmäinen automaatiopalkki havainnollistaa tätä. Kuten tämän kappaleen alussa kerrottiin, keskimääräisesti automaatiotestitulokset ovat vaihdelleet 7 ja 8 minuutin välillä. Toinen automaatiopalkki esittää 8 minuutin testitulosta ja kolmas palkki esittää 10 minuutin testitulosta. Viimeinen palkki on pyöristetty 12 minuuttiin ja 30 sekuntiin, ja se esittää pylväänä tämän hetkisen huonoimman automaatiotestin testitulosta, joka oli 12 minuuttia ja 36 sekuntia.

Kuviossa 1 esitetyistä palkeista voidaan päätellä, että automaatiotestit vievät huomattavasti vähemmän aikaa, kuin manuaalisesti tehdyt testit. Jos tarkastellaan nopeinta automaatiotestiä, jonka kesto on 7 minuuttia ja hitainta manuaalista testiä, jonka kesto on 36 minuuttia, voidaan todeta, että automaatiotesti on noin neljä kertaa nopeampi vaihtoehto. Nopein manuaalisesti tehty testaus kestää noin 13,5 minuuttia ja hitain automaatiotesti 12,5 minuuttia. Automaatiotestejä ajettaessa tuloksena saadaan keskimääräisesti aina nopeampia tuloksia, kuin manuaalisesti tehdyissä.

Yksi tavoite oli jakaa automaatiotestejä myös muille verkkokauppatiimin työntekijöille. Tähän mennessä, kun opinnäytetyön kirjoitusprosessi on loppumaisillaan, on testauspaketti esitelty ja jaettu yhdelle työntekijälle. Lopuille tiimiläisille aihe on tarkoitus esitellä syksyllä 2019 ja jakaa testausmateriaali halukkaille.

5 Pohdinta

Tämän opinnäytetyön tarkoituksena oli tuottaa toimeksiantajalle regressiotestauksen rinnalle automaatiotestejä, jotka tuli toteuttaa selaimeen asennettavalla Selenium IDE -työkalulla. Tällä pyrittiin helpottamaan manuaalisen testaamisen parissa työskentelevien työntekijöiden työtaakkaa, vähentämällä jatkuvaa manuaalisten verkkokauppatilausten tekoa. Lisäksi tavoitteena oli saada toimiva yhdeksän verkkokauppatilauksen testauspaketti, jota pystyisi jakamaan muille verkkokauppatiimiläisille.

Verkkokauppatilausten testauspaketin luominen tavoitteisiin nähden onnistui hyvin ja testit toimivat luotettavasti. Olisin voinut laajentaa opinnäytetyötä vielä tekemällä useita erilaisia testitapauksia ja käyttämällä laajemmin erilaisia tuotteita, toimitustapoja tai maksutapoja. Tämä olisi kuitenkin voinut aiheuttaa aikataulullisia riskejä, ja projektin koko olisi saattanut muodostua liian laajaksi. Tästä johtuen pitäydyin alkuperäisessä suunnitelmassa ja tein sovitun tavoitteen mukaisen testauspaketin. Nyt kun ensimmäiset automaatiotestit on luotu onnistuneesti, on tämän jälkeen helpompi lähteä laajentamaan testitapauksia ja lisätä niihin tulevaisuudessa muitakin regressiotestauksen osia. Näiden automaatiotestien avulla voidaan lopulta säästää testaajien aikaa ja siirtää sitä muuhun tekemiseen.

Verkkokaupan testaamisesta yleisesti tai siihen liittyvistä aiheista oli vaikea löytää tutkimuksia. Opinnäytetöitä on tehty useita, mutta niissä on yleensä ollut toimeksiantaja mukana ja tutkimuksessa on tarkasteltu esimerkiksi regressiotestausta kyseisen yrityksen näkökulmasta. Vuonna 2004 on tehty Kuopion yliopistossa Juha Holopaisen toimesta regressiotestauksen tehostamisesta selvitys. Selvityksen tulokset regressiotestauksen automatisoinnista vahvistavat tämän opinnäytetyön tuloksia. Tuloksissa kerrotaan regressiotestauksen automatisoinnin eduista. Yhtenä etuna nähtiin, että automaatio mahdollistaa täysin saman testitapauksen toistamisen uudelleen ja uudelleen. Lisäksi virheiden löytäminen ja korjaaminen on helpompaa, sillä automaation avulla testaaja pystyy heti tarkistamaan, että korjaukset todella toimivat.

Tämän työn automaatiotestauspaketin tekoon meni paljon enemmän aikaa, kuin manuaalisesti testit tehtyinä. Testauksia suoritetaan kuitenkin lähes viikoittain, joten uskon, että ne alkavat maksamaan itseään takaisin ajan myötä.

Aikaisempi kokemus manuaalisesta verkkokaupan testaamisesta Stockmannilla antoi mielestäni hyvän lähtökohdan kirjoittaa opinnäytetyötä juuri tästä aiheesta. Testaustyökalu Selenium IDE:n käyttöönotto ja sen käyttöön tutustuminen oli mielenkiintoista ja opettavaista. Se antaa jatkossa työelämässä ohjelmistotestaukseen lisäapuja. Minulla ei ollut aikaisempaa kokemusta testausautomaatiosta, joten opin paljon uutta matkan varrella ja tekeminen oli mielekästä. Uskon, että tulevaisuudessa testausautomaation käyttö lisääntyy entisestään, sillä sen avulla yritykset voivat tehostaa testaamista ja tehdä mahdollisesti säästöjä työntekijöiden kustannuksissa.

Lähteet

AltexSoft. 2018. The Good and the Bad of Selenium Test Automation Tool. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-selenium-test-automation-tool/. 10.5.2019.

Ammann, P. & Offutt, J. 2008. Introduction to Software Testing. New York: Cambridge University Press. http://www.cse.hcmut.edu.vn/~hiep/KiemthuPhanmem/Tailieutham-khao/Introduction%20to%20Software%20Testing.pdf. 8.5.2019.

Applitools. 2019. 16 reason why to use Selenium IDE in 2019 (and 2 why not). https://applitools.com/blog/why-selenium-ide-2019. 7.8.2019.

ATR Soft. 2018. Testausautomaatio tehostaa laadunvarmistusta. https://www.atrsoft.com/testausautomaatio-tehostaa-laadunvarmis-tusta/.13.6.2019.

Chrome web store. 2019. Selenium IDE. https://chrome.google.com/webs-tore/detail/selenium-ide/mooikfkahbdckldjjndioackbalphokd.

Craig, R.D. & Jaskiel, S.P. 2002. Systematic Software Testing. Artech House Computer Library. http://repository.unimal.ac.id/286/1/2002%20Sys-tematic%20Software%20Testing_ok.pdf. 6.5.2019.

Dustin, E. & Garrett, T. & Gauf, B. 2009. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley Professional. https://learning.oreilly.com/li-brary/view/implementing-automated-software/9780321619600/. 8.5.2019.

Haikala, I. & Märijärvi, J. 2004. Ohjelmistotuotanto (10. painos). Helsinki: Talen-tum.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Helsinki: Ta-lentum.

Holopainen, J. 2004. Regressiotestauksen tehostaminen. Kuopion Yliopisto. Tietojenkäsittelytieteen laitos. Selvitys.

International Software Testing Qualifications Board. 2010. Sertifioitu Testaaja: Perustason sertifikaattisisältö. http://www.fistb.fi/sites/fistb/files/liit-teet/FL%20Syllabus%2020101123_0.pdf. 6.5.2019.

Juvonen, R. 2018. Ohjelmistoprojektin sudenkuopat ja miten ne vältetään. Hel-sinki: Books on Demand.

Kasurinen, J.P. 2013. Ohjelmistotestauksen Käsikirja (e-Pub-versio). Jyväskylä: Docendo.

Loveland, S., Miller, G., Prewitt, R. & Shannon, M. 2005. Software Testing Techniques: Finding the Defects that Matter.

Lui, K.M. & Chan, K.C.C. 2008. Software Development Rhythms: Harmonizing Agile Practices for Synergy. Hong Kong: John Wiley & Sons.

Pressman, R.S. 2010. Software Engineering: A Practitioner's Approach (7. painos). New York: McGraw-Hill. http://dinus.ac.id/reposi-tory/docs/ajar/RPL-7th_ed_software_engineering_a_practitioners_ap-proach_by_roger_s._pressman_.pdf. 7.5.2019.

Ranorex. 2019. Selenium WebDriver Guide. https://www.ranorex.com/re-sources/testing-wiki/selenium-testing/. 4.6.2019.

Selenium. 2019. Introduction: Test Automation for Web Applications. https://www.seleniumhq.org/docs/01_introducing_selenium.jsp. 20.8.2019.

Selenium IDE. 2019. Selenium IDE. https://www.seleniumhq.org/selenium-ide/. 10.5.2019.

Smart Education. 2019a. Testaus lyhyesti. Jyväskylän Yliopiston Informaatioteknologian tiedekunnan monitieteinen tutkimus- ja koulutusalue. http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testaus-lyhyesti. 6.5.2019.

Smart Education. 2019b. Testauksen tasot. Jyväskylän Yliopiston Informaatioteknologian tiedekunnan monitieteinen tutkimus- ja koulutusalue. http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot 7.5.2019.

Sommerville, I. 2007. Software Engineering (8. painos). China: Pearson Education. https://doc.lagout.org/science/0_Computer%20Science/Software%20Engineering%2C%208th%20Edition.pdf. 7.5.2019.

Sommerville, I. 2011. Software Engineering (9. painos). Boston: Pearson Education. https://inspirit.net.in/books/academic/Ian%20Sommerville%20Software%20Engineering,%209th%20Edition%20%20%20%202011.pdf. 7.5.2019.

Suomen Standardisoimisliitto SFS ry. 2019. Ohjelmistotuotanto ja järjestelmäkehitys. https://www.sfs.fi/standardien_laadinta/sfs_n_standardisointiryhmat/it-standardisointi/it_-_aihealueet/ohjelmistotuotanto_ja_jarjestelmakehitys. 6.5.2019.

Tech Target: Search Software Quality. 2007. Bug. https://searchsoftwarequality.techtarget.com/definition/bug. 9.5.2019.

Terimaa, T. 2019. Käyttäjäkokemuksen suunnittelu ketterässä ohjelmistokehityksessä. Jyväskylän Yliopisto. Informaatioteknologian tiedekunta. Kandidaatintutkielma.

Trello. 2019. https://trello.com/. 9.8.2019.

Yarn, J. 2016. Selenium IDE: The Good, the Bad, and the Ugly. https://www.lucidchart.com/techblog/2016/09/13/selenium-ide-the-good-the-bad-and-the-ugly/. 13.6.2019.