Haaga-Helia
ammattikorkeakoulu Oy

Regressiotestauksen suunnittelu ja toteutus

Rami Suomalainen

Opinnäytetyö

Tietojenkäsittelyn

koulutusohjelma

2017

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Tekijä
Rami Suomalainen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön nimi
Regressiotestauksen suunnittelu ja toteutus

Sivu- ja liitesivumäärä
47

Testaamisella tarkoitetaan työtä, jota tehdään sen varmistamiseksi, että toteutettavasta ohjelmistosta tai esimerkiksi verkkosivusta tulee toivotun kaltainen. Lyhyesti testauksen määritelmä voidaan tiivistää yhteen lauseeseen: varmistetaan, että tehdään oikeaa tuotetta ja, että tuote on tehty oikein. Testauksella voidaan todeta, että ohjelmisto täyttää kaupalliset- ja tekniset vaatimukset ja ohjelmisto toimii odotetusti.

Ohjelman laadunvarmistamiseksi testaaminen on erittäin tärkeä osa ohjelmistokehitystä. Tässä opinnäytetyössä kerrotaan, miksi testaamista suoritetaan sekä mitä tämä on työelämässä. Ajoittain törmää sovelluksiin, jotka ovat siirtyneet tuotantoon viallisena, koska testaamista ei ole suoritettu kunnolla loppuun asti tai testaamista ei suoritettu jopa ollenkaan. Tämä on aiheuttanut ylimääräistä työtä yritykselle sekä nostanut myös kustannuksia, joita hyvällä testaamisella olisi voinut pienentää.

Regressiotestauksella suoritetaan jonkin ohjelman uudelleentestaus tai testausprosessin viimeinen vaihe, jolla etsitään viimeisetkin virheet ohjelmistosta. Jos testattu alusta läpäisee regressiotestauksen on alusta valmis tuotantoon vientiin.

Opinnäytetyössä avataan näitä asioita lukijalle, miksi testaamista suoritetaan ja mitä testaaminen oikein on. Käsiteltäviä aiheita ovat yleisjohdanto testaamiseen, testauksen suunnittelu, testauksen teoria, testausmenetelmät, testitapaukset, kyselytutkimus sekä vastauksista kerätty yhteenveto. Lopussa vielä kerätään opinnäytetyössä onnistuneet asiat sekä käydään läpi, mikä olisi voinut mennä paremmin.

Testaamista suoritetaan opinnäytetyössä manuaalisesti sekä automaatiotestauksella. Automaatiotestausta suoritetaan opinnäytetyössä Seleniumilla, Robot Frameworkilla sekä Jenkinsillä. Kysely on lähetetty henkilöille, jotka työskentelevät testaajina.

Asiasanat
Testaaminen, regressiotestaus, automaatiotestaus, manuaalinen testaus, testausmenetelmät

Sisällysluettelo

1 Johdanto

Opinnäytetyön aiheena on regressiotestauksen suunnittelu ja toteutus. Tässä opinnäytetyössä käydään läpi testaamisen suunnittelun vaiheita, miten testaamista suoritetaan, miten havaittujen virheiden kanssa tulee toimia, testauksen teoriaa, eri testausmenetelmiä, suoritetaan testitapauksia sekä suoritetaan kyselytutkimus, jossa käydään läpi mitä testaaminen on työelämässä. Itse testaamista suoritetaan opinnäytetyössä manuaali- ja automaatiotestauksella. Opinnäytetyö on tehty myös siksi, että testaajan työ on joidenkin mielestä jokseenkin aliarvostettua. Testaaja voidaan myös nähdä työpaikoissa riesana varsinkin ohjelmistokehittäjien mielestä, koska testaajathan etsivät virheitä heidän työstään. Tämä voi usein aiheuttaa erimielisyyksiä testaajan ja ohjelmoijan välille.

”Kukaanhan ei ole niin etevä, että kokonaisen järjestelmän osaisi täysin sokkona ohjelmoida ilman, että tarvitsee mitenkään suorittaa kehitysympäristössään sitä järjestelmän osaa, jota on kehittämässä”. (Berglund 2015.)

1.1 Yleisjohdanto

Testaus on yksi ohjelmistotuotannon piiriin kuuluva kokonaisuus. Testaus on kuitenkin kokonaisuutena paljon laajempi kuin ohjelmointityö tai tekninen kirjoittaminen. Testaaja joutuu työssään tekemään montaa erilaista asiaa, kuten kirjoittaa koodia, tekemään dokumentaatiota tai vaikkapa pitämään koekäyttäjien haastatteluja riippuen siitä, millaista testausta halutaan tehdä. Tästä johtuen testaajan työ voi vaihdella paljonkin erilaisten ohjelmointitalojen välillä. Peliyritys on varmasti kiinnostunut kokeilemaan, että sen kaikki graafiset elementit toimivat ja tämän parissa on hauskaa viettää aikaa. Useat tutkimukset osoittavat, että testauksen osuus ohjelmistokehitysprosessin kustannuksista on keskimäärin noin 30 prosenttia kokonaiskustannuksista. (Kasurinen 2013, 10; Savolainen 2005.)

Testauksella saadaan tietoa ohjelmiston toimivuudesta sekä laadusta. Testaaminen tarjoaa myös eri näkökulman ohjelmistoon, jonka avulla asiakas pystyy ymmärtämään paremmin ohjelmiston riskejä. Ohjelmistotestauksen avulla voidaan myös todeta, että ohjelmisto täyttää kaupalliset- ja tekniset vaatimukset, ohjelmisto toimii odotetusti tai ohjelmistoa voidaan vielä muokata halutulla tavalla.

Testaaminen voidaan suorittaa missä vaiheessa ohjelmointia tahansa. Yleisin vaihe on ohjelmoinnin jälkeen, kun todetaan ohjelmiston täyttävän määritellyt vaatimukset. Eri osa-

alueiden testaaminen on myös mahdollista kesken ohjelmistokehityksen. Ohjelmistokehityksen kannalta voidaan miettiä kolminaisuutta, jossa testaaminen esittää tärkeää roolia: vaatimukset, tekninen suunnittelu ja toteutus sekä testaaminen. Ensin suunnitellaan ohjelmiston vaatimukset. Tämän jälkeen suunnitellaan tekniset vaatimukset ja toteutus. Testaaminen on viimeinen osa tätä kokonaisuutta, mutta ohjelma joka ei läpäise testiprosessia voi joutua takaisin suunnitteluvaiheeseen. Testaustyötä voidaan pitää myös jatkuvana vertailutehtävänä. Testauksen työ on tarkastaa se, että mitä on saatu aikaiseksi vastaa sitä, mitä on ollut tarkoituksena tehdä sekä tunnistaa ne kohdat joissa poiketaan suunnitelmasta. (Kasurinen 2013, 10; Pyhäjärvi 2006.)

1.2 Tavoitteet

Tavoitteena on kasvattaa lukijan tietoa ja mielenkiintoa testaamista kohtaan. Opinnäytetyö voisi toimia tietopakettina henkilölle, joka on kiinnostunut testaajan työstä, mutta kenellä ei ole vielä laajaa tietämystä testaamisesta. Testaajan työ on monelle tuntematon aihe, joten tärkeimpänä tavoitteena on avata lukijalle mitä testaaminen työelämässä on, mitä testaajan on hyvä osata sekä mitä testaaminen pitää sisällään. Tätä varten on suoritettu kyselytutkimus testaajille, jotta pystytään ammattilaisten mielipiteiden kautta kertoa paremmin lukijalle testaamisesta.

Toinen tavoite opinnäytetyölle on tekijän henkilökohtaisella tasolla oppia automaatiotestauksen suorittamista ja suorittaa eri testitapauksia onnistuneesti. Opinnäytetyöntekijä tavoittelee sellaisia testauskokemuksia, joista on hyötyä myös työelämässä.

1.3 Testaajan työ

”Ohjelmistotestaajan paras ominaisuus on uteliaisuus. Pitää uskaltaa heittäytyä ja uskaltaa kysyä myös ne tyhmät kysymykset, joita muut eivät uskalla kysyä. Se kun saa testattavan lopputuotteen niin rikki, että se pitää asentaa kokonaan uudelleen, siitä saa aikaan mehukkaita keskusteluja kehitysporukan kesken. On tärkeää, että bugit saadaan korjattua. Pitää ymmärtää, mitä ongelmia virheestä olisi voinut seurata, jos se olisi jäänyt ohjelmistoon.” (Niittyviita 2012.)

Jos testaaja havaitsee monta virhettä ohjelmistossa, hän on tehnyt työnsä hyvin. Testaajan tärkein tehtävä on löytää ohjelmistosta virheitä eli bugeja. Bugien korjaaminen ohjelmistossa ennen julkaisua ja käyttöönottoa voi säästää yritykselle paljon rahaa. Tämä voi olla ohjelmoijan näkökulmasta

rasittavaa, koska ohjelmoija ei välttämättä halua kuulla, että hän on tehnyt virheen tai näkemyseroja voi tulla testaajan kanssa. Kun ohjelma saapuu testaajalle ja virheitä havaitaan paljon niin tulee muistaa, että ohjelma on saapunut jo valmiiksi viallisena testaukseen.

“Testing is the process of executing a program with the intent of finding errors. Testing is the process of establishing confidence that a program does what it supposed to do.” (Meyers 2004.)

1.4 Mitä testaaminen ei ole

Työelämässä on yleensä vääriä käsityksiä siitä, mitä testaajan työ on. Näitä käsityksiä voi kohdata työssä henkilöiltä, jotka eivät ole aikaisemmin työskennelleet testaajien kanssa tai ketkä eivät ymmärrä testaajien roolia. Alla olevassa listassa on esimerkkejä mitä testaaminen ei ole:

- Jotain jota ainoastaan erikseen nimetyt henkilöt (testaajat) suorittavat
- Sarja helppoja ja suoraviivaisia toimintoja
- Samojen asioiden toistamista uudestaan ja uudestaan (Testaus voi tosin olla samanlaista erilaisissa tilanteissa varsinkin manuaalisessa testaamisessa)
- Testaaja käy läpi vain toiminnallisuuksien läpikäyntiä
- Testaaja ei ole vastuussa ohjelmiston laadusta

(Pyhäjärvi 2006.)

1.5 Testauksen historia

Testaaminen on ollut olemassa niin kauan kuin ohjelmistotuotantokin. Testaaminen arvioidaan alkaneeksi 1950–luvulla. Ensimmäiset testitapaukset luotiin paperille ja testaaminen oli monimutkaista algoritmien oikeellisuuden todistamista ja testaamista. Tällöin tätä kutsuttiin lasilaatikkotestaamiseksi. Suurta huomioita ei testaamiseen kiinnitetty, koska rajallinen määrä testaustoimenpiteitä pystyi kattamaan koko järjestelmän. Ohjelmistotestaus suoritettiin lähes aina viimeisenä vaiheena ohjelmistotuotannossa, eikä testausta silloin jaettu eri osiin.

Uusi aikakausi testaamisessa alkoi kaupallisen ohjelmistokehityksen ja PC (Personal Computer) laitteiden markkinoille saapumisen myötä. Online–järjestelmät, asiakas-palvelimet ja graafiset käyttöliittymät (GUI) toivat

testaamiselle aivan uusia haasteita. Merkkipohjaiset järjestelmät olivat muuttuneet graafisiksi ja testattavana olivat syötteiden sijaan graafiset komponentit ja ominaisuudet. Ajan myötä testaus- ja testaustyökalut ovat kehittyneet. Kehityksiä voidaan myös mainita olevan testausmenetelmissä ja prosesseissa. (Savolainen 2005.)

1.6 Termit

Testattavuus - Miten hyvin ohjelmiston pystyy jakamaan eri osiin. Testattavuus tulee ottaa huomioon jo ohjelmistokehityksestä alkaen. (Niittyviita 2012.)

Testiajo - Yhden tai useamman testitapauksen suorittaminen koottuna yhdeksi kokonaisuudeksi (eli yhdeksi ajoksi).

Testitapaus - Määrittelee lähtötilanteen eli alkuehdot testitapauksen suoritukselle. Määrittelee syötteet ja odotetut tulokset. Onnistuneella testitapauksella on todennäköistä paljastaa aiemmin tuntemattomia vikoja ohjelmistossa. (Niittyviita 2012.)

Selenium – Seleniumia käytetään yleensä C#, Groovy, Java, Perl, PHP, Python, Ruby ja Scale kielillä tehtyjen ohjelmistojen testaamiseen. Seleniumia käytetään tunnetuimmissa käyttöjärjestelmissä Windows, Linux ja Mac OS. (Seleniumhq 2016.)

Robot Framework – Robot Framework on automaatiotestauksen sovelluskehys. Sovelluskehysten on tarkoitus yhtenäistää tapaa, jolla ohjelmiston kehittäjät käskyttävät ohjelmistorajapintoja ja ohjelmistoja. Yhtenäinen tapa kirjoittaa testejä merkitsee sitä, ettei testien kirjoittajan tarvitse opetella useita eri tapoja tehdä asioita. (Nisbet 2012.)

Jenkins – Jenkins on avoimen lähdekoodin automaatiotestaus sovellus, jonka avulla voidaan testata ja julkaista eri ohjelmistoja. Käytetään paljon myös ohjelmistokehityksessä. (Meet Jenkins 2016.)

XPath – (XML Path Language) Käytetään XML–dokumenttien eri osien osoittamiseen ja XML–dokumentin rakenteen tiedon luontiin. XPathilla mahdollistetaan eri dokumentin osien poimintaa tietyillä valintakriteereillä. (W3Schools 2014.)

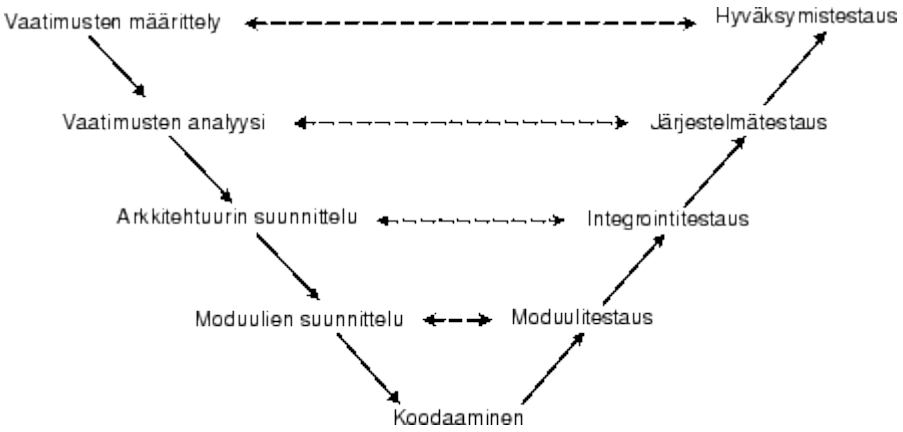
2 Testauksen suunnittelu

Tässä osiossa kerrotaan, mitä testaamisen suunnittelussa tulee huomioida sekä miten testaaminen on jaettu eri osiin sekä vaiheisiin. Eri testauksen vaiheista kerrotaan teoriana sekä näiden rooli ja tarkoitus testausprosessissa. Vaiheet esitetään sille paikalle, jossa ne prosessissa sijaitsevat. Osiossa kerrotaan myös mitä testitapausta luodessa tulee huomioida, miten testitapauksia ajetaan sekä miten tulee toimia, kun testaamisen aikana havaitaan jokin virhe eli bugi ohjelmistossa.

Testaus on tärkeä osa ohjelmistoprojektia, joten testaus on suunniteltava huolellisesti sekä mahdollisimman tarkasti. Hyvän testitapauksen ominaisuuksiin kuuluu, että testauksella löytää useita virheitä, testaaminen ei ole monimutkaista ja testaukseen ei kuluisi liian pitkä aika. Suunnitelmaa tulee tarvittaessa muuttaa projektin edetessä. Suunnitelmaa tehdessä on otettava huomioon ajankäyttö, tavoitteet mitä testauksella tulisi saavuttaa, resurssit sekä organisointi. Suunnitelmassa tulee myös mainita, miten testaaminen päätetään.

2.1 Testauksen osat

Testaus voidaan jakaa useisiin osiin. Yksi tapa on jakaa testaus eri vaiheisiin V-mallin mukaan. Tässä kuva 1 V – mallista.



Kuva 1. V-malli

V-mallissa testausvaiheet on liitetty yhteen tuotteen suunnitteluvaiheen kanssa. Testauksen tulokset tarkastellaan vertaamalla niitä vastaavan suunnittelutason vaatimusdokumentteihin. V-malli on suosittu tapa mallintaa testauksen eteneminen, tosin muutama ongelma löytyy V-mallissa. Sitoutuminen ohjelmistokehityksen vesiputousmalliin on luultavasti suurin ongelmista. Vesiputousmallissa olevat ongelmat siirtyvät suoraan V-mallin mukaisesti suoritettuun testaukseen.

Testaus jaetaan myös turhan tiukasti eri tasoihin V–mallissa. Ajoittain olisi järkevää suorittaa moduuli- ja integrointitestaus lomittain tai yhdistää nämä, mutta V–malli ei rohkaise suunnittelutasojen yhdistämiseen. (Paakki 2000.)

2.2 Testauksen vaiheet

V-mallissa testaus on jaettu seuraaviin tasoihin:

Moduulitestaus – Moduulitestauksessa, joka tunnetaan paremmin nimellä yksikkötestaus keskitytään yksittäisen ohjelman osaan tai komponenttiin, joka suorittaa yksittäisen funktion. Yksikkötestauksen tarkoituksena on varmistaa, että ohjelman osat toimivat itsenäisesti määriteltyjen vaatimusten mukaisesti. Moduulitestauksen tavoitteena on varmistaa, että juuri toteutettu toiminto ja muutos toimii ainakin periaatteessa. Moduulitestauksen suorittaa yleensä ohjelmoija. (Kasurinen 2013, 51; Lahtinen 2016.)

Integrointitestaus – Integrointitestauksessa yhdistetään moduuleita osajärjestelmiksi ja tärkein tavoite integrointitestauksella on testata, että osat toimivat yhdessä. Testauksen painopiste on moduulien välisten rajapintojen toimivuuden tutkimisessa ja testauksen tuloksia voidaan verrata teknisiin määrittelyihin. Integrointitestauksella on myös tarkoitus havaita virheitä, jotka tapahtuvat yksittäisten komponenttien rajapinnoilla ja varmistaa, että yksiköt toimivat osajärjestelminä ja lopulta kokonaisena järjestelmänä. (Kasurinen 2013, 54; Mäkelä 2000.)

Järjestelmätestaus – Järjestelmätestauksella testataan järjestelmää kokonaisuutena, että tämä toimii vaatimusten mukaisesti. Testaus kestää yleensä pitkän ajan ja vaatii enemmän resursseja kuin muut testauksen vaiheet. Järjestelmätestaus voidaan myös jakaa eri tyyppeihin, kuten esimerkiksi funktionaaliseen testaukseen, suorituskyky- ja luotettavuustestaukseen. Järjestelmätestauksessa tärkeää on, että testaajana toimii kehitystyöstä riippumaton henkilö tai useampi henkilö. (Mäkelä 2000.)

Järjestelmätestaus suoritetaan testiympäristössä, ei varsinaisessa lopullisessa kohdeympäristössä. Tärkein ero hyväksymistestauksen ja järjestelmätestauksen välillä onkin siinä, että järjestelmätestauksen vaiheessa virheitä edelleen etsitään myös yksittäisistä komponenteista, eivätkä muutokset toteutettavaan järjestelmään ole epätavallisia. Painopiste siirtyy toiminnallisuuksien todentamiseen, eikä järjestelmälle tehdä enää merkittäviä muutoksia. (Kasurinen 2013, 56.)

Hyväksymistestaus – Hyväksymistestauksen suorittavat ohjelmiston asiakkaat ja käyttäjät, jotka työskentelevät ohjelmiston parissa. Tarkoitus on varmistaa ohjelmiston vaatimusten täyttyminen. Normaalisti tämä testaus tarkoittaa sitä, että asiakas hyväksyy tuotteen valmistumisen ja onnistuneeksi todetun hyväksymistestauksen jälkeen ohjelmisto siirtyy asiakkaan omaisuudeksi. Hyväksymistestaus voidaan jakaa kahteen eri osaan, alfa-testaukseen ja beta–testaukseen. Alfa–testauksen suorittavat todelliset käyttäjät kehitystyön suorittaneessa yrityksessä. Beta–testauksen suorittavat asiakkaat omassa käyttöympäristössään. (Kasurinen 2013, 57; Mäkelä 2000.)

Regressiotestaus – Regressiotestauksella tarkoitetaan aiemmin testatun ohjelman uudelleentestaamista, pyrkimyksenä varmistaa, että ohjelmaan tehty muutos on oikeellinen eikä ole aiheuttanut virheitä ohjelmassa ennen muutosta olleeseen toiminnallisuuteen. Ohjelmaan tehty muutos voi olla koodin muuttamista, lisäämistä tai poistamista. Toimintaympäristöllä tarkoitetaan ohjelmaan yhteydessä olevia ulkopuolisia ohjelmia tai järjestelmiä, kuten myös laitteistoa ja käyttöjärjestelmää, joilla ohjelmaa suoritetaan. Regressiotestauksen voi suorittaa missä vaiheessa prosessia tahansa. Yleensä regressiotestaus suoritetaan, kun ohjelmisto uudelleen testataan korjauksen tai muiden muutoksien jälkeen testauksen viimeisenä vaiheena. (Holopainen 2005.)

2.3 Testauksen tavoitteet

Testaajalta vaaditaan tarkkaavaisuutta testausprosessin aikana, jotta havaittuja virheitä ei käyttöönottovaiheessa enää ohjelmistossa olisi. Onnistuneessa testitapauksessa tulisi testata kaikki mahdollinen ohjelmistosta eri arvoilla. Testaaminen tulee myös suorittaa tarkoituksellisesti väärin esimerkiksi antamalla jokin arvo väärin. Testitapausta luodessa tulee huomioida käyttäjän näkökulma, mikä käyttäjän mielestä voisi olla häiritsevää tai mistä käyttäjät eivät pitäisi ollenkaan. Tässä muutama esimerkki tavoitteista joita testaamisella on:

- Löytää viimeiset virheet ennen ohjelmiston tuotantoon vientiä
- Suorittaa testaus mahdollisimman läheltä käyttäjän kokemusta
- Testata jokaisen muuttujan jokainen mahdollinen syöte
- Testata jokainen mahdollinen yhdistelmä syötteitä ja muuttujia
- Testata jokainen mahdollinen toimintoketju ohjelmiston läpi
- Testata jokainen laitteisto/ohjelmistokonfiguraatio, mukaan lukien palvelinkonfiguraatiot, jotka eivät ole hallinnassasi
- Testata jokainen tapa, jolla käyttäjä saattaisi yrittää käyttää sovellusta

- Suorittaa testaus myös virheellisillä arvoilla tai tiedoilla, jotta näkee, miten ohjelmisto reagoi virheisiin

(Pyhäjärvi 2006.)

2.4 Virheiden havaitseminen

Virheiden etsinnällä parannetaan ohjelmiston laatua ja toimivuutta. Nopeasti havaitut virheet voivat säästää paljon vaivaa sekä rahaa, koska julkaisun jälkeen havaittujen virheiden korjaaminen voi tulla erittäin kalliiksi. (Usenius 2009.)

Havaitusta virheestä tulee luoda bugi. Bugi luodaan, jotta pystytään selittämään mahdollisimman tarkasti, mikä meni testissä vikaan. Bugia luodessa tulee selittää vika mahdollisimman tarkasti, koska tämä helpottaa ohjelmoinnissa virheiden korjaamista. Bugin luomisen jälkeen tieto lähetetään ohjelmoijalle, joka tarkistaa koodista, mikä meni vikaan ja miten tämä korjataan. Korjauksen jälkeen testitapaus ajetaan uudestaan regressiotestitapauksena. Se, miten bugi ilmenee ohjelmistossa, tulee selittää mahdollisimman tarkasti, jotta ohjelmistokehittäjällä on mahdollista löytää kyseinen bugi helposti. Tähän auttaa bugin toteaminen myös steppien avulla. Esimerkiksi ensin steppi missä sivulla vika ilmenee, mitä tietoja sivulle annettiin sekä miten lopullinen bugi ilmenee ohjelmistossa.

3 Manuaalinen testaus versus automaatiotestaus

Tämä on opinnäytetyön testauksen teorian osuus, jossa kerrotaan manuaalisesta sekä automaatiotestauksesta. Osiossa selitetään näiden tarkoitukset sekä käydään läpi manuaalisen ja automaatiotestauksen hyviä ja huonoja puolia. Luvussa käydään läpi automaatiotestauksen eri menetelmiä: millä eri sovelluksilla testaamista voi suorittaa. Testaussovellusten perusominaisuuksia esitellään ja lopuksi näytetään, miltä sovellus näyttää asentamisen jälkeen. Lisäksi kerrotaan, miten testitapauksien luominen testisovelluksilla onnistuu. Testitapauksien luominen sovelluksilla sekä testitapauksien suorittaminen tulevat esille opinnäytetyössä kappaleessa 5. Testitapaukset.

3.1 Manuaalinen testaus

Manuaaliset testitapaukset eivät aina ole monimutkaisia ja testattavana voivat olla hyvinkin yksinkertaiset asiat. Ajoittain testitapauksia voi joutua ajamaan useaan kertaan, tämän takia kannattaa harkita käyttämään automaatiotestausta. Manuaalinen testaus on hyvää harjoittelua aloitteleville testaajille, jotta he ymmärtävät paremmin, mistä testaamisesta on kyse ja mitä sillä tavoitellaan.

Manuaalisessa testaamisessa IT–taitojen ei välttämättä tarvitse olla kovin kummoisia. Monet jotka työskentelevät tietokoneiden parissa tai osaavat käyttää tietokonetta voivat suorittaa manuaalista testausta. Manuaali testaukseen ei myöskään välttämättä tarvitse käyttää erikoisempia sovelluksia vaan testitapauksia voi luoda vaikka Microsoft Exceliin. (Testauksen osaamisyhteisö 2012.)

Tässä mainittuja manuaali testauksen hyviä ja huonoja puolia:

+ Virheitä voi löytää paremmin manuaali testauksella
+ Parhaimmillaan manuaalinen testaus on tutkivaa testaamista
+ Manuaalista testausta tarvitaan jokaisen osa-alueen testaamiseen ainakin kerran
+ Aloittava testaaja oppii hyvin testaamisen tarkoituksen ja perusteet manuaali testauksella

- Manuaalinen testaus voi kestää liian kauan

- Manuaalinen testaus voi olla liian helppoa
- Testitapauksen moneen kertaan suorittaminen voi käydä tylsäksi
(Testauksen osaamisyhteisö 2012.)

3.2 Automaatiotestaus

Testaaminen sisältää työläitä toimenpiteitä, joten automatisoidulla testaamisella on mahdollista käyttää henkilöresurssit järkevämmin. Tämän avulla voidaan pohtia testausta syvällisemmin järjestelmän sekä käyttäjän näkökulmasta, eikä ole tarvetta laittaa henkilöitä käymään testitapauksia robottimaisesti läpi. (Berglund 2015.)

Tässä lista automaatiotestauksen hyvistä ja huonoista puolista:

+ Kustannukset pienenevät
+ Henkilöstöresurssien tarve vähenee
+ Testaus voidaan suorittaa milloin vain ajastamalla testaus (esim. keskellä yötä)
+ Käyttöliittymän saa testattua tarkemmin
+ Toistettavuus, testauksen voi suorittaa kuinka monta kertaa tahansa
+ Nopeus, testin saa ajettua nopeammin kuin manuaalisesti
+ Monikielisen sivuston testaus on manuaalisesti vaikeaa, automatisoidulla paljon helpompaa

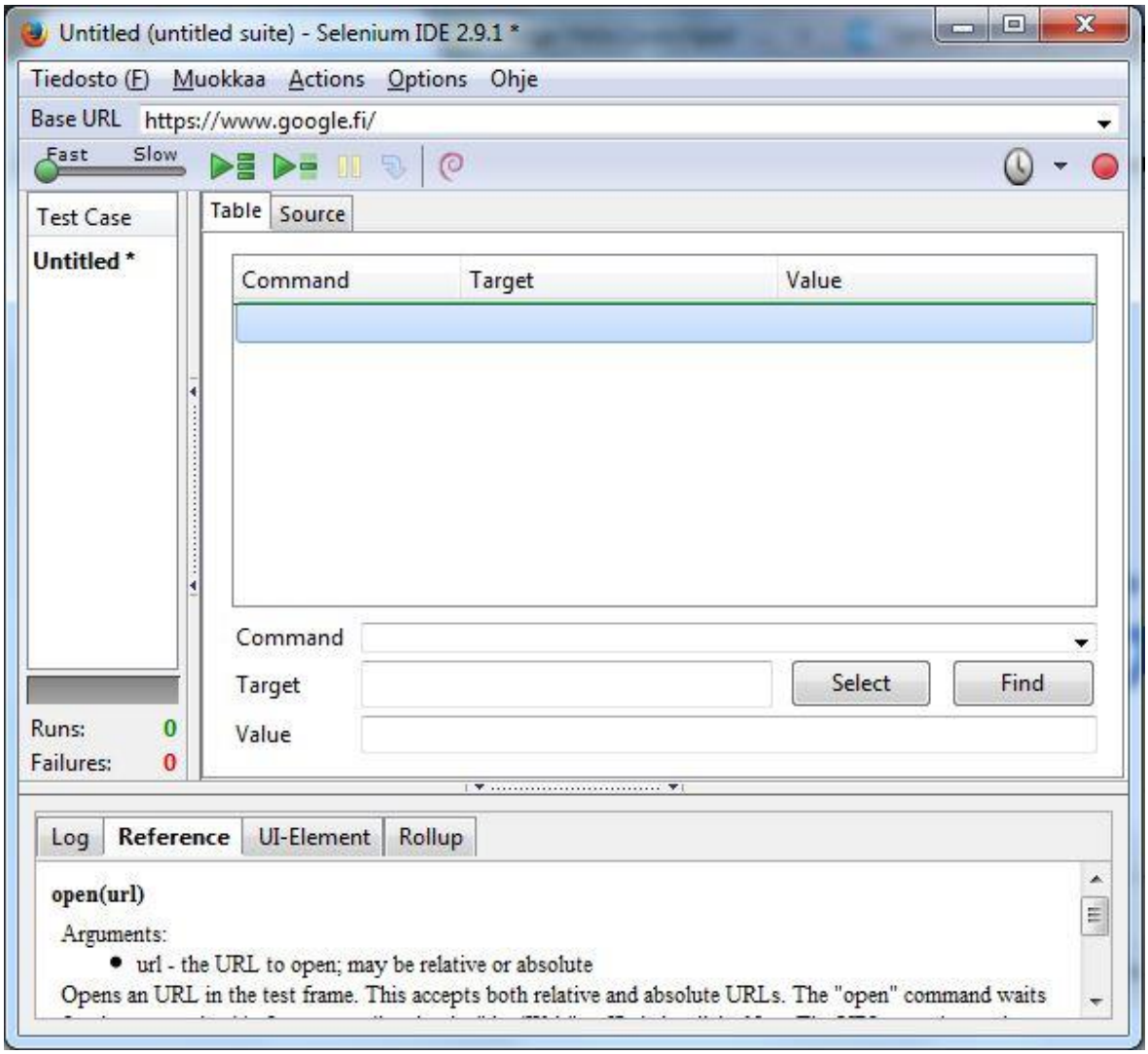
- Mitä enemmän testattavaa, sitä suuremmalla todennäköisyydellä testikierrokset eivät ole täysin identiteettisiä muiden kanssa
- Jos testaajia on useita, tulee käyttöliittymä testattua eri tavalla joka kerta
- Testitapaukset joiden vaatimukset muuttuvat usein eivät ole soveltuvia automatisoituun testaamiseen
(Pohjolainen 2003.)

3.3 Automaatiotestauksen sovellukset

Tässä osiossa käydään läpi tunnetuimpia sovelluksia, joita automaatiotestauksessa on ja samoilla sovelluksilla suoritetaan automaatiotestausta opinnäytetyössä kappaleessa 5. Sovellukset ovat Selenium, Robot Framework (jota toteutetaan Pythonilla) sekä Jenkins. Itse testitapaukset ovat normaaleja tapauksia työelämässä ja testaustilanteen voidaan kuvitella olevan regressiotestausta vaille valmis. Regressiotestauksen jälkeen sivusto voidaan siirtää tuotantoon.

3.3.1 Selenium

Automaatiotestaus voidaan aloittaa Selenium IDE:llä. Tätä työkalua käytetään yleisesti Firefox selaimella eri sivustojen toiminnan ja ominaisuuksien testaamiseen. Lataus onnistuu Firefoxin Addons-sivun kautta, jossa onnistuu Firefoxin eri lisäosien lataamiset. Selenium IDE ei toimi uusimpien Firefox-versioiden kanssa, joten Seleniumin testaamiseen on käytetty vanhempaa Firefox 54 versiota. Tässä kuvassa 2 näkyy, miltä Selenium IDE näyttää asennettuna.



Kuva 2. Selenium IDE asennettuna

Seleniumin yläosassa olevaan Base URL-kohtaan tulee antaa verkkosivu, jota on tarkoitus testata. Seleniumilla testaaminen onnistuu kahdella tavalla, antamalla komentoja Command-kohtaan, joilla käsketään mitä Seleniumin tulee tehdä sekä Source-välilehdellä, joka on enemmän ohjelmoinnintapaista testitapauksen luomista. Source-välilehteä ei ole tässä osiossa käytetty, koska komentojen käyttäminen Seleniumin kanssa on kätevämpää.

Testitapauksissa komennot tulee antaa kuvassa näkyvään Command-riville. Komennon viereen Target-kohtaan annetaan tarkempi tieto, mitä komennon tulee tehdä. Esimerkiksi, jos käytetään yksinkertaista klikkaus komentoa Click, annetaan komentoriville sekä Target-riville tieto, mistä tietystä kohdasta tulee sivulla klikata. Jos sivustolla halutaan klikata jotakin tiettyä linkkiä, tulee tämä antaa Target-kohtaan muotoon link=Linkki. Jos halutaan testitapauksen painavan jotain painiketta, tulee Click-komennolle Target-kohtaan antaa //div/button lause.

Value-kohtaa käytetään testitapauksissa ainoastaan Type-komennoilla, joilla määriteltiin, mikä teksti tulee sivulle antaa.

Tässä lista komennoista, joita käytettiin Selenium-testaamiseen:

Click – Klikkaa määritellystä kohdasta

ClickAndWait – Klikkaa annetusta kohdasta ja odottaa sivun lataamista

Open – Avaa selaimen

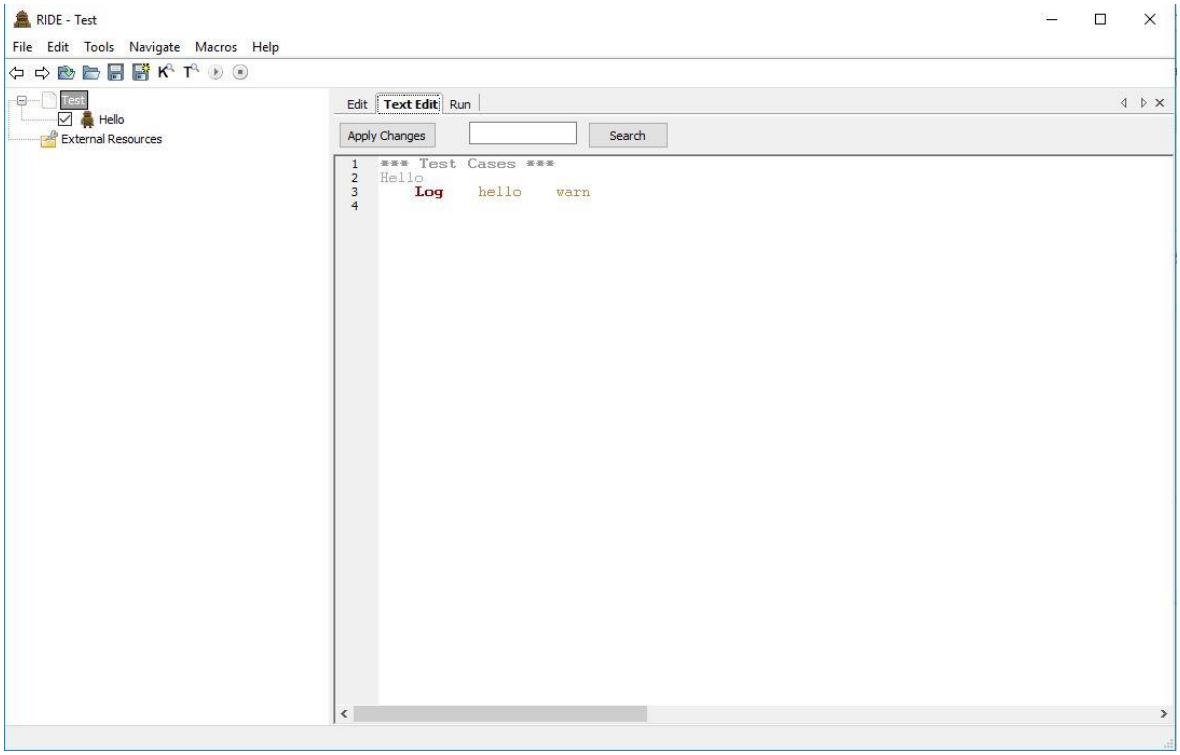
Type – Kirjoittaa käsketyn tekstin sille määritettyyn kohtaan

Kuvitellaan, että testitapauksessa testataan Googlen hakukenttää. Base URLiin annetaan osoitteeksi https://www.google.fi/. Command-kohtaan tulee antaa Seleniumiin komento Type, jolla Selenium ymmärtää kirjoittaa tekstiä. Target-kohtaan lause id-lst-ib, jolla määritellään, mihin kohtaan teksti tulee sekä Value-kohtaan vielä haluttu teksti, jota Googlessa haetaan. Lopuksi tulisi vielä antaa Click-komento, jolla Selenium painaa Google-haku painiketta ja Google suorittaa haun.

3.3.2 Robot Framework

Robot Framework testaukseen käytetään Python-kieltä. Pythonista käytetään vanhempaa versiota 2.7.8. Tämä syystä, että uusimmissa versioissa on ongelmia toimivuuden kanssa.

Robot Framework asennuksen jälkeen.



Kuva 3. Robot Framework testauksen sovellus Ride asennettuna.

Testaaminen Ridellä onnistuu hyvin samalla tavalla kuin Seleniumilla. Antamalla komentoja sekä antamalla tarkemmat tiedot komennolle, mihin kohtaan sivustolla annettu komento tulee suorittaa. Seleniumista poiketen, Robot Frameworkilla käytetään XPath-lauseita. Tässä esimerkki XPath-lauseesta: xpath=//input[@class='gsfi lst-d-f']. Tällä lauseella sijoitetaan sivustolla jokin teksti haluttuun kohtaan.

Esimerkiksi kirjoittaessa tekstiä sille tarkoitettuun kenttään, tulee käyttää Input Text -komentoa ja tämän perään XPath-lause, jolla määritellään, mihin kohtaan teksti tulee. Twitter -sivuston testaamisessa (tulee esille myöhemmin opinnäytetyössä) käyttäjätunnuksen antamiseen tulee antaa komento Input Text ja XPath-lause xpath=//input[@class='text-input email-input'].

Tässä lista Robot Frameworkilla käytetyistä komennoista:

Open browser – Aukaisee valitun selaimen

Input Text – Syöttää tekstin tekstikenttään

Sleep – Selain odottaa sivuston latautumista käsketyn ajan verran

Click Button – Painaa valittua painiketta

Click Element – Painaa valittua ominaisuutta

Click Image – Painaa sivuston kuvaa tai tiettyä kuvan osaa

Close Browser – Sulkee selaimen

Maximize browser window – Suurentaa selaimen ikkunan

Testitapauksissa on käytetty myös muuttujia, joilla annettiin käyttäjätunnus ja salasana. Nämä tulee luoda Robot Frameworkiin Text Edit välilehteen. Muuttujat annetaan muodossa ${Käyttäjätunnus} ja ${Salasana}. Nämä kirjoitetaan sivustolla oikeaan kohtaan Input Text -komennolla ja XPath-lauseessa annetun komennon mukaisesti. Muuttujien luonti on yksinkertaista. Luodaan muuttuja ${käyttäjätunnus} ja annetaan sulkujen sisälle kyseinen käyttäjätunnus. Salasana muuttujan luominen onnistuu samalla tavalla.

Tässä kuva 4 kyseisistä muuttujista, jotka auttavat hahmottamaan näiden sisältöä.

Kuva 4. Muuttujat

3.3.3 Jenkins

Jenkins on avoimen lähdekoodin automaatiopalvelin. Jenkins on ohjelmoitu Javalla ja Jenkins toimii Windowsilla, Mac OS X:llä ja UNIX-

käyttöjärjestelmillä. Jenkinsin avulla yritykset voivat nopeuttaa ohjelmistokehityksen prosessia automaation kautta. Jenkins hallitsee ja kontrolloi ohjelmistokehityksen elinkaaren kaikenlaisia prosesseja, kuten buildit, dokumentointi, testit, paketit, deployt ja paljon muuta. Jenkins voidaan ohjelmoida tarkistamaan, onko versionhallintaan, kuten SVN tai Git, tullut muutoksia, ajamaan testit ja sen jälkeen ryhtyä toimenpiteisiin: palauttamaan ohjelmiston entiselleen, mikäli testit epäonnistuvat (rollback) tai jatkaa eteenpäin, mikäli testit menivät läpi. (Cloudbees 2016.)

Aikaisemmista sovelluksista poiketen Jenkinsillä ei anneta komentoja testitapauksiin, vaan testitapaukset suoritetaan scriptien avulla. Jenkinsiä voi myös käyttää yhdessä Githubin kanssa, jonne voi luoda testitapauksen oman lähdekoodin ja ajaa tämä Jenkinsin avulla. Tässä opinnäytetyössä testitapauksissa on käytetty yleisessä käytössä olevia Github-projekteja Cucumberbasic sekä Selenium With Cucumber. Nämä löytyvät sivuilta https://github.com/executeautomation/cucumberbasic sekä https://github.com/executeautomation/SeleniumWithCucucumber

Tässä kuva 5. Jenkinsistä asennettuna.

Kuva 5. Jenkins asennettuna

4 Testausmenetelmät

Tässä osiossa käydään läpi tunnetuimpia testausmenetelmiä. Testausmenetelmiä on kolme kappaletta. Nämä ovat Black–box testaus, White–box testaus sekä Gray–box testaus. Aluksi käydään läpi näiden teoriaa, mitä testausmenetelmällä haetaan, mitä menetelmä pitää sisällään sekä miten testausmenetelmää suoritetaan. Menetelmiä vertaillaan keskenään sekä käydään läpi näiden eroja.

4.1 Black-box testaus

Black-box testauksessa tutustutaan enimmäkseen sovelluksen toiminnallisuuteen, eikä koodin rakenteeseen tai muun sovelluksen sisältämiin rakenteisiin. Koodin tuntemus ei siis ole välttämätöntä Black-box testaamisessa. Keskittyminen on enimmäkseen vaatimuksissa ja määrittelyissä. Black-box testaus on siitä erittäin kätevä, että tätä voidaan käyttää todella laajasti. Tällä menetelmällä voidaan sovelluksien lisäksi testata käyttöjärjestelmiä (kuten Windows ja Linux), verkkosivuja ja tietokantojen toimintaa.

Black–box testausmenetelmistä kaksi tunnetuinta ovat UFT sekä HP Loadrunner. UFT (Unified Functional Testing) tunnettiin aikaisemmin nimellä QTP (QuickTest Professional). UFT on HP:n omistama ja nykyään tuottama, automatisoitu testausmenetelmä, jota käytetään monissa yrityksissä varsinkin laadun varmistukseen. Ennen HP:n omistusta UFT:n edeltäjä QTP:n luojana toimi Mercury Interactive Corporation. Kyseessä on Windows-pohjainen menetelmä, jolla pystytään testaamaan toisia Windows-pohjaisia sovelluksia. Käytännössä UFT:llä luodaan, hallinnoidaan sekä suoritetaan testiskriptejä. Tämä käyttää skriptikielenä VBScrpit-kieltä.

HP Loadrunner on nimensä mukaisesti myös HP:n omistama testausmenetelmä. Suurin ero Loadrunnerissa verrattuna UFT:hen on se, että Loadrunnerilla pystyy simuloimaan tuhansien käyttäjien määrä ohjelmistossa. Toisin sanoen testaus suoritetaan samalla tavalla kuin sovellusta käyttäisi tuhannet käyttäjät samaan aikaan. Loadrunner käyttää enimmäkseen skriptikielenä ANSI C kieltä, mutta Loadrunnerissa on myös mahdollista käyttää Javaa ja .NET kieltä. Elokuussa 2015 julkaistussa 12.05 versiossa kieliksi lisättiin vielä JavaScript.

Testausmenetelmiä Black–box testauksella on monia. Yleisimmät näistä ovat toiminnallinen testaus, ei–toiminnallinen testaus ja regressiotestaus. Toiminnallisen testauksen suorittaa testaaja. Tämä testausmenetelmä kuuluu toiminnallisiin vaatimuksiin, joita järjestelmässä löytyy. Tärkeintä toiminnallisella testauksella on testata sovelluksen toimivuus, jotta tämä vastaa vaadittuja ominaisuuksia sekä toiminnallisuuksia.

Ei–toiminnallisessa testausmenetelmässä testaus ei taas ole sidonnainen testaukseen tai tiettyyn toimivuuteen. Ei–toiminnallisessa testauksessa keskitytään enimmäkseen muihin sovelluksen vaatimuksiin, kuten suorituskykyyn ja käytettävyyteen. (Guru99 2013.)

4.2 White-box testaus

White-box testaus poikkeaa Black-box testaamisesta siten, että menetelmässä käydään läpi myös koodia. Kun Black–box testauksella keskitytään enemmän sovelluksen toimintaan loppukäyttäjän näkökulmasta, niin White–box testauksella testataan enemmän ohjelmoijan näkökulmasta. Tässä voidaan myös keskittyä vain yhteen ominaisuuteen tai osaan sovelluksessa. (Guru99 2013.)

Testaus voidaan jakaa kolmeen osaan: yksikkötestaus, integraatiotestaus sekä regressiotestaus. Yksikkötestauksella varmistetaan, että koodi toimii odotetusti ennen kuin yhdistymistä tapahtuu aikaisempaan jo kirjoitettuun koodiin. Tarkoituksena on löytää koodista vikoja jo alkuvaiheessa, jotta yhdistyminen onnistuu ongelmitta ja, että myöhemmin ohjelmoinnissa tulisi mahdollisimman vähän uusia virheitä.

Integraatiotestaus suoritetaan yksikkötestauksen jälkeen, kun koodi on testattu niin, että tämä toimii määrittelyiden mukaan. Integraatiotestauksella testataan vaikutukset, joita yhdistyminen on aiheuttanut sovellukselle. Tärkeintä on testata sovelluksen toimivuus ja käyttäytyminen. (Software testing fundamentals 2017.)

Regressiotestauksessa käytetään samoja White–box testitapauksia, joita käytettiin yksikkö- ja integraatiotestauksessa. Tässä vaiheessa testausta on tärkeää löytää viimeisetkin bugit, joita ei havaittu aikaisemmissa vaiheissa.

White–box testauksessa on siis tärkeää ymmärtää lähdekoodia. Myös miten sovelluksen tulisi toimia, tulee testaajan tietää. Tähän voi auttaa suunnittelijoiden kanssa sovelluksen ominaisuuksien kattava läpikäynti sekä hyvä ja laaja testitapauksen luominen. Testitapausta luodessa tulee ottaa huomioon valmisteluvaiheessa kaikki oleelliset perusominaisuudet, kuten tekniset tiedot, yksityiskohtainen suunnittelu ja toimiva lähdekoodi.

Suunnittelussa tulee myös huomioida riskien analysointia. Tämä tulee ottaa huomioon koko testauksen aikana niin testauksen suunnittelussa, testitapauksia luodessa, testauksia ajaessa sekä lopputulosten läpikäynnin aikana. Aivan lopuksi tulee vielä tehdä raportti, miten testitapaus onnistui. Oliko virheitä, miten nämä korjattiin sekä läpäisikö sovellus testauksen. (Guru99 2013.)

4.3 Grey-box testaus

Grey–box testaus on näistä kolmesta vähiten käytetty testausmenetelmä. Kyseessä on yhdistelmä White ja Black–box testaamista. Tämän testauksen tavoitteena on löytää virheitä koodin rakenteessa sekä sovelluksen väärin käytössä. Testausmenetelmä on kätevä varsinkin verkkosovelluksien testaamiseen.

Esimerkiksi Black–box testaaja ei tunne sisäistä rakennetta ja koodia, kun taas White–box testaaja tuntee sovelluksen koodin sekä rakenteen, miten sovellus on rakennettu. Grey–box testaaja eroaa tästä siten, että testaaja tuntee sisäisen rakenteen, johon kuuluvat pääsy tarvittaviin dokumentteihin, jotka sisältävät tietoa rakenteesta, miten sovelluksen tulisi koostua. Grey-box testaajat tarvitsevat siis yksityiskohtaisia asiakirjoja, jotka sisältävät kuvauksen sovelluksesta, jota on kerätty testitapauksen määrittelemiseksi. (Madsen. C & T. 2016.)

5 Case Study: Testitapaukset

Osioon on luotu esimerkkitestitapaukset. Näissä esimerkeissä käydään läpi regressiotestitapauksia, joilla testataan sivuston yleistä toimintaa sekä sivujen tärkeimpien ominaisuuksien toimintaa. Testitapauksissa voidaan kuvitella tilanne suoraan työelämästä. Ohjelmointi on saatu valmiiksi sekä aikaisemmat testauksen vaiheet on ajettu onnistuneesti ja virheet on sivuilta korjattu. Viimeisenä vaiheena on regressiotestauksen aika ennen sivuston tuotantoon vientiä. Kaksi ensimmäistä testitapausta ovat manuaalisia testitapauksia ja loput viisi kappaletta ovat automaatiotestauksen testitapauksia. Tämän osion testitapaukset on suoritettu Black–box testausmenetelmillä. Jenkinsin käytössä voidaan puhua lievästä White–box testauksesta, koska käytössä on lähdekoodi testaamisen suorittamiseen, mutta testauksella ei puututa sivuston ohjelmointiin.

5.1 Manuaaliset testitapaukset

Testitapaus 1. www.katsomo.fi

Testitapauksessa 1 testataan ensin www.katsomo.fi sivulle sisäänpääsyä, kirjaudutaan sisälle käyttäjätunnuksella ja salasanalla, testataan hakukentän toimintaa, sekä testataan, että haluttu ohjelma lähtee pyörimään. Taulukossa 1 näkyy testitapauksen stepit, steppien odotettu lopputulos sekä tieto, onnistuiko steppi.

Step	Odotettu lopputulos	Passed/Failed
Mene selaimella osoitteeseen www.katsomo.fi	Selaimella siirrytään osoitteeseen www.katsomo.fi	Passed
Valitse oikeasta yläkulmasta kirjaudu painike	Siirrytään ”Kirjaudu sisään ja näe kaikki ilmaiseksi sivulle”	Passed
Anna käyttäjätunnus ja salasana	Sähköposti tulee olla muotoa esimerkki@gmail.com. Salasanan tulee olla vähintään 8 kirjainta (A-z) ja salasana voi sisältää numeroita (0-9)	Passed
Valitse Kirjaudu painike	Kirjaudutaan onnistuneesti sisälle	Passed
Valitse Haku painike	Sivu siirtyy "Hakusana" sivulle	Passed
Kirjoita hakukenttään Salatut Elämät	Hakusivulle ilmestyy uusin jakso	Passed
Painetaan uusinta jaksoa	Sivu aloittaa näyttämään valittua jaksoa	Passed

Taulukko 1. Katsomo testitapaus

Jokainen steppi on testitapauksessa merkattu onnistuneesti Passed tilaan. Testi on ajettu onnistuneesti ja sivuston voi siirtää tuotantoon.

Testitapaus 2. Verkkokauppa.com tilauksen suorittaminen.

Testitapauksessa 2 testataan ensin, että Verkkokauppa.com sivusto tulee esille, sisäänkirjautuminen onnistuu, tuote sivun toiminta, tuotteen lisääminen ostoskoriin, tilauksen loppuosan suorittaminen sekä uloskirjautuminen. Taulukossa 2 esitetään testitapauksen 2 stepit.

Step	Odotettu lopputulos	Passed/Failed
Kirjaudu sisään valitsemalla Kirjaudu painike	Siirryt kirjaudu sisään sivulle	Passed
Anna sähköpostiosoite ja salasana. Valitse Kirjaudu sisään	Sisäänkirjautuminen onnistuu	Passed
Valitse vasemmalla oleva Komponentit painike	Näytä tuotealue tulee esille	Passed
Valitse tuotealueesta Prosessorit	Prosessorien valmistajat tulevat esille	Passed
Valitse tuotealue	Tuotealue tulee esille	Passed
Lisää tuote ostoskoriin	Tuote siirtyy ostoskoriin	Passed
Siirry Kassalle	Siirryt Ostoskori sivulle	Passed
Ostoskori sivulla valitse Seuraava	Siirryt Yhteystietosi sivulle	Passed
Tarkista yhteystiedot ja paina seuraava painiketta	Siirryt toimitustapa sivulle	Passed
Valitse Nouto myymälästä kohdasta Nouto Helsingistä tai 24h kioskista. Siirry eteenpäin valitsemalla seuraava	Siirryt valitse maksutapa sivulle	Passed
Valitse maksutavaksi Maksu kassalle noudettaessa. Valitse seuraava	Siirryt yhteenveto sivulle	Passed
Yhteenveto sivulla tarkista tilauksen tiedot oikeiksi. Valitse Vahvista tilaus	Tilauksen lähetys onnistuneesti viesti tulee esille	Failed

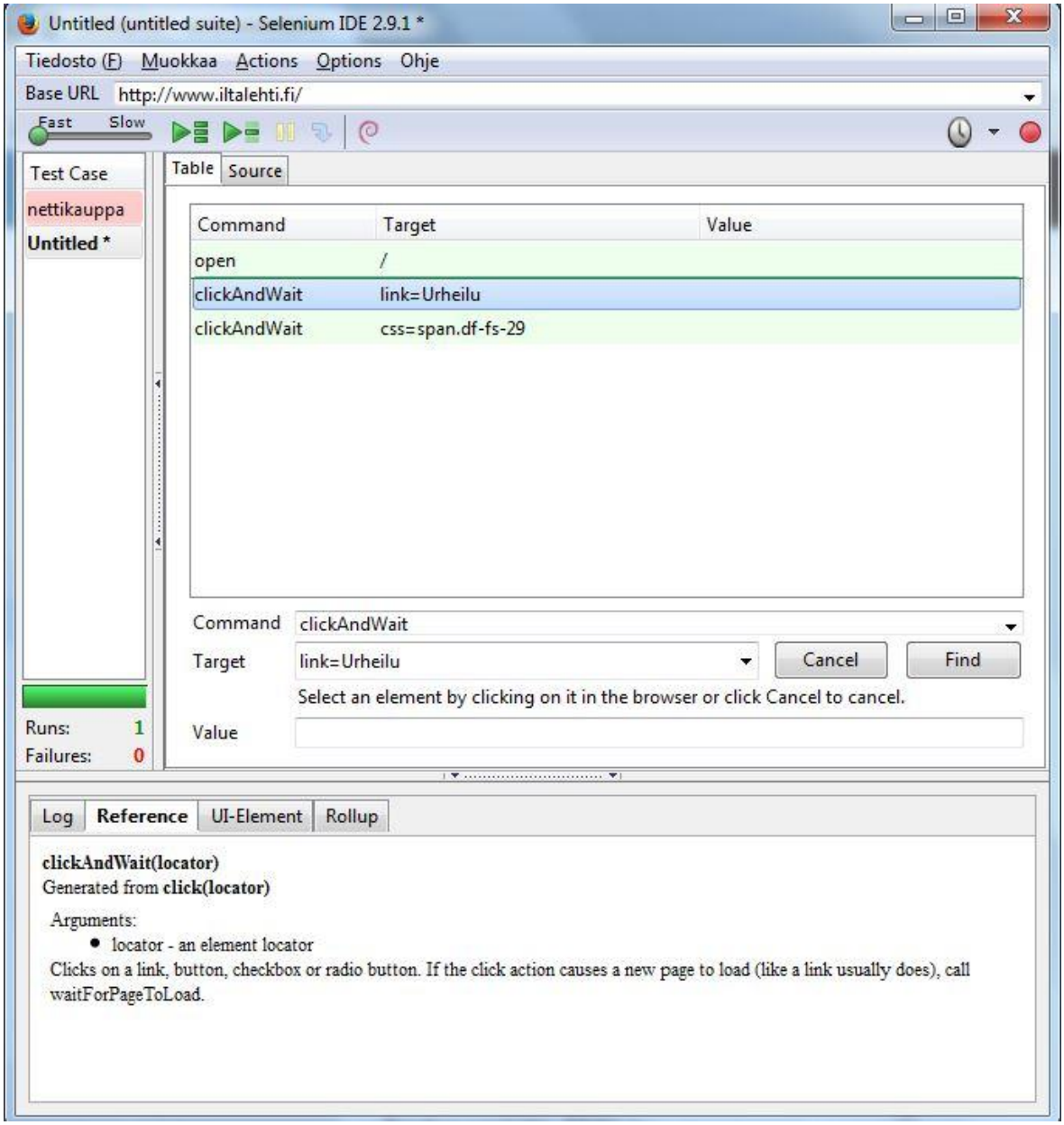
Taulukko 2. Manuaalinen testi Verkkokauppa.com sivulla

Testitapauksessa merkittiin viimeinen steppi Failed-tilaan eli tilauksen vahvistaminen ei onnistunut. Virhe estäisi sivun tuotantoon viennin ja tästä tulisi luoda bugi, jonka ohjelmoijan olisi korjattava ennen sivun tuotantoon vientiä.

5.2 Selenium testaus

Automaatiotestaus aloitetaan Seleniumilla. Ensimmäisessä testitapauksessa testataan yksinkertaista tapausta, jossa mennään Iltalehden etusivulle, siirrytään urheilu-uutisiin ja klikataan ensimmäistä urheilu-uutista. Ensimmäisessä testitapauksessa sivun osoite on http://www.iltalehti.fi/.

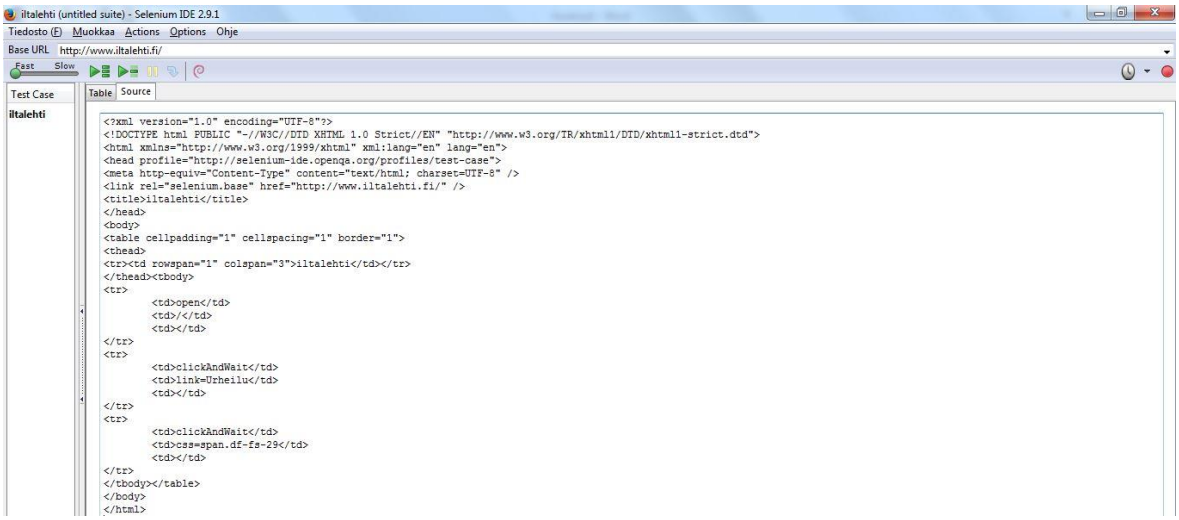
Testitapaus 3. Iltalehti



Kuva 6. www.iltalehti.fi

Aluksi testitapauksessa oli ongelmia Iltalehden sivun lataamisen kanssa. Ongelma oli, että Selenium ei antanut sivun latautua tarpeeksi kauan. Selenium oli jo klikkaamassa urheilusivua, ennen kuin sivu oli edes ehtinyt latautua loppuun. Tähän auttoi, kun vaihtoi Click-komennon ClickAndWait-komentoon. Tällä komennolla saatiin kyseinen testitapaus ajettua onnistuneesti.

Tässä kuvassa 7 näkyy Source-välilehdessä itse koodia, miten myös testitapauksen olisi voinut suorittaa.

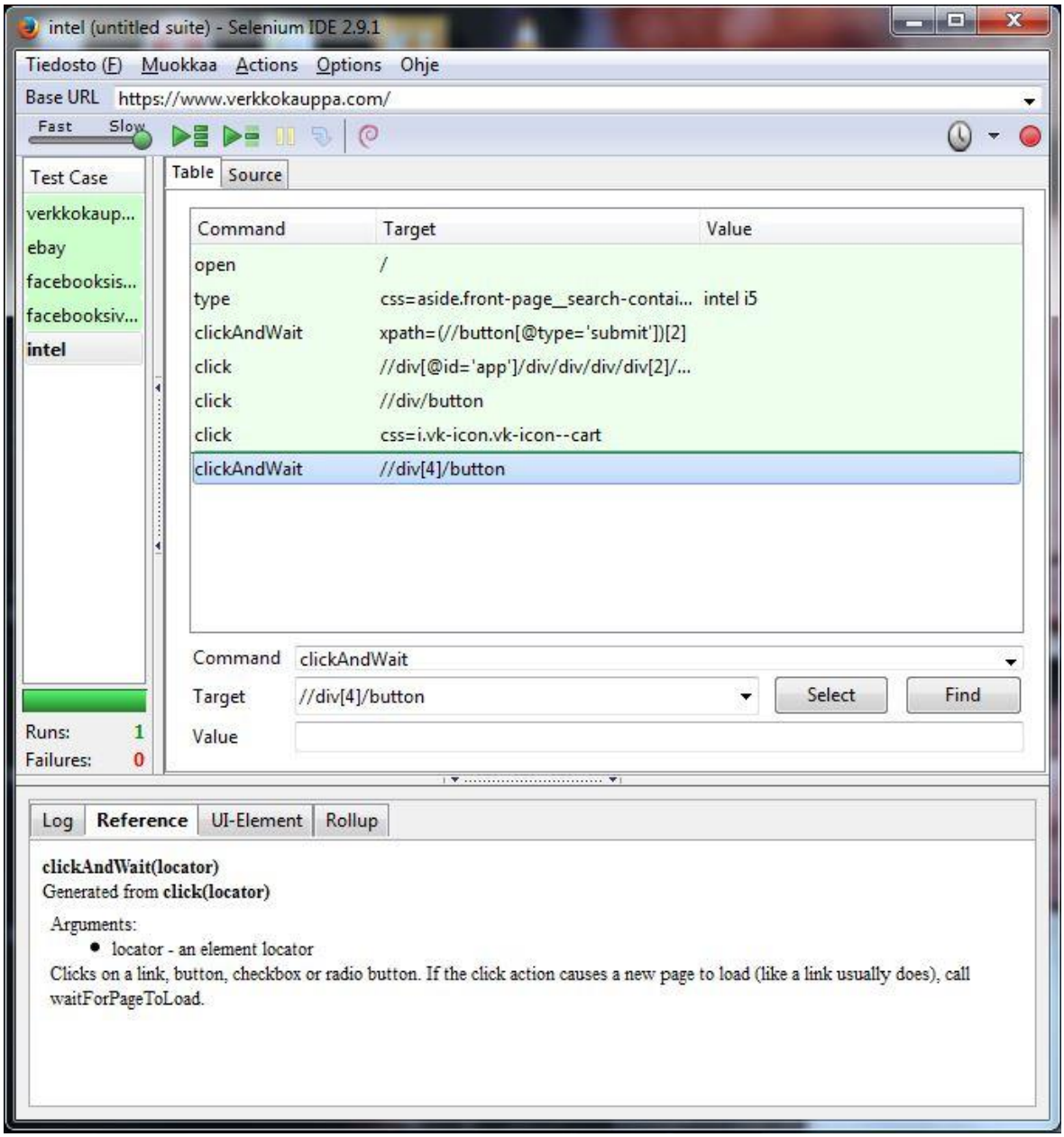


Kuva 7. Iltalehti-testauksen koodi

Koodissa on samat asiat kuin Table-sivussa. Koodi sisältää samat komennot kuin Command-kohdassa sekä Target-kohdan, johon annetaan tieto siitä, mihin kohtaan Seleniumin tulee toimia. Sisältö annetaan <tr> rivien väliin ja itse komentojen ja Targettien sisältö <td> rivien sisälle.

Testitapaus 4. Verkkokauppa

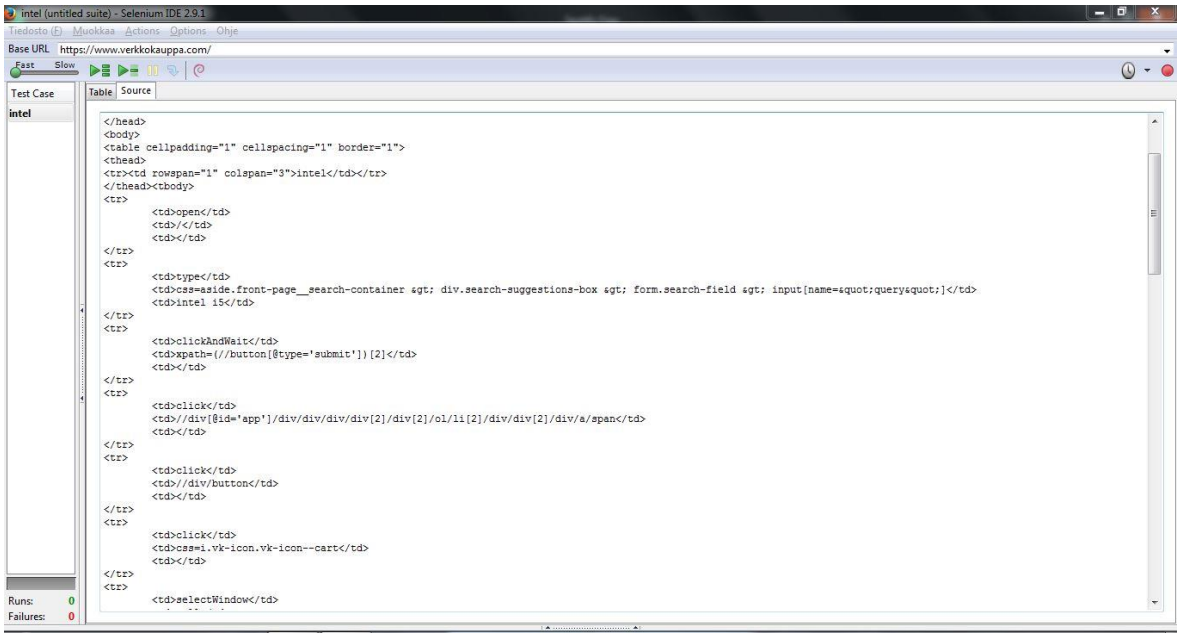
Seuraava testitapaus on haastavampi. Testataan Verkkokauppa.com-sivulla prosessori-tilauksen tekemistä, joka oli aikaisemmin tehty manuaalisena testauksena. Tämä testitapaus on lyhyempi eikä steppejä ole yhtä paljon käytetty kuin manuaali testitapauksessa. Aluksi Base URL – kohtaan tuli vaihtaa osoitteeksi https://www.verkkokauppa.com/. Tässä kuva 8 testitapauksesta.



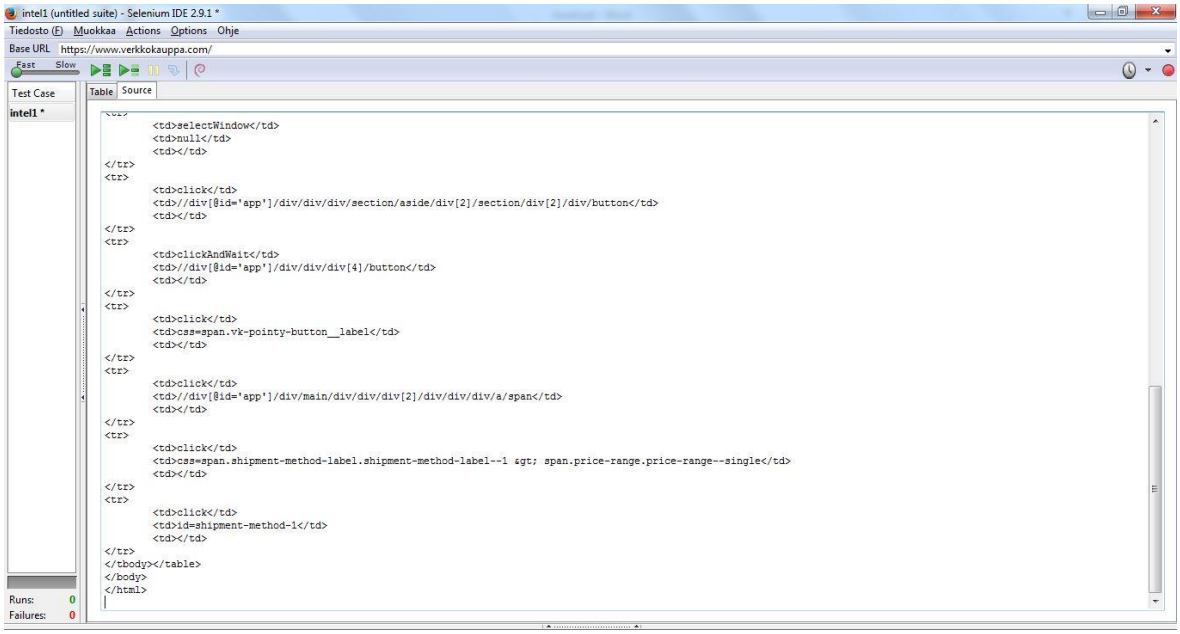
Kuva 8. Verkkokauppa.com tilaus

Tässä ongelmaksi muodostui OpenWindow-komennot, joilla yritettiin päästä sivustolla eteenpäin seuraavalle sivulle. Näiden muuttaminen Click-muotoon helpotti testitapauksen luomista. Myös lievempi ongelma toisessa testitapauksessa oli, että muutamassa kohdassa tuli sivun antaa latautua, jotta klikkaaminen onnistui seuraavalla sivulla. Tähän auttoi jälleen ClickAndWait- komennon käyttö.

Tässä vielä kaksi kuvaa siitä, miten tämän testitapauksen voi luoda koodi muotoon (kuva 9 ja kuva 10). Jälleen tämä sisältää samat ominaisuudet kuin komentotyyliin koottu testitapaus.



Kuva 9. Verkkokauppa.com testi kuva 1



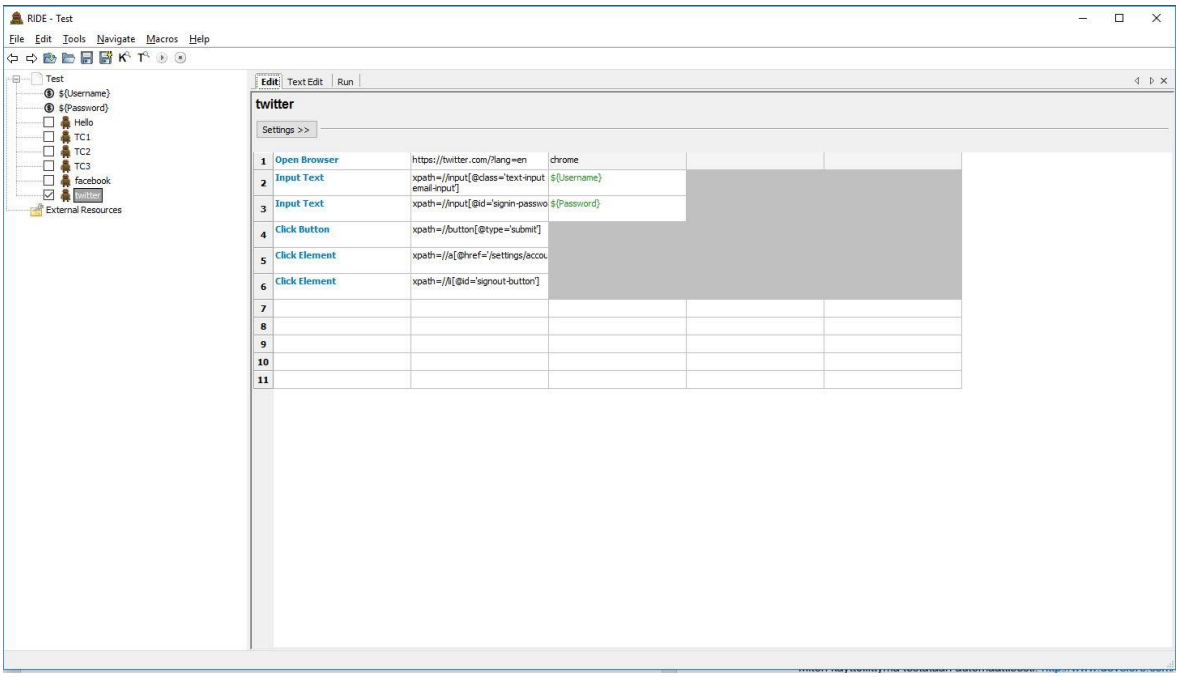
Kuva 10. Verkkokauppa.com testi kuva 2

5.3 Robot Framework testaus

Tässä osiossa testataan Robot Frameworkilla, jota suoritetaan Python-kielellä.

Testitapaus 5. Twitter-kirjautuminen.

Testitapauksessa kirjaudutaan sisälle Twitter-tiliin ja saman tien kirjaudutaan ulos. Testitapauksessa käytetään Google Chrome -selainta. Tässä kuva 11 komennoista ja lauseista, joita testitapauksessa käytettiin.

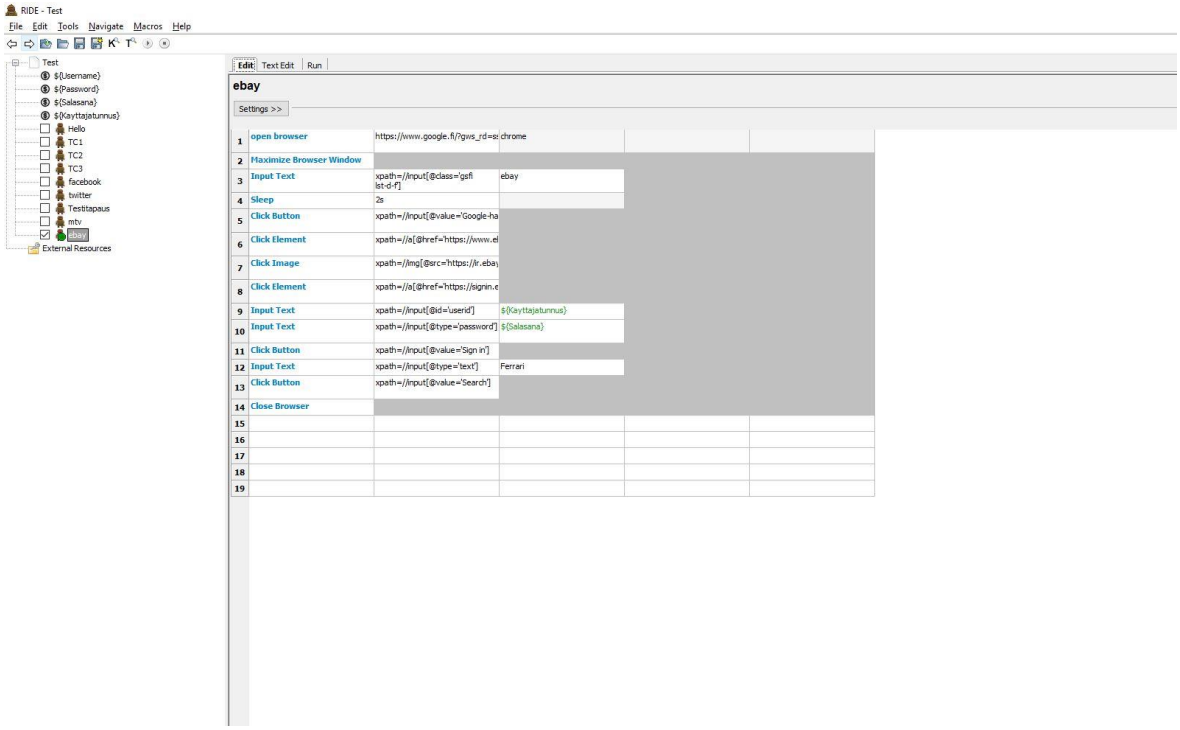


Kuva 11. Twitter-tiliin sisään- ja uloskirjautuminen

Open Browser -komennolla avataan selain ja mennään sivulle https://twitter.com. Input Text -komennolla ja XPath-lauseella annetaan käyttäjätunnus ja salasana oikeaan kohtaan. Click Button -komennolla painetaan Log in -painikkeesta. Sisäänkirjautumisen jälkeen Click Element -komennolla painetaan oman käyttäjätunnuksen ikonista ja samalla komennolla valitaan Log out -painike. Testitapaus ajettiin onnistuneesti.

Testitapaus 6. Ebay

Robot Frameworkilla testattiin vielä yhtä haastavampaa testitapausta, joka sisälsi enemmän steppejä sekä eri ominaisuuksia. Testitapauksessa haetaan ensin Googlella Ebay, kirjaudutaan sisälle Ebay tunnukseen sekä testataan sivuston hakukentän toimintaa. Tässä kuva 12 Ebay–testitapauksesta.

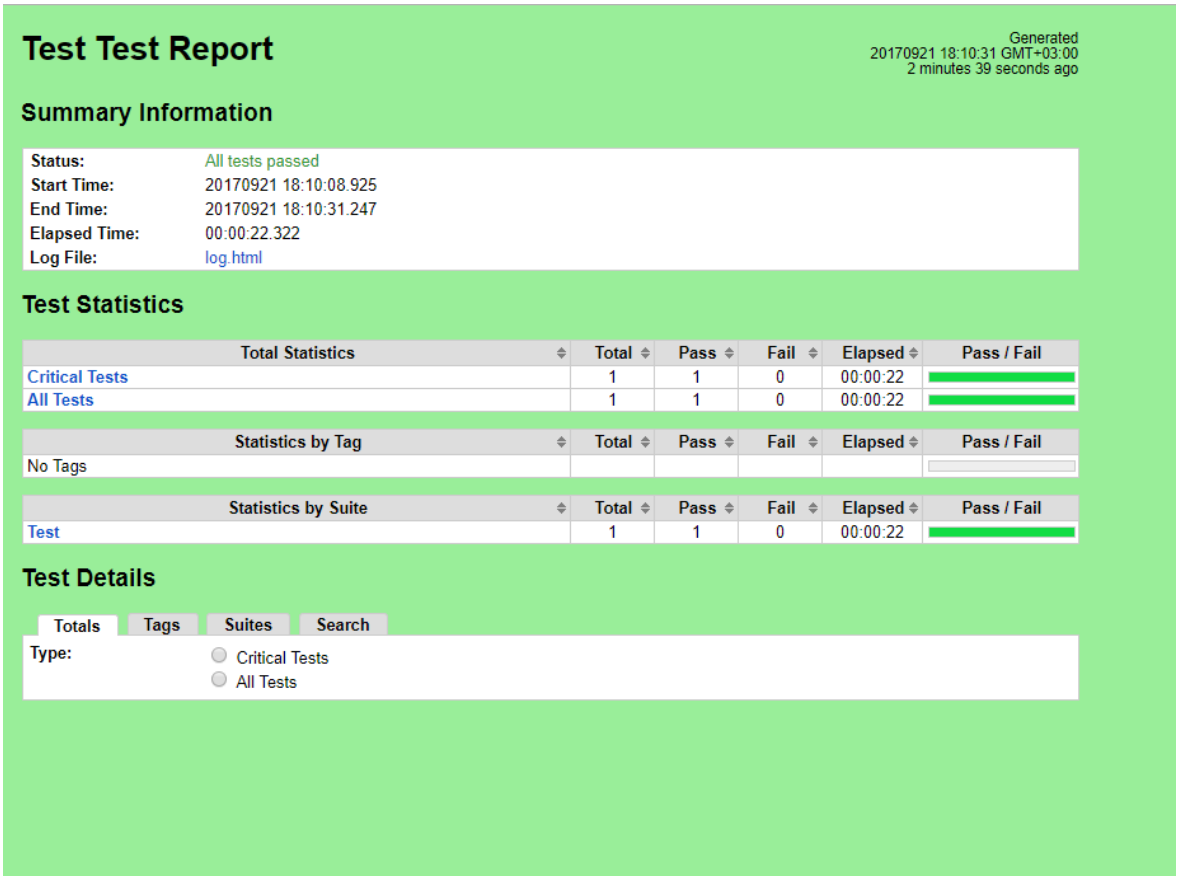


Kuva 12. Ebay–sivuston testaus

Tässä testitapauksessa ensin avataan selain ja siirrytään Googleen. Teksti sijoitetaan Googlen hakukenttään Input Text -komennolla. Siirrytään kyseiselle sivulle Click Element -komennolla, joka valitsee Googlesta Ebay linkin. Kirjaudutaan sisälle muuttujilla käyttäjätunnus ja salasana. Painetaan Sign in -painiketta Click Button -komennolla. Haetaan sivustolla haulla Ferrari, jälleen Input Textilla määritellään teksti ja Click Buttonilla painetaan Search painiketta. Lopuksi vielä testitapaus sulkee selaimen Close Browser -komennolla.

Testitapaus onnistui hyvin. Ainoa mikä aiheutti ongelmia oli ponnahdusikkuna, joka tuli esille Ebayn etusivulla. Tämän sain poistettua Click Image -komennolla, jolla käskettiin painamaan kuvan X kohdasta, joka sulki ponnahdusikkunan (mainoksen). Testitapaus onnistui hyvin, tosin järjestystä komennoille tuli miettiä tarkasti.

Katsotaan vielä lopuksi tämän testitapauksen raporttia. Robot Framework antaa testauksen lopuksi linkin, josta pääsee tarkastamaan jokaisen testiajon raporttia.



Kuva 13. Ebay–testauksen raportti

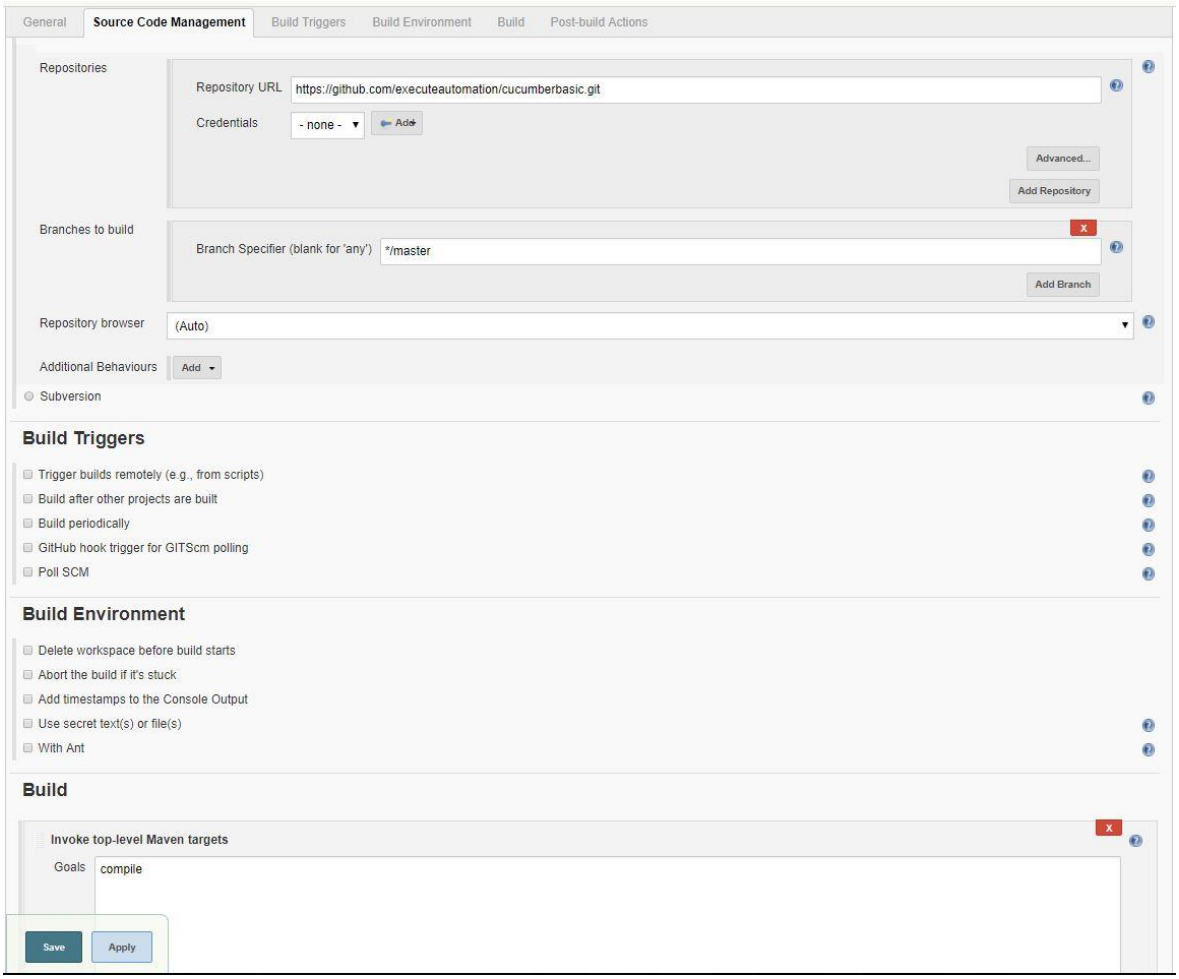
Kuvassa 13 nähdään raportti tästä testitapauksesta. Kuvasta näkee, että testitapauksen kaikki stepit on ajettu onnistuneesti. Testitapauksen jälkeen sivuston voisi työelämässä siirtää tuotantoon.

5.4 Jenkins testaus

Viimeisenä sovelluksena automaatiotestausta suoritetaan Jenkinsillä. Jenkinsillä on mahdollista luoda eri projekteja eri tarkoituksiin. Tässä osiossa luodaan kaksi Jenkins projektia Freestyle ja Pipeline. Testitapauksen tilanteen voi kuvata työelämästä. Verkkosivussa on todettu olevan vika sisäänkirjautumisen kanssa. Ohjelmoija on mielestään saanut vian korjattua ja ohjelmoija on suorittanut moduulitestauksen. Nyt regressiotestauksella testataan sisäänkirjautumista.

Testitapaus 7. Execute Automation -sivulle sisäänkirjautuminen

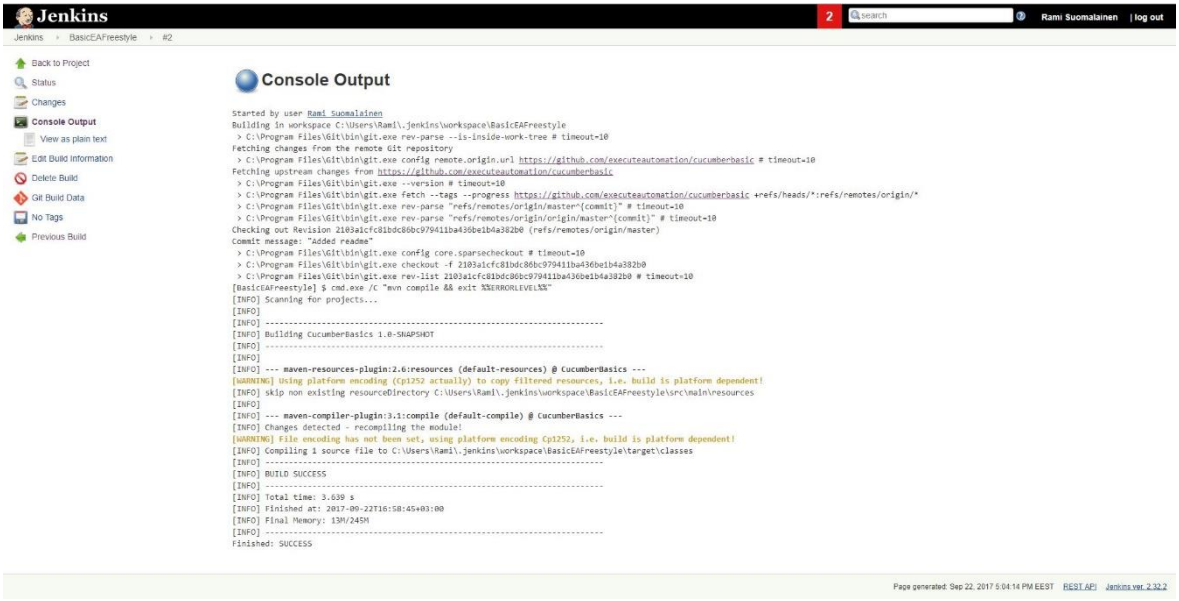
Aloitetaan luomalla Freestyle-projekti, jolla testataan yhteyttä Githubiin, josta löytyy testitapauksen lähdekoodi. Käytössä on aikaisemmin mainittu yleinen kaikkien käytettävissä oleva Githubista löytyvä projekti Cucumberbasic. Tässä kuva 14 tiedoista, joilla testitapaus on luotu.



Kuva 14. Freestyle-projekti

Testitapausta luodessa tärkeintä oli kuvassa 14 valita Source Code Management -kohdassa Git ja aukeavaan osaan antaa oikea URL–osoite Github projektiin. Kuvan 14 alaosassa näkyvään Build -kohtaan tuli valita Invoke top-level Maven targets, sekä tähän antaa arvoksi Compile, jotta projektin saa koottua projektiksi Jenkinsiin.

Ensimmäisellä kerralla testitapauksen ajaminen ei onnistunut. Tähän auttoi Jenkinsin uudelleenkäynnistys. Uudelleenkäynnistyksen sekä uuden Freestyle-projektin luomisen jälkeen samoilla arvoilla, testitapauksen ajo suoritettiin onnistuneesti. Tämä testitapaus mahdollisti lähdekoodin käytön Githubissa, joten tämän yhteyden saaminen toimivaksi oli tärkeää testitapauksen kannalta. Kuvassa 15 näkee, että testitapaus on ajettu onnistuneesti.



Kuva 15. Ensimmäisen testitapauksen ajo

Tämän jälkeen luodaan Pipeline-projekti, jolla itse sisäänkirjautuminen suoritetaan Githubista löytyvällä lähdekoodilla. Pipeline-projekti käydään läpi sen takia, koska Pipeline-projektien suosio on nousussa. Testitapauksen olisi voinut myös suorittaa pelkästään Freestyle-projektina, mutta tämä oli liian yksinkertaista. Pipeline-projektissa tuli nähdä testitapauksen luomiseen enemmän vaivaa. Tähän testiprojektiin käytetään toista Github-projektia Selenium With Cucumber. Poiketen Freestyle-projektista tämän testitapauksen ajamiseen käytetään scriptiä. Alustaminen Pipeline-projektiin oli hyvin samanlaista kuin Freestyle-projektissa, mutta URL tuli vain muuttaa Selenium With Cucumber osoitteeksi. Kyseisestä osoitteesta löytyvät tarvittavat lähdekoodit, joita on kaksi kappaletta. Alla kuvat 16 ja 17 lähdekoodeista, jotka löytyvät Githubista.

47 lines (33 sloc) | 1.06 KB Raw Blame History

```java
package steps;

import Base.BaseUtil;
import cucumber.api.Scenario;
import cucumber.api.java.After;
import cucumber.api.java.Before;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

/**
 * Created by Karthik on 10/17/2016.
 */
public class Hook extends BaseUtil{


    private BaseUtil base;

    public Hook(BaseUtil base) {
        this.base = base;
    }

    @Before
    public void InitializeTest() {

        System.out.println("Opening the browser : Firefox");

        /*System.setProperty("webdriver.firefox.marionette", "D:\\Libs\\geckodriver.exe");
        base.Driver = new FirefoxDriver();*/


        //Chrome driver
        System.setProperty("webdriver.chrome.driver", "C:\\Libs\\chromedriver.exe");
        base.Driver = new ChromeDriver();
    }


    @After
    public void TearDownTest(Scenario scenario) {
        if (scenario.isFailed()) {
            //Take screenshot logic goes here
            System.out.println(scenario.getName());
        }
        System.out.println("Closing the browser : MOCK");
    }

}
```

Kuva 16. Lähdekoodi 1

```
 * Created by Karthik on 10/15/2016.
 */
public class LoginStep extends BaseUtil{

    private  BaseUtil base;

    public LoginStep(BaseUtil base) {
        this.base = base;
    }

    @Then("^I should see the userform page$")
    public void iShouldSeeTheUserformPage() throws Throwable {

        Assert.assertEquals("Its not displayed", base.Driver.findElement(By.id("Initial")).isDisplayed(), true);
    }

    @Given("^I navigate to the login page$")
    public void iNavigateToTheLoginPage() throws Throwable {

        System.out.println("Navigate Login Page");
        base.Driver.navigate().to("http://www.executeautomation.com/demosite/Login.html");
    }


    @And("^I click login button$")
    public void iClickLoginButton() throws Throwable {
        LoginPage page = new LoginPage(base.Driver);
        page.ClickLogin();
    }


    @And("^I enter the following for Login$")
    public void iEnterTheFollowingForLogin(DataTable table) throws Throwable {
        //Create an ArrayList
        List<User> users =  new ArrayList<User>();
        //Store all the users
        users = table.asList(User.class);

        LoginPage page = new LoginPage(base.Driver);

        for (User user: users){
           page.Login(user.username, user.password);
        }
    }

    @And("^I enter ([^\"]*) and ([^\"]*)$")
    public void iEnterUsernameAndPassword(String userName, String password) throws Throwable {
        System.out.println("UserName is : " + userName);
        System.out.println("Password is : " + password);
    }

    @Then("^I should see the userform page wrongly$")
    public void iShouldSeeTheUserformPageWrongly() throws Throwable {

        Assert.assertEquals("Its not displayed", base.Driver.findElement(By.id("sdfgdsfsd")).isDisplayed(), true);
    }

```

Kuva 17. Lähdekoodi 2

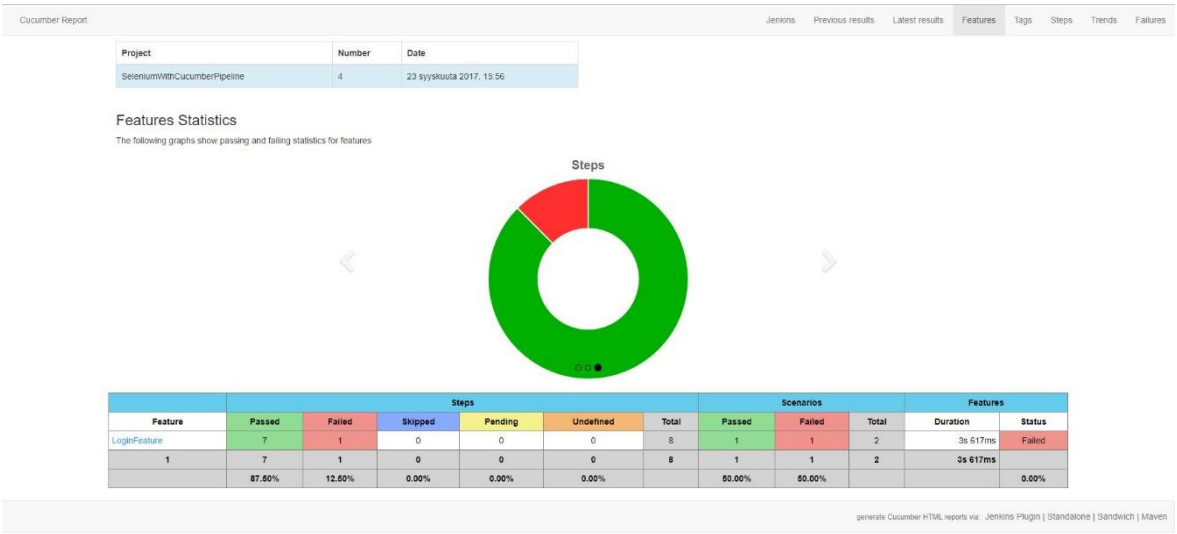
Luodaan scripti, jolla kirjaudutaan sisälle Execute Automation -sivulle käyttäjätunnuksella sekä salasanalla. Aloitetaan luomaan scripti. Tässä kohtaa todella hyödylliseksi työkaluksi Jenkinsissä osoittautui Pipeline Syntax. Tässä kuva 18 Pipeline Syntaxista.

Kuva 18. Pipeline Syntax -työkalu

Kyseisellä työkalulla onnistuu yksinkertaisesti luomaan eri scriptejä antamalla Sample Text -kohtaan haluttu scriptin muoto, avautuvaan osaan haluttu tieto (esimerkiksi URL–osoite) sekä valitsemalla Generate Pipeline Script. Tässä kuva 19 luodusta scriptistä, joka luotiin Pipeline Syntaxin avulla.

Kuva 19. Scripti

Scriptissa Git komennolla haetaan projektin lähdekoodi Githubista sekä mvn (Maven) komennolla ajetaan kyseinen testitapaus. Tämän jälkeen tallennetaan testitapaus Save-painikkeella ja suoritetaan testitapauksen ajo. Testitapauksen ajo suoritettiin onnistuneesti. Katsellaan myös Jenkinsissä testitapausten tuloksia raporttien avulla. Tämän sai asennettua Jenkinsiin Pluginnin kautta. Asennettava ominaisuus on nimeltään Cucumber Reports.



Kuva 20. Raportti Jenkins testitapauksesta

Kuvan 20 raportista nähdään, että ainoastaan yksi steppi viimeisessä testitapauksessa on ajettu epäonnistuneesti ja muut stepit ajettu onnistuneesti. Tämän testitapauksen voi merkata onnistuneeksi. Työelämässä yhdestä epäonnistuneesta stepistä tulisi luoda vikailmoitus. Muuten voi todeta, että sivusto on valmis siirrettäväksi tuotantoon.

Case Study osiossa suoritettiin yhteensä seitsemän testitapausta. Kaksi ensimmäistä testitapausta olivat manuaali testitapauksia testitapaus 1. www.katsomo.fi sekä testitapaus 2. www.verkkokauppa.com tilauksen tekeminen. Näihin testitapauksiin ei kummoisia testisovelluksia tarvitse. Case Study osiossa testitapaukset olivat luotu Microsoft Exceliin, mutta testitapauksiin voisi käyttää esimerkiksi pilvipalvelua, johon onnistuu testitapauksien luominen ja suorittaminen.

Automaatiotestitapaukset Seleniumilla olivat testitapaukset 3. www.iltalehti.fi sekä testitapaus 4. www.verkkokauppa.com (tässä testattiin samat asiat kuin testitapauksessa 2). Robot Frameworkilla suoritettiin testitapaukset 5. Twitter-kirjautuminen ja testitapaus 6. Ebay-testaaminen. Viimeiseksi automaatiotestausta suoritettiin Jenkinsillä testitapauksessa 7. Execute Automation -sivulle sisäänkirjautuminen.

6 Kyselytutkimus

Tässä opinnäytetyössä on tehty Case Studyn lisäksi kyselytutkimus, joka on suoritettu tutkivana kyselynä. Kyselytutkimuksessa selvitetään, mitä testaajan työ pitää kokonaisuutena sisällään, mitä testaajan työ on, mitä testaustapoja tai menetelmiä käytetään sekä mikä koulutuksen taso tai muut ominaisuudet ovat tärkeitä testaajan työssä tai testaamisesta kiinnostuneille. Päätavoitteena kyselyssä oli selvittää yleistä tietoa testaamisesta, mitä tämä sisältää, miten testaamista tulisi suorittaa ja mitkä ominaisuudet testaajan on hyvä hallita. Lisäksi selvitettiin mikä testaamisessa on parasta ja huonointa, miten vastaajat näkevät testaamisen tulevaisuuden, miten kouluissa ja työpaikassa tulisi huomioida testaus ja mielipiteitä automaatio- ja manuaali testaamisesta. Myös heidän mielipidettään testaajien arvostuksesta käydään läpi kyselytutkimuksessa. Ensiksi käydään läpi kysymykset sekä kysymyksien tarkoitus. Lopuksi käydään vastauksia läpi sekä kerätään vastauksista kokonaisuus.

Kysely lähetettiin viidelle henkilölle, jotka työskentelevät testaajana. Kaikki viisi henkilöä vastasivat kyselyyn. Osallistujien nimikkeet ovat Junior Software Tester, QA Manager, QA Engineer sekä kaksi osallistujista ovat nimikkeiltään QA Consultant.

6.1 Kysymykset

Tässä osiossa käydään läpi kyselytutkimukseen valitut kysymykset sekä käydään läpi, mitä kysymyksillä selvitetään.

Kysymys 1. Kuinka pitkään olet työskennellyt testaajana?

Kysymyksellä selvitetään, miten pitkä kokemus vastaajilla on testaamisesta ja lasketaan vastaajien työkokemus keskiarvona.

Kysymys 2. Mikä on koulutuksesi IT- tai muulla alalla?

Kysymyksellä selvitetään osallistujien koulutukset sekä näiden taso. Koulutuksia vertaillaan keskenään, kuinka monella on korkeakoulututkinto ja kenellä ammattikoulutason tutkinto.

Kysymys 3. Näetkö vielä tulevaisuudessa työskenteleväsi testaamisen parissa? Miksi?

Selvitetään osallistujien mielenkiinto jatkaa testaajana työskentelyä myös tulevaisuudessa ja mitkä ovat suurimmat syyt, miksi testaajana työskentely kiinnostaa.

Kysymys 4. Mikä on testaajan työssä parasta? Ja mikä taas huonointa?

Selvitetään, mikä vastaajia kiinnostaa testaajan työssä, mikä testaamisessa on miellyttävin asia ja mitkä taas ovat testaajan työn huonoja puolia. Näitä vertaillaan keskenään ja kerätään kokonaisuus eri vastauksista, mitkä vastaukset saivat eniten huomiota.

Kysymys 5. Miten mielestäsi testaajia arvostetaan työelämässä?

Aikaisemmin opinnäytetyössä mainittiin, että testaajat ovat joidenkin mielestä jokseenkin aliarvostettuja työelämässä. Selvitetään osallistujien mielipiteitä, mitä mieltä he ovat tästä asiasta. Onko heillä sama vai eri mielipide testaajien arvostamisesta, kuin opinnäytetyön tekijällä.

Kysymys 6. Sopeutuuko mielestäsi kuka vain testaajaksi vai tarvitaanko testaajaksi IT–alan tai muu tekniikan alan koulutus?

Testaajan työstä on väärä kuva, että kuka vain soveltuu testaajaksi. Selvitetään osallistujien mielipiteitä, minkä tason koulutus testaajalla tulee vastaajien mielestä olla tai mitkä seikat ovat tärkeitä, jos testaajan työ kiinnostaa.

Kysymys 7. Mitkä ovat testaajan tärkeimpiä ominaisuuksia?

Kerätään vastauksia eri ominaisuuksista, jotka vastaajat näkevät tärkeiksi testaajan työssä. Mitä tulee osata ja mitä tulee kehittää tulevaisuudessa, jos testaajana haluaa omaa uraansa kehittää?

Kysymys 8. Mitä tulee mielestäsi ottaa huomioon testitapausta luodessa?

Kysymyksellä selvitetään testitapauksen luomisen prosessia. Selvitetään, mitä testitapausta luodessa tulee ottaa huomioon, mitä testitapauksen tulee sisältää sekä miten laajasti tulee eri testitapaukset suunnitella.

Kysymys 9. Kuinka paljon testaamisessa tulee mielestäsi ymmärtää ohjelmistokehityksen eri vaiheita?

Kysymyksellä selvitetään, onko ohjelmistokehityksen eri vaiheiden ymmärtäminen tärkeää sekä miten oleellista testaajalle on ymmärtää ohjelmistokehityksen eri vaiheet?

Kysymys 10. Miten yrityksissä tulisi mielestäsi kehittää testaamista?

Selvitetään, miten yrityksien tulisi panostaa testaamiseen ja miten mahdollisesti testaamista tulisi eri työpaikoissa kehittää.

Kysymys 11. Tulisiko kouluissa opettaa enemmän testaamista?

Kouluissa ei testaamisesta puhuta paljoa sekä opetus kouluissa on hyvin vähäistä. Tässä tiedustellaan osallistujien näkökulmaa, tulisiko koulujen opettaa enemmän testaamista sekä mahdollisia tapoja, miten tämän voisi kouluissa toteuttaa?

Kysymys 12. Mitä mieltä olet automaatiotestaamisesta?

Automaatiotestauksen suosio on kasvussa. Selvitetään, mitä mieltä vastaajat ovat automaatiotestaamisesta sekä miten tätä tulisi työelämässä hyödyntää.

Kysymys 13. Mikä on mielestäsi manuaalisen testauksen tulevaisuuden näkymät?

Yhä useampi yritys on siirtymässä automaatiotestaamiseen, joten selvitetään osallistujien mielipiteitä manuaalisen testaamisen tulevaisuudesta ja miten osallistujat näkevät manuaalisen testaamisen roolin testaajan työssä tulevaisuudessa.

Kysymys 14. Käyttäisitkö enemmän manuaalista tai automaatiotestausta?

Selvitetään vastaajien mielipiteitä testitavoista, joita he ovat tottuneet käyttämään ja kumpaa testaustapaa vastaajat mielellään käyttävät.

6.2 Vastauksien analysointi

Tässä osiossa kerätään kokonaisuus vastauksista, jotka saatiin jokaiseen kysymykseen. Jokaista vastausta ei olla mainittu erikseen vaan osioon on kerätty kokonaisuus eri vastauksista, joita kysymykset saivat.

Kysymys 1. Osallistujien keskiarvo testaajana työskentelyssä on 13 vuotta. Vähiten kokemusta on vastaajalla, joka on työskennellyt testaajana 1,5 vuotta, kun pisimpään työskenneellä vastaajalla on kokemusta 23 vuotta testaamisesta.

Kysymys 2. Vastaajista kaksi ovat suorittaneet yliopistossa tietojenkäsittelytieteen koulutuksen. Kaksi ovat insinöörejä, toinen automaatioinsinööri sekä tietoliikennetekniikan insinööri. Yhdellä on ammattikoulutason koulutus sekä ohjelmoijan koulutus.

Kysymys 3. Kaikki osallistujat näkevät työskentelevänsä vielä tulevaisuudessa testaajana. Mainintana oli myös muut IT – alan työt, jotka voisivat tulevaisuudessa kiinnostaa, mutta testaajana työskentely koko uran ajan ei ole pois suljettu vaihtoehto. Kiinnostuksesta jatkaa testaajana mainittiin laadun tarve ohjelmistoissa, joka ei katoa tulevaisuudessa mihinkään. Testaajan työ antaa moninaisia haasteita ja erilaiset projektit opettavat jatkuvasti uusia aloja ja tekniikoita. Testauksessa on mahdollisuus oppia testattavista järjestelmistä laaja-alaisesti, josta on hyötyä myös muissa työtehtävissä toimiessa.

Kysymys 4. Parhaana puolena testaajan työssä vastaajien mielestä on, että pääsee vaikuttamaan sovelluksen laatuun sekä miten tämä näyttäytyy loppuasiakkaalle. Virheiden havaitseminen, jotka ovat kriittisiä sovelluksissa, näiden löytäminen myös mainittiin hyvänä puolena testaajan työssä. Mainintoja saivat myös IT–alan kiinnostus, etätyöskentelyn mahdollisuus, pääsee keskustelemaan ja kommunikoimaan eri sidosryhmien kesken ja jokainen työpäivä ei ole samanlainen vaan vaihtelevuus on suurta.

Huonona puolena vastaajat näkivät kiireessä testaamisen tai ilman tarvittavia taustatietoja tai dokumentaatiota siitä, mitä pitäisi testata tai miten testattavan ominaisuuden pitäisi toimia. Testaajien arvostus mainittiin myös olevan joissain yrityksissä vähäistä, jolloin testaamiseen ei panosteta

tarpeeksi. Muita huonoja puolia olivat jatkuva muutos ja näiden tuomat paineet testaamiselle, testauksen rutinoituminen ja manuaalinen toisto.

Kysymys 5. Vaikka aikaisemmassa kohdassa mainittiin testaajien arvostus olevan joissain yrityksissä vähäistä, niin kokonaisuutena testaajan arvostus nähtiin hyvänä tai kohtuullisen hyvänä riippuen yrityksen politiikasta. Testauksella on oltava yrityksen johdon tuki. Johdon tuella näkee, että testaamista arvostetaan eikä pidetä riesana. Ohjelmistokehittäjät näkevät työelämässä testaamisen tärkeänä eikä myöskään testaajien ja kehittäjien välillä ole köydenvetoa vaan yleensä he tulevat hyvin toimeen keskenään. Mainintana olivat myös asiakkaat, jotka eivät välttämättä ymmärrä testauksen tärkeyttä ohjelmiston toimivuuden kannalta.

Kysymys 6. IT–alan koulutus ei nähty välttämättömäksi riippuen testaustehtävistä, mutta tekniikan alan koulutuksesta on hyötyä testaajan työssä. Myös kaupallinen koulutus on tuottanut hyviä testaajia. Tärkeää on kiinnostus tai harrastuneisuus esimerkiksi ohjelmistokehitystä kohtaan. Koulutus nähtiin tärkeäksi, jotta ymmärtää miten asioita toteutetaan ja miten niiden toimintaa voi verifioida. Myös koodin lukutaidosta, taidoista käyttää eri päätelaitteita ja käyttöjärjestelmiä sekä kielitaidosta on hyötyä.

Kysymys 7. Tärkeimpinä ominaisuuksina vastaajat näkivät kiinnostuksen testaamista kohtaan, uteliaisuus, rohkeus kysyä myös tyhmiä kysymyksiä, tarkkaavaisuus, pikkutarkkuus, ymmärrystä miten ohjelman tulee toimia asiakkaan näkökulmasta, kommunikaatiotaidot, kärsivällisyys sekä hyvä paineensietokyky.

Kysymys 8. Vastaajien mielestä testitapauksen tulee olla yksinkertainen, jossa testataan yhtä asiaa kerrallaan. Testitapauksia tulee myös luoda negatiivisille testeille eli testata toiminnallisuutta väärillä syötteillä. Jokaisessa testitapauksessa tulisi olla ennakoehdot ennen suoritusta, stepit ja odotetut lopputulokset eli hyväksymiskriteerit. Testitapauksella tulee olla kattava kuvaus. Tarkka kuvaus opastaa paremmin kokematonta testaajaa. Testitapauksen luomiseen tarvitaan myös loogista päättelykykyä testitapauksen tavoitteen saavuttamiseksi.

Kysymys 9. Ohjelmistokehityksen eri vaiheiden ymmärtäminen sai vaihtelevia mielipiteitä vastaajilta. Kaksi ei nähnyt vaiheiden ymmärtämistä kauhean tärkeäksi tai perusasioiden ymmärtäminen riittää. Kolme vastaajista

näki tämän erittäin tärkeänä. Ilman niiden ymmärtämistä on erittäin vaikea suunnitella testauksen aikataulua tai edes minkälaista testausta tulisi tehdä. Kuinka suuria muutoksia versioissa tulee ja minkälaista testausta niille on jo suoritettu on tärkeää suunnittelun kannalta. Tärkeää on myös ymmärtää missä vaiheessa testaus tulee suorittaa ohjelmistokehityksessä. Vaikka sanotaan, että ”tieto tuo tuskaa” ohjelmistokehityksessä tämä mahdollistaa ennakoinnin mahdollisiin vikoihin. Testaajan tulisi olla sovelluksen kehityksen alkuvaiheista lähtien mukana, antamassa omaa näkemystään.

Kysymys 10. Testaamisen kehittämistä yrityksissä tulisi vastaajien mielestä antaa tarpeeksi aikaa testaukselle, jotta haluttu lopputulos voidaan saavuttaa. Projekteissa tulisi olla enemmän kuin yksi testaaja, jotta yhdelle testaajalle ei tule liikaa tehtäviä. Testaajia työskentelee paljon konsultteina ja joitain testaajia kannattaa pitää jatkuvuuden nimissä omilla palkkalistoilla, vaikka työt välillä vähenisivätkin. Silloin he voivat suunnitella ja ratkoa testauksen ongelmia seuraavia projekteja varten. Myös tähän kysymykseen vastattiin testaajien tärkeys olla mukana ohjelmiston suunnitteluvaiheessa.

Kysymys 11. Kouluissa testaamisen opettaminen sai vaihtelevia vastauksia. Yksi vastasi koulusta saaneen tarvittavan pohjan testaamiselle, mutta käyttöliittymä- ja automaatiotestausta voisi kouluissa käydä enemmän läpi. Neljän mielestä kouluissa pitäisi opettaa testaamista enemmän. Vaihtoehtoina olivat kurssina suorittaminen tai suuntautumislinja sekä ylemmän, että alemman tason koulutuksessa. Testaus ohitetaan usein koulutuksessa sivulauseella. Vastaajista yksi mainitsi, että oman koulunsa aikana ei testaamisesta puhuttu mitään.

Kysymys 12. Automaatiotestaaminen nähtiin hyvänä, koska tämä vapauttaa rutiineista. Hyvin suunniteltuna ja toteutettuna automaatiotestaaminen vähentää manuaalisen testauksen tarvetta. Vastaajien mukaan automaatiotestausta kannattaa käyttää yksikkötestauksessa, jotta saataisiin mahdollisimman hyvä kattavuus kaikista poluista, jotka pitää käydä lävitse. Ylemmillä testaustasoilla se soveltuu hyvin regressiotestaamiseen. Automaatiotestauksessa on se huono puoli, että sillä harvoin löydetään todellisia vikoja koska scriptit tehdään toimivaksi. Automatisointi ei poista missään tapauksessa muun testauksen tarvetta tai automaattisesti paranna testaamisen laatua.

Kysymys 13. Tulevaisuudessa manuaalisen testaamisen tarve nähtiin tarpeelliseksi. Osa painopisteistä tulee manuaalisessa testaamisessa vaihtelemaan, mutta se ei ole katoamassa mihinkään. Manuaalinen testaaminen nähtiin hyödylliseksi varsinkin uusien ja kehityksen alla olevien ominaisuuksien testaamisessa, kun automaatiotestien ylläpitoon menisi enemmän aikaa.

Kysymys 14. Viidestä vastaajasta kaksi käyttäisi automaatiotestausta ja yksi manuaalista testaamista. Kaksi mainitsivat molemmat olevan yhtä tärkeitä. Mainintaa sai myös automaatiotestitapaukset, jotka tulisivat ainakin kerran ajaa manuaalisesti. Automaatiotestaus vapauttaa resursseja manuaaliseen testaukseen sekä tutkivaan testaukseen säästyy enemmän aikaa.

7 Johtopäätökset ja pohdinta

Tässä osiossa käydään yhteenvetona opinnäytetyön tärkeimmät kohdat sekä havainnot, jotka ilmenivät tutkimuskyselyssä. Lopuksi vielä käydään yleisesti läpi opinnäytetyön tekijän itsearviointi sekä mielipiteitä, miten opinnäytetyö eteni, missä opinnäytetyössä onnistuttiin sekä mikä olisi voinut mennä opinnäytetyössä paremmin.

7.1 Yhteenveto tutkimuskyselystä

IT–alan koulutus ei ole pakollinen koulutus, jolla testaajan työpaikan voi saada vaan yleisellä kiinnostuksella ohjelmistokehitystä kohtaan tai omien harrastuksien kautta on mahdollista kehittää itseään testaajana. Osallistujat ovat kiinnostuneita kehittämään uraansa testaajana, koska laadun tarve ohjelmistoissa ei tule katoamaan tulevaisuudessa mihinkään.

Hyvänä puolena testaajan työssä vastaajat näkivät sen, että ohjelmiston laatuun pääsee vaikuttamaan sekä kehittäjiä pystyy auttamaan ohjelmaa luodessa. Vikojen löytäminen, jotka heikentävät ohjelmiston laatua on myös arvokasta testaajan työssä. Huonoimpana puolena ylivoimaisesti eniten mainintoja saivat kiireessä testaaminen. Tutkimuskyselyssä selvisi, että aikaa on varattava sopiva määrä testaamiseen, eikä tätä tule suorittaa hätäisesti. Tämä parantaa testaamisen laatua sekä samalla ohjelmiston laatu paranee myös, kun mahdollisimman monta vikaa löydetään ohjelmistosta. Dokumentaatio tulee olla kunnossa testaajan kannalta, joka helpottaa työskentelyä huomattavasti. Rutiininomaiset testitapaukset sekä manuaalinen toisto nähtiin myös huonona puolena. Yrityksien on hyvä panostaa koulutuksien tarjotaan sekä päästää testaajat mukaan jo ohjelmiston suunnitteluvaiheeseen. Yhdelle testaajalle ei tule antaa liikaa vastuuta vaan tarvittaessa projektissa tulee olla toinen testaaja. Vastaajat pitävät testaamista arvostettuna työpaikoissa eivätkä pitäneet testaajan työtä aliarvostettuna.

Tärkeimmät ominaisuudet testaajalla ovat yleisiä ominaisuuksia, joita työelämässä tarvitaan ja varsinkin IT–alalla. Kommunikaatiotaidot, uteliaisuus, mielenkiinto, rohkeus kysyä kysymyksiä, luova hulluus, halu ymmärtää asiakkaan tarpeet, tarkkuus sekä oma-aloitteisuus ja hyvä paineensietokyky ovat ominaisuuksia joilla testaajana pärjää työelämässä. Kouluissa tulisi panostaa enemmän testaamiseen opettamiseen eikä vaan

ohittaa tätä sivulauseena. Eri mahdollisuuksia voisivat olla valinnaiset kurssit tai jopa erikoistumislinja testaajaksi tai laadunkehitykseen.

Vastaajat näkivät automaatiotestaamisen hyvänä asiana, koska tämä vapauttaa rutiineista, joita manuaalisella testauksella voi olla. Automaatiotestaaminen toimii hyvin regressiotestauksessa sekä hyvin suunniteltuna ja toteutettuna vähentää manuaalisen regressiotestauksen tarvetta ja mahdollistaa testauksen maksimoinnin. Vastaajien mielestä manuaalista testausta tarvitaan myös tulevaisuudessa. Ihmissilmää, aivoja ja kykyä ajatella ei voida automatisoida. Toimintatapojen matkiminen automaatiolla ei myöskään ole tehokasta. Manuaalinen testaus tulee suunnitella niin, että se on taloudellisesti järkevää. Automatisoidun ja manuaalisen testin käytön määrä nähtiin tasavertaisena. Rutiininomaiset tapaukset tulee automatisoida, mutta tapaukset, jotka testataan laajemmalla kaavalla on hyvä suorittaa manuaalisena.

7.2 Itsearviointi

Opinnäytetyön tekeminen aloitettiin elokuussa 2017 sekä päätettiin saman vuoden marraskuussa. Opinnäytetyö eteni tasaisesti hyvässä vauhdissa sekä olen mielestäni nähnyt hyvin vaivaa opinnäytetyön tekoa varten, eikä opinnäytetyön tekeminen pysähtynyt missään vaiheessa. Ainoat esteet tekemiselle olivat työkiireet sekä harrastuksien kiireet, mitä minulla ajoittain oli. Aihe on minua kiinnostava sekä kyselytutkimukseen osallistujia kiinnosti aihe suuresti, koska keneltäkään ei saatu kieltävää vastausta kyselyyn osallistumiseen. Kysymykset saivat vastaajilta kehuja ja he huomasivat, että kysymyksissä oli aiheita, joita hekin käyvät läpi työelämässä. Opinnäytetyötä voivat tulevaisuudessa hyödyntää henkilöt, jotka ovat kiinnostuneet testaamisesta, mutta kenellä ei ole tietoa mitä testaaminen on. Opinnäytetyössä käytiin kattavasti läpi aiheita, joita testaaja kohtaa työelämässä.

Automaatiotestauksessa käytiin läpi tämän perustietoja, jotka on hyvä sisäistää automaatiotestausta opetellessa. Jos automaatiotestaus kiinnostaa harjoittelu tulisi aloittaa Robot Frameworkilla tai Seleniumilla. Seleniumia ei työelämässä käytetä enää paljoa, mutta automaatiotestauksen harjoitteluun tämä on oiva sovellus. Jos tarkoitus on harjoitella automaatiotestausta työelämään, Robot Framework soveltuu tähän parhaiten.

7.3 Mikä onnistui?

Aiheita on kerätty kattavasti koskemaan testaamista. Opinnäytetyössä käytiin muitakin aiheita läpi regressiotestauksen lisäksi ja avattiin muita testauksen mahdollisuuksia. Automaatiotestauksessa testitapauksien suorittaminen oli uusi asia, josta on varmasti minulle hyötyä tulevaisuudessa. Sovellukset, joilla suoritettiin automaatiotestausta olivat tunnetuimpia sovelluksia, joten näiden läpikäynti oli hyvä asia.

Käytin työssäni paljon eri lähteitä ja näitä on hyödynnetty laajasti. Testausmenetelmistä kerrottiin laajasti sekä näiden tarkoitus käytiin kattavasti läpi, varsinkin Black–box ja White–box testausmenetelmissä. Kyselytutkimukseen saatiin niin paljon vastauksia, kuin opinnäytetyöhön oli tavoitteena saada sekä vastaukset olivat hyvin kattavia ja pitivät sisällään paljon tärkeää tietoa.

7.4 Mikä olisi voinut mennä paremmin?

Testitapauksia olisi voinut olla muitakin, kuin pelkästään verkkosivujen toiminnan testaamista manuaalisessa sekä automaatiotestauksessa. Vaihtoehtona olisi voinut olla esimerkiksi jonkin mobiilisovelluksen testaaminen. Pilvipalveluiden sekä tehtävienhallintaohjelmistojen roolia testaamisessa olisi voinut käydä kattavammin läpi, koska testaajilla on työssään näitä käytössä kohtalaisen paljon. Näistä läpi käytäviä sovelluksia olisivat voineet olla esimerkiksi pilvipalvelu Microsoft Visual Studio Team Services tai tehtävienhallintaohjelma Jira. Manuaalisesta testaamisesta olisi voinut kerätä enemmän materiaalia sekä selittää aiheesta muutakin testitapauksen suorittamisen lisäksi. Myös Scrumin roolia testaamisessa olisi voitu käydä läpi sekä miten tätä testaamisessa hyödynnetään työelämässä.

Jenkinsillä olisi voinut myös toteuttaa toisen testitapauksen eikä vain yhtä testitapausta, koska kyseessä on tärkeä automaatiotestauksen sovellus. Tähän syynä oli minun muut kiireet sekä aika opinnäytetyön kanssa oli loppuvaiheessa käymässä vähiin.

8 Lähteet

Berglund, M. 11.5.2015. Miten käyttöliittymä testataan automaattisesti. Luettavissa: http://www.develore.com/artikkeli/miten-kayttoliittyma-testataan-automaattisesti/. Luettu 15.8.2017

Berglund, M. 9.4.2015. Ohjelmistojen testaus on vakaan vauhdin tae. Luettavissa: http://www.develore.com/artikkeli/ohjelmistojen-testaus-vakaan-vauhdin-tae/. Luettu 15.8.2017

DAutoBotsTutorials. 19.10.2016. Robot Framework Installation. Katsottavissa: https://www.youtube.com/watch?v=932H4Asd3hU. Katsottu 10.9.2017

Duncan, N. 16.7.2012. Test Automation Basics – Levels, Pyramids & Quadrants. Luettavissa: http://www.duncannisbet.co.uk/test-automation-basics-levels-pyramids-quadrants. Luettu 17.10.2017

Execute Automation Youtube–kanava. 11.2.2017. Katsottavissa: https://www.youtube.com/channel/UCO1aucBAJgFR8odzfXOZ5uw. Katsottu 22.9.2017

Guru99. 2.1.2013. What is Black-box testing. Katsottavissa: https://www.youtube.com/watch?v=Wi75S5TTfQ0. Katsottu 21.8.2017

Guru99. 24.2.2013. What is White-Box testing. Katsottavissa: https://www.youtube.com/watch?v=3bJcvBLJViQ. Katsottu 28.8.2017

Holopainen, J. 29.4.2005. Regressiotestaus ja testien valintatekniikat. Luettavissa: http://www.cs.uku.fi/tutkimus/sose/material/Regressiotestaus.pdf. Luettu 8.8.2017

Immonen, J. 18.3.2003, Kansainvälisen viestinnän laitos, Luentomoniste. Luettavissa: http://cs.joensuu.fi/~jimmonen/jot_moniste/jot_moniste_121.html. Luettu 2.8.2017

Kasurinen, J–P. 2013, Ohjelmistotestauksen käsikirja. Docendo. Helsinki.

Korpimies, A. 3.5.2012. Näitä ominaisuuksia ohjelmistotestaaja tarvitsee. Luettavissa: http://www.tivi.fi/Arkisto/2012-05-03/N%C3%A4it%C3%A4-ominaisuuksia-ohjelmistotestaaja-tarvitsee-3191863.html. Luettu 8.8.2017

Lahtinen, S. 4.2.2016. Yksikkötestaus. Luettavissa: http://myy.haaga-helia.fi/~swd1tn002/tietokantamateriaali/vko14/yksikkoTestaus.pdf. Luettu 6.8.2017

Madsen, C & T. 2016. Grey Box. Luettavissa: http://energy.imm.dtu.dk/models/grey-box.html. Luettu 29.10.2017

Mäkelä, S. 26.11.2000. Testaustyökalut. Luettavissa: https://www.cs.helsinki.fi/u/laine/otv/testaustyokalut.pdf. Luettu 5.8.2017

Nenonen, M. 2017. Testiautomaation parhaat työkalut ja käytännöt. Luettavissa: https://www.theseus.fi/bitstream/handle/10024/130574/Nenonen_Marko.pdf?sequence=1&isAllowed=y. Luettu 29.10.2017

Onur, M. 25.4.2016. Getting Started with RobotFramework on Windows. Luettavissa: http://www.swtestacademy.com/getting-started-robotframework/. Luettu 10.9.2017

Oulun seudun ammattiopisto. 23.2.2006. Kehittämistyön vaiheet ja elinkaarimallit. Luettavissa: http://www.okol.org/verkkokurssit/datanomi/tietojarjestelmien_kaytto_ja_kehittaminen/johdatus_tietojarjestelmiin/kehittamistyon_vaiheet_ja_elikaarimallit/kehittamistyon_vaiheet_ja_elinkaarimallit_asia.htm. Luettu 3.8.2017

Pohjolainen, P. 12.12.2003. Ohjelmiston testauksen automatisointi. Luettavissa: http://cs.uef.fi/uku/tutkimus/Teho/PenttiPohjolainen_Gradu.pdf. Luettu 11.10.2017

Poranen, T. 2.8.2011. Testauksesta ja testaussuunnitelmasta. Luettavissa: http://www.sis.uta.fi/~tp54752/projektikurssit/2004_5/luennot/testaus.pdf. Luettu 4.8.2017

Pyhäjärvi, M. 27.10.2006. Ohjelmistojen testaus. Luettavissa: http://users.jyu.fi/~kolli/testaus2006/materiaali/Maaret_27102006.pdf. Luettu 9.8.2017

Robot Framework. 1.9.2017. Selenium2Library. Luettavissa: http://robotframework.org/Selenium2Library/Selenium2Library.html. Luettu 14.9.2017

Savolainen, O. 3.3.2005. Ohjelmistotestaus: Testausprosessin luonti ja kehittäminen. Luettavissa: http://users.jyu.fi/~jorma/kandi/2005/Kandi_OSavolainen.pdf. Luettu 13.8.2017

Software Testing Material. 19.5.2016. How to Download and Install Selenium WebDriver. Katsottavissa: https://www.youtube.com/watch?v=6mvCDlwPFiE&t=272s. Katsottu 5.9.2017

Testauksen osaamisyhteisö (TestausOSY). 2012. Laatu ja testaus. Automatisointi. Luettavissa: http://testausosy.fi/wp-content/uploads/2012/11/LT-Vol1Ed2.pdf. Luettu 29.10.2017

Tuovinen, A-P. 14.3.2013. Ohjelmistotestauksen perusteita II. Luettavissa: https://www.cs.helsinki.fi/u/aptuovin/testaus/Ohj_testaus_2013_2.pdf. Luettu 5.8.2017

Usenius, T. 10.11.2009. Virheiden etsintä: katselmoinnit tai testaaminen? Luettavissa: https://www.cs.helsinki.fi/u/paakki/Semis09-Usenius.pdf. Luettu 9.8.2017

Väyrynen, J. 13.4.2014. Ohjelmistotestauksen käytännöt ja ongelmat–katsaus kyselytutkimuksista. Luettavissa: http://mikamantyla.eu/V4_Vayrynen_Joni.pdf. Luettu 13.8.2017

W3Schools. 8.4.2014. XPath Tutorial. Luettavissa: https://www.w3schools.com/xml/xpath_intro.asp. Luettu 29.10.2017

White-box Testing Fundamentals. 25.2.2017. White Box Testing. Luettavissa: http://softwaretestingfundamentals.com/white-box-testing/. Luettu 28.8.2017