TAMPEREEN
AMMATTIKORKEAKOULU

TESTIAUTOMAATIO OSANA JATKUVAA OHJELMISTOKEHITYSTÄ

Eerik Timonen

Opinnäytetyö
Huhtikuu 2019
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

TIMONEN EERIK:
Testiautomaatio osana jatkuvaa ohjelmistokehitystä

Opinnäytetyö 31 sivua
Huhtikuu 2019

Opinnäytetyön tavoitteena oli aloittaa automaattinen hyväksymistestaus toimeksiantajan web-ohjelmistoon, jonka arvioitiin vähentävän manuaaliseen testaamiseen kuluvaa aikaa. Toimeksiantaja opinnäytetyössä oli Black Woodpecker Software ja tuote, johon testejä tehtiin, on heidän tekemä sisäpiiriluetteloiden hallinnointityökalu. Testien tekemiseen käytettiin Robot Framework -kehystä ja siihen saatavaa Selenium2Library -kirjastoa, jolla voidaan testata web-käyttöliittymiä.

Opinnäytetyössä käytiin läpi Robot Frameworkin asentaminen ja miten Selenium2Library saadaan otettua käyttöön testeissä. Opinnäytetyössä selvitettiin, miten Robot Framework toimii ja mitä testien tekemiseen vaaditaan. Siinä tutkittiin myös minkälaisia testien tulisi olla ja miten tehdään hyviä ja ymmärrettäviä testejä. Opinnäytetyössä käytiin läpi, mitä testitapauksia ohjelmiston testaukseen tarvitaan ja miten testit saadaan Dockerin avulla Jenkins -palveluun pyörimään ja siten mukaan jatkuvaan ohjelmistokehitykseen.

Testaus on tärkeä osa ohjelmistokehitystä ja testaustapoja on monenlaisia. Opinnäytetyössä tutustuttiin ohjelmistokehityksen V-malliin ja perehdyttiin tarkemmin siihen kuuluvan hyväksymistestauksen tarkoitukseen ja tavoitteisiin.

Asiasanat: testaus, robot framework, automaatio, ohjelmistokehitys

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Software Production

TIMONEN EERIK:
Test Automation along with Continuous Software Development

Bachelor's thesis 31 pages
April 2019

The objective of the thesis was to implement acceptance tests for one of the client's web-applications. The tests were estimated to reduce the time that is needed to carry out manual testing. The client of the thesis was Black Woodpecker Software and the product where the tests were applied was their inner circle list management tool. The tests were made by using the Robot Framework and Selenium2Library which is used to test web-interfaces.

In the thesis it is explained how to install Robot Framework and how to get Selenium2library to work with it. Research was done on how Robot Framework works and what is needed for developing the tests. The thesis also focuses on how the test should look like and how to make good and understandable tests. The test cases, which were needed for the tests of the project, are covered. Research was also done on how to get the test working on Jenkins using Docker and, in that way, how to include them in continuous software development.

Testing is an important part of software development and there are many ways to test applications. The thesis introduces software development with V-model and takes a closer look at why acceptance testing is important and what its goals are.

Key words: testing, robot framework, automation, software development

SISÄLLYS

TERMIT

Robot Framework	Avoimen lähdekoodin testiautomaatio ja hyväksymistestaus -kehys
Python	Ohjelmointikieli
IronPython	C#-kielinen toteutus Pythonista
Jython	Java-kielinen toteutus Pythonista
Pip	Pythonin pakettien hallinnointi sovellus
Selenium2Library	Robot Framework kirjasto, jolla testataan nettisivuja
Jenkins	Avoimen lähdekoodin automaatio serveri
Docker	Ohjelmistojen ajamiseen virtuaaliympäristössä tarkoitettu ohjelma

1 JOHDANTO

Nykypäivänä ohjelmistoja julkaistaan todella paljon ja koodia syntyy miljoonia rivejä. Koodia syntyy niin paljon, että virheiltä ja bugeilta ei voida välttyä ja virheitä sattuu kaiken tasoisille koodareille. Niitä voidaan ehkäistä testaamalla koodia ja ohjelmistoa, jotta vältyttäisiin isompien bugien pääsemiseltä tuotantoon. Ohjelmistoissa ja koodissa kaiken testaaminen vie todella paljon työtunteja ja ei ole ollenkaan realistista edes suunnitella kaikkien mahdollisten tapausten testaamista.

Opinnäytetyön toimeksiantaja on Black Woodpecker Software Oy, lyhyemmin BWS. BWS on vuonna 2015 perustettu startup-yritys, joka toimii Tampereella ja Helsingissä. Heillä on tällä hetkellä kolme tuotetta, jotka ovat Sidonnaisuusrekisteri, Timber ja Ticker. Opinnäytetyössä on tavoitteena ottaa Ticker-palvelussa käyttöön Robot Framework -kehys, joka on tarkoitettu hyväksymistestauksen automatisointiin. Tarkoitus on ottaa Robot Framework mukaan kehitysprosessiin ja sen avulla vähentää ohjelmiston käyttöliittymän ja ominaisuuksien testaukseen kuluvaa aikaa.

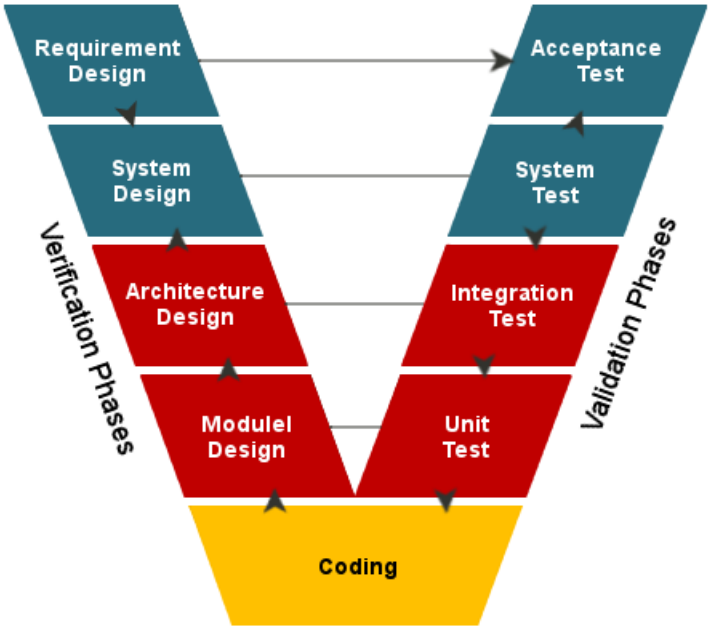
Opinnäytetyössä käydään läpi pikaisesti Robot Frameworkin asennus ja selvitetään, miten tehdään mahdollisimman ymmärrettäviä ja käyttökelpoisia testejä web-käyttöliittymiä varten. Siinä selvitetään myös mikä Robot Framework on ja mitä testien tekemiseen tarvitaan. Näiden lisäksi työssä käydään läpi testitapaukset, joita Tickerin testaukseen vaaditaan, sekä miten testit saadaan mukaan ohjelmiston jatkuvaan kehitykseen.

2 OHJELMISTON KEHITYSSYKLI JA HYVÄKSYMISTESTAUS

2.1 Ohjelmiston kehityssykli

Ohjelmistojen kehityssykliin on kehitetty monta erilaista mallia, joiden avulla saadaan suunniteltua, kehitettyä ja testattua korkealaatuisia ohjelmistoja. Ne pyrkivät tuottamaan ohjelmistoja, jotka täyttävät asiakkaan odotukset aika- ja kustannusarvioiden puitteissa. Mallit koostuvat yleensä vaatimusten analysoinnista ja niiden rajaamisesta, ohjelmiston suunnittelusta ja sen toteutuksesta, testauksesta, julkaisemisesta ja ohjelmiston ylläpidosta. (Tutorialspoint.)

Yksi näistä malleista on V-malli, joka on laajennus vesiputousmallista (kuva 1). Se koostuu verifiointi- ja validointivaiheesta, jotka yhdistää koodausvaihe. Verifiointivaiheessa, eli todentamisvaiheessa, on neljä vaihetta: vaatimusten määrittely, järjestelmän suunnittelu, arkkitehtoninen suunnittelu ja modulaarinen suunnittelu. Vaatimusten määrittelyssä asiakkaalta kerätään tuotteeseen liittyvät odotukset ja tarkat vaatimukset. Järjestelmän suunnittelussa käydään läpi koko järjestelmässä käytettävät laitteisto- ja kommunikointijärjestelyt. Arkkitehtonisessa suunnittelussa päätetään, mitä teknillisiä menetelmiä käytetään. Modulaarisessa suunnittelussa puolestaan käydään läpi, mitä järjestelmän eri moduuleissa tarvitaan. (Tutorialspoint.)



KUVA 1. Ohjelmistokehityksen V-malli (ProfessionalQA 2019)

Todentamisvaiheen jälkeen tulee koodausvaihe, jossa tehdään itse tuote. Tämän jälkeen alkaa validointivaihe, eli kelpuutusvaihe, jossa on myös neljä vaihetta. Ne ovat yksikkö- (unit), integraatio- (integration), järjestelmä- (system) ja hyväksymistestaus (acceptance). Yksikkötestaus on kooditasolla tapahtuvaa testausta, jolla ehkäistään bugien syntymistä. Integraatiotestauksella testataan järjestelmän moduulien yhteiseloa ja keskustelua keskenään. Järjestelmän testauksessa taas testataan koko järjestelmän toimivuutta ulkopuolisten järjestelmien kanssa. Lopuksi tapahtuvassa hyväksymistestauksessa testataan, että ohjelmisto pystyy toteuttamaan sille annetut vaatimukset. (Tutorialspoint.) Opinnäytetyössä keskitytään ohjelmiston hyväksymistestaukseen.

2.2 Hyväksymistestaus

Ohjelmistoja voidaan testata monella tavalla ja testaamiseen kuluu erittäin paljon aikaa. Kysymys kuuluukin miksi hyväksymistestaus, joka vie paljon aikaa itse ohjelmiston kehityksestä, pitäisi ottaa mukaan ohjelmistokehityssykliin mukaan? Hambling, B. ja Van Goethem, P. (2013, 18) kertovat neljä syytä, jotka ovat siteerattu merkittäviksi syiksi, joiden takia järjestelmät epäonnistuvat:

1. Asiakkaan on usein vaikea kuvitella, mitkä heidän vaatimuksensa ohjelmiston suhteen ovat. Ajan ja kehityksen myötä ideat siitä, minkälaiseksi ohjelmistosta halutaan tulevan, muuttuvat.
2. Vaikka asiakkaalle vaatimukset ovat selkeät, saattaa heille olla vaikeaa kommunikoida ne kehitystiimille. Vaatimukset, jotka on kirjattu ylös, saattavat olla jotain muuta kuin mitä oli tarkoitettu.
3. Ohjelmiston kehittäjät saattavat tulkita vaatimuksia väärin, koska he tuovat omia oletuksia mukaan kehitykseen.
4. Vaikka vaatimukset olisi kuviteltu, kommunikoitu, kirjoitettu ja ymmärretty oikein, kehittäjät saattavat tehdä inhimillisiä virheitä.

Yksikkötestauksessa testataan, että ohjelmistoon tehty ominaisuus toimii teknillisesti oikein, kun taas hyväksymistestauksessa testataan, että ominaisuus toimii asiakkaan haluamalla tavalla. Hyväksymistestauksessa yksinkertaisesti katsotaan, että ohjelmistossa pystytään tekemään siltä vaadittuja asioita, mutta sillä on kuitenkin monia muitakin tavoit-

teita. Sen avulla voidaan varmistaa uusia ominaisuuksia lisätessä, että vanhoja ominaisuuksia ei ole rikottu tai muutettu. Virheet ja bugit tulevat esiin niin, kuin ne olisivat tulleet käyttäjille. Ominaisuuksista, joita testataan, syntyy kuvaus, jota normaalit ihmiset pystyvät lukemaan ilman teknillistä osaamista. (Sale, D. 2014, 89.)

Hyväksymistestausta voidaan tehdä käsin käymällä läpi ohjelmiston ominaisuuksia tai se voidaan automatisoida käyttämällä erilaisia työkaluja. Opinnäytetyössä automatisoidaan hyväksymistestaus käyttäen Robot Framework automatisointikehystä.

3 ROBOT FRAMEWORK

3.1 Tausta

Robot Framework on vuonna 2005 alkunsa saanut geneerinen automaatiotestaus -kehys, jota on tarkoitettu käytettävän hyväksymistestauksessa ja hyväksymistestausvetoisessa kehityksessä. Se on alkujaan Pekka Klärckin diplomityöstä syntynyt projekti, joka kehitettiin Nokia Networksissa. (Bisht, S. 2013, 26) Nykyään sen toinen versio on Apache lisenssi 2.0 alla julkaistu avoimen lähdekoodin projekti, jota Robot Framework Foundation sponsoroi. Robot Framework -projekti löytyy Githubista. (Robot Framework, 2019). Robot Frameworkin käytöstä ja testien tekemisestä löytyy kattavat dokumentaatiot, joiden avulla sen opetteleminen ja käyttäminen on helppoa. Sen aktiivinen ohjelmistokehitys ja yhteisö auttavat ongelmatilanteissa.

Robot Framework käyttää testeissään helppoa lähestymistapaa, joka perustuu avainsanavetoiseen testaukseen. Se on tehty Pythonilla, mutta sitä voidaan käyttää myös Jythonilla sekä IronPythonilla. (Robot Framework, 2019). Sitä käytetään eniten Selenium-kehyksen kanssa nettisivujen automaatiotestaukseen, mutta sitä voidaan hyödyntää myös moneen muuhun tarkoitukseen. (Bisht, S. 2013, 26). Sen käyttämiseen ei periaatteessa tarvitse aikaisempaa ohjelmointiosaamista, mutta siitä on tietysti hyötyä myös vaativampia testejä tehtäessä.

3.2 Asentaminen

Robot Frameworkin asentaminen on helppoa. Se tarvitsee toimiakseen Pythonia. Robot Framework 3.0 tukee Python-versioita 2.6, 2.7, 3.3 ja uudempia (Installation instructions 2018). Windowsilla Pythonin polku (path) pitää lisätä ympäristömuuttujiin, jotta sen toimintoja voidaan käyttää komentoriviltä. Nykyään Pythonin asennusohjelmassa on mahdollisuus lisätä polku suoraan Windowsin ympäristömuuttujiin (Installation instructions 2018). Python on sen sijaan valmiiksi asennettuna uusimmissa Mac-käyttöjärjestelmissä. Esimerkiksi 2012 julkaistussa Mac versiossa 10.8 tulee Python 2.7 mukana (Python setup and usage 2019).

Itse Robot Framework asennetaan käyttäen Pip-pakettienhallintatyökalua. Pip tulee 2.7.9, 3.4 ja uudempien Python versioiden mukana, joten sitä ei pitäisi tarvita asentaa erikseen (Pip Installation, 2019). Komentoriville kirjoittamalla *pip* voidaan tarkistaa, onko se asennettuna. Asennettuna se antaa listan toiminnoista, joita sen kanssa voidaan käyttää. Robot Framework asennetaan komennolla *pip install robotframework* (Install instructions, 2018). Asennuksen onnistumisen voidaan tarkistaa komennolla *robot –version*, joka näyttää koneelle asennetun Robot Frameworkin version.

3.3 Testien rakenne

Testit tehdään testitapaustiedostoihin (test suite), joissa voidaan käyttää muutamaa eri tiedostotyyppiä. Ne ovat plain text, HTML, tab-separated values (TSV) ja reStructuredText (reST). Näistä vaihtoehdoista suositellaan käyttämään plain text -tiedostotyyppiä. Muita tyyppejä käytetään, jos testien tekemisessä on erityistarpeita ja niihin tarvitaan näiden muiden tiedostotyyppien ominaisuuksia. (Robot Framework User Guide, 2018). Opinnäytetyössä käytetään plain text -tyyppiä, koska erityistarpeita ei ole ja se on suositeltu tyyppi.

Robot Framework -testit koostuvat muuttujista (variable), avainsanoista (keyword) ja testitapauksista (test case). Tiedostoissa niillä jokaisella on oma taulunsa, joiden nimet ovat Variables, Keywords ja Test Cases. Taulut merkitään tiedostoissa kuuden asteriskin väliin. Esimerkiksi muuttujataulu merkitään tiedostossa *** Variables ***. Näiden kolmen lisäksi on vielä asetukset (settings), jossa importoidaan testeissä käytetyt kirjastot ja tehdään testien alustaminen sekä alasajo.

3.3.1 Muuttujat

Robot Frameworkilla on neljä muuttujatyyppiä ja ne ovat olennaisia osia testejä tehtäessä. Muuttujatyypit ovat skalaarinen muuttuja, lista-, sanakirja- ja ympäristömuuttuja. Muuttujia käytetään yleensä avainsanojen tai avainsanapöytien argumentteina. Niitä pystytään käyttämään myös kaikkien asetuksien arvoina. Muuttujia ei voida kuitenkaan käyttää normaalien avainsanojen niminä. (Robot Framework User Guide 2018)

Skalaarinen muuttuja on yleisimmin käytetty muuttujatyyppi, jota käytetään. Sen syntaksi on *${NAME}*. Skalaarinen muuttuja on yleensä merkkijono, mutta se voi olla myös numero, objekti tai lista. Listamuuttujan *@{LIST_NAME}* tulee olla Python-lista tai listan kaltainen objekti. Merkkijonot eivät käy listoista Robot Frameworkissa. Sanakirja *&{DICT_NAME}* koostuu listasta, joka sisältää avain–arvo-pareja. Se vaatii Python sanakirjan tai sanakirjan kaltaisen objektin. Ympäristömuuttujat *%{ENV_NAME}* voivat olla vain merkkijonoja. Ne ovat globaaleja muuttujia, joten yhdessä testitapauksessa määritettyä ympäristömuuttujaa voidaan käyttää myös muissa testitapauksissa. (Robot Framework User Guide 2018). Siihen tehdyt muutokset eivät kuitenkaan säily muissa testitapauksissa. Muuttujatyyppien syntaksissa muuttuu vain etuliite.

3.3.2 Avainsanat

Robot Framework testeissä käytetään avainsanavetoista tyyliä. Valmiita avainsanoja löytyy Robot Frameworkiin tehdyistä avainsanakirjastoista. Niitä voidaan käyttää importoimalla kirjasto tiedoston asetuspöydässä. Avainsanat ovat niitä, jotka tekevät testeissä asioita. Esimerkiksi Selenium2Library -avainsanakirjastosta löytyvä *Go To* avainsana menee selaimessa sille argumenttina annettuun osoitteeseen. Avainsanat ottavat usein muuttujia argumentteina vastaan. Tiedostoissa avainsanatauluihin tulee itse tehtyjä korkean tason avainsanoja (user keywords). Ne käyttävät alemman tason avainsanoja kirjastoista tai toisia itse tehtyjä avainsanoja. (Robot Framework User Guide, 2018).

Testitapaustiedoston avainsanapöydän avainsanoja voidaan käyttää vain siinä tiedostossa, jossa ne on tehty. Omia avainsanoja voidaan tehdä myös resurssitiedostoissa (resource file) testitapaustiedostojen sijaan. Niitä avainsanoja voidaan käyttää tiedostoissa, joihin resurssitiedosto on importoitu. Resurssitiedostot eroavat testitapaustiedostoista siten, että niissä ei voi olla testitapauksia ollenkaan. Asetukset, joita niissä voidaan käyttää, ovat myös rajoitettu vain kirjastojen ja muiden resurssitiedostojen importointiin. Muuttuja- sekä avainsanapöydät toimivat samalla tavalla kuten testitapaustiedostoissa. (Robot Framework User Guide, 2018).

3.3.3 Kirjastot

Valmiita avainsanakirjastoja on paljon ja niitä voidaan hyödyntää Robot Framework testejä tehtäessä. Robot Frameworkin mukana tulee valmiiksi kymmenen kirjastoa. Niiden

nimet ovat BuiltIn, Collections, DateTime, Dialogs, OperatingSystems, Process, Screenshot, String, Telnet ja XML. Niistä BuiltIn, joka sisältää usein tarvittuja geneerisiä avainsanoja, on aina käytössä ja sitä ei tarvitse erikseen importoida tiedostoihin. (Robot Framework User Guide, 2018.) Myös muita ulkopuolisia kirjastoja on kehitetty paljon eri tarkoituksiin. Julkisista ulkopuolisista kirjastoista löytyy lista Robot Frameworkin sivuilta (Robot Framework User Guide, 2018).

Kirjastoista useimmat asennetaan käyttäen komentoa *pip install kirjaston nimi*. Joillakin ulkopuolisilla kirjastoilla voi olla kuitenkin täysin erilainen asennusmenetelmä. Ne saattavat tarvita myös joitain muita asennusvaatimuksia toimiakseen. (Robot Framework User Guide, 2018.) Esimerkiksi Seleniun2Libraryn kanssa täytyy asentaa selainten, joita testeissä käytetään, erilliset ajurit. Asentamisen jälkeen kirjaston avainsanoja voidaan käyttää testitapaustiedostoissa importoimalla kirjasto tiedoston asetuspöydässä. Omia kirjastoja voidaan myös tehdä Pythonilla tai Javalla.

Opinnäytetyön kannalta yksi tärkeimmistä kirjastoista on nettisivujen testaamiseen tarkoitettu kirjasto Selenium2Library. Se käyttää Selenium WebDriver moduuleja kontrolloidakseen selaimia. Sen avainsanojen avulla voidaan olla vuorovaikutuksessa nettisivujen elementteihin. Avainsanat, jotka tekevät elementtien kanssa asioita, ottavat argumenttina vastaan elementin paikantimen, eli elementin id:n, nimen tai Xpath-viittauksen (kuva 2). Niiden avulla avainsana tietää, mitä elementtiä etsitään. (Robot Framework SeleniumLibrary, 2019.) Tämän takia on tärkeä pitää huolta siitä, että samalla sivulla olevien elementtien nimet ja id:t ovat uniikkeja. Varsinkin id:n tulee olla aina uniikki, sillä useat samat id:t aiheuttaa ongelmia ohjelmoidessa.

```
 9      Click Element  id
10      Click Button  name
11      Click Element  //*[contains(text(),"example")]
```

KUVA 2. Selenium avainsanan paikantaja-argumentit.

3.3.4 Testitapaukset

Testitapaukset ovat syntaksiltaan avainsanojen kaltaisia ja ne eroavat avainsanoista oikeastaan vain siinä, että ilman niitä testejä ei voida ajaa. Testitapaustiedostoissa ne tulevat

testitapauspöytään. Testitapauksissa voidaan käyttää omia tai kirjastoista otettuja avainsanoja, mutta omia avainsanoja suositellaan käytettävän niissä. (Robot Framework User Guide, 2018.) Testitapauksien nimien tulisi kertoa suunnilleen, mitä siinä olevat avainsanat tekevät.

Testitapauksissa voidaan käyttää myös omia asetuksia (kuva 3). Asetuksia, joita voidaan käyttää testitapauksissa ovat Documentation, Tags, Setup, Teardown, Template ja Timeout. Asetukset merkitään testitapauksissa hakasulkeiden sisään. (Robot Framework User Guide, 2018.)

```
8   *** Test cases ***
9   Test With Settings
10      [Documentation]  This is a test with settings
11      [Tags]   example  test  settings
12      Log  Hello!
```

KUVA 3. Testitapaus asetuksien kanssa

3.4 Testien kirjoittaminen

Yksinkertaisen testin saa hyvin nopeasti ja helposti tehtyä. Testin voi tehdä yhteen tiedostoon, kuten esimerkiksi esimerkkitestissä asetukset, muuttujat, avainsanat ja testitapaukset ovat yhdessä *example.robot* -tiedostossa (kuva 4). Esimerkkitesti avaa selaimen ja menee haluttuun osoitteeseen ja sulkee lopuksi selaimen.

Testin asetuksissa on importoitu Selenium2Library -kirjasto, jotta sen web-testaukseen tarkoitettuja avainsanoja voidaan käyttää testeissä. Asetuksien dokumentaatiokohta ei ole pakollinen, mutta siihen voidaan kirjoittaa testitapauksista selosteen, joka sitten näkyy komentorivillä ja testien raporteissa. Esimerkkiin on tehty kaksi muuttujaa: browser ja url, joita testitapauskohdassa oleva *Open Browser* -avainsana käyttää argumentteinaan. Yksinkertaiseen testiin ei tarvitse periaatteessa luoda omia avainsanoja, sillä testit toimivat pelkästään testitapauksilla. Luvussa Hyvä testirakenne käsitellään yksityiskohtaisemmin, miksi ei ole hyvä tapa olla tekemättä omia avainsanoja testeihin.

Esimerkissä näkyy hyvin plain textin syntaksi. Toimintojen erottajana käytetään kahta tai useampaa välilyöntiä. Eli esimerkiksi muuttujien nimien ja niiden arvojen välillä on kaksi

välilyöntiä. Tämä sääntö pätee kaikkialla. *Open Browser* -avainsanan ja ensimmäisen muuttujan välissä on kaksi välilyöntiä sekä ensimmäisen ja toisen muuttujan välissä on kaksi välilyöntiä.

example.robot

```
*** Settings ***
Documentation  These are example tests
Library  Selenium2Library

*** Variables ***
${browser}  chrome
${url}  http://www.google.com

*** Keywords ***

*** Test cases ***
Open Browser And Go To Google
    Open Browser  ${url}  ${browser}
    Close All Browsers
```

KUVA 4. Esimerkkitesti

Testin voi ajaa komentorivillä komennolla *robot example.robot*. Testitapaukset ajetaan läpi ja komentoriville ilmestyy pieni raportti testeistä (kuva 5). Siinä näkyy, mitkä testitapaukset ovat menneet läpi ja mitkä ovat epäonnistuneet. Epäonnistuneissa testitapauksissa näkyy myös syy, miksi testi ei ole toiminut. Raportissa näkyy myös, mihin paikkaan tarkemmat raportit testeistä ovat menneet.

```
C:\Users\Erqq\Desktop\example>robot example.robot
==============================================================================
Example
==============================================================================
Open Browser And Go To Google
DevTools listening on ws://127.0.0.1:57653/devtools/browser/606f4af6-c2de-4a69-
Open Browser And Go To Google                                         | PASS |
------------------------------------------------------------------------------
This Test Fails                                                       | FAIL |
Keyword 'Selenium2Library.Open Browser' expected 1 to 6 arguments, got 0.
------------------------------------------------------------------------------
Example                                                               | FAIL |
2 critical tests, 1 passed, 1 failed
2 tests total, 1 passed, 1 failed
==============================================================================
Output:  C:\Users\Erqq\Desktop\example\output.xml
Log:     C:\Users\Erqq\Desktop\example\log.html
Report:  C:\Users\Erqq\Desktop\example\report.html
```

KUVA 5. Testien raportti komentorivillä.

3.5 Hyvä testirakenne

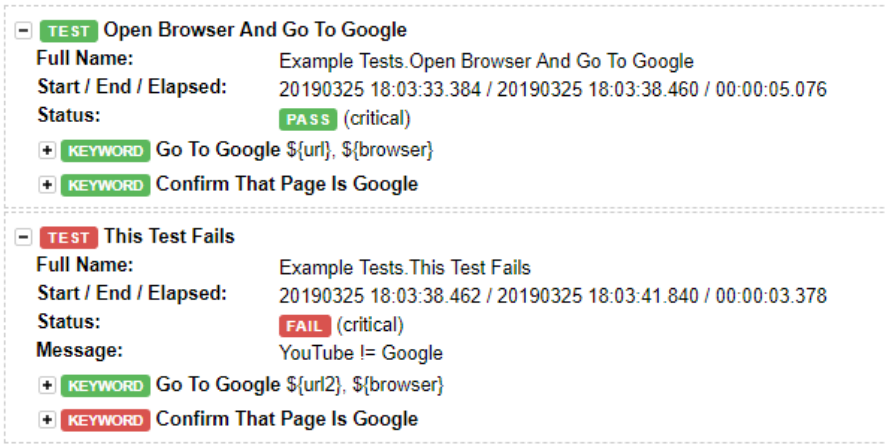
Testitapausten pitäminen mahdollisimman helposti ymmärrettävinä on tärkein ohjesääntö testien tekemiselle. Se helpottaa testien ylläpitoa, koska ei tarvitse arvailla, mitä kyseinen testi tekee. Hyvin ymmärrettäviä testejä edesauttaa hyvin nimetyt testitapaukset ja avainsanat sekä niiden rakenne.

Testitapausten nimien tulee kuvata, mitä siinä tapahtuu. Niiden nimet voivat olla aika pitkiä, jos on tarve, ja nimissä käytettävien sanojen ensimmäiset kirjaimet tulee olla isoja. Avainsanojen nimien on hyvä olla myös kuvaavia ja niiden täytyy olla selkeitä. Niiden tulee kertoa, mitä kyseinen avainsana tekee, ei miten se tekee sille annetut tehtävät. Avainsanoilla ei ole tarkkaa määritelmää siitä, pitääkö niissä käytettävien sanojen olla isoilla alkukirjaimilla. Pitkissä lausemaisissa avainsanoissa on hyväksyttävää käyttää isoa alkukirjainta vain ensimmäisessä sanassa. (How to write good test cases 2018.) Opinnäytetyössä kaikissa avainsanojen sanoissa on kuitenkin käytetty isoja alkukirjaimia yhtenäisyyden vuoksi.

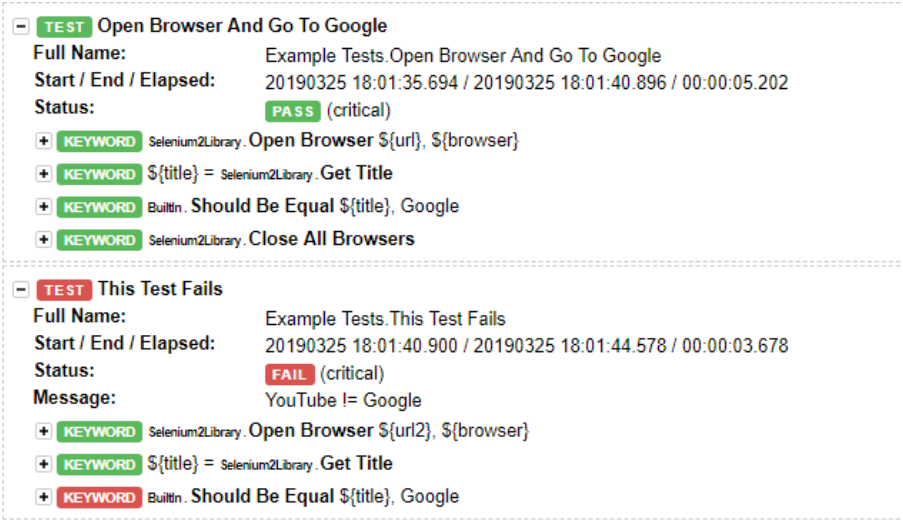
Testitiedostoilla on aika selkeä rakenne, jota niiden tulisi noudattaa. Tiedostossa olevien testien tulee liittyä toisiinsa ja yhdessä tiedostossa ei saa olla liian montaa testiä. Yhdessä testitiedostossa on hyvä olla maksimissaan kymmenen testiä, jotta niistä ei tule liian pitkiä ja siten epäselviä. Testien on hyvä olla itsenäisiä eli niiden ei pitäisi riippua toisistaan. Tätä ei kuitenkaan voida aina välttää ja toisistaan riippuvia testejä syntyy pakosti. Pitkiä ketjuja toisistaan riippuvista testeistä pitää kuitenkin välttää. (How to write good test cases 2018.)

Testeissä on hyvä välttää *Sleep*-avainsanan käyttöä, koska se on todella herkkä tapa synkronoida testeissä tapahtuvia asioita. Se on kuitenkin joskus helpoin tapa saada testi toimimaan, mutta sitä pitää käyttää silti varauksella. Parempi vaihtoehto *Sleep*-avainsanalle on avainsanat, jotka odottavat, kunnes jotain tapahtuu. (How to write good test cases 2018.) Esimerkiksi Selenium2Library- kirjaston avainsana *Wait Until Page Contains Element* odottaa, kunnes sivulle on lataantunut kyseinen elementti ja jatkaa vasta sitten seuraavaan avainsanaan. Kyseisiä *Wait*-avainsanoja löytyy moneen eri tarkoitukseen ja niillä voidaan varmistaa, että esimerkiksi jokin sivu tai elementti on lataantunut kokonaan.

Testitapauksien tulee testata vain yhtä asiaa kerrallaan ja niiden tulee olla helposti ymmärrettäviä (How to write good test cases 2018). Ymmärrettävyyteen auttaa, kun testitapauksissa käytetään vain itse tehtyjä korkean tason avainsanoja, joiden avulla pystytään kuvaamaan hyvin, mitä testitapauksessa tehdään. Se vaikuttaa myös testien analysointiin lokitiedostoissa ja raporteissa, jotka selkeentyvät huomattavasti, kun testitapauksissa käytetään omia avainsanoja. Omilla avainsanoilla testitapauksista tulee todella selkeitä ja avainsanat kuvaavat hyvin, mitä testeissä tapahtuu, sekä raporteista nähdään, missä yhteydessä testit epäonnistuvat (kuva 6). Jos taas testitapauksissa ei käytetä omia avainsanoja, tulee niistä erittäin epäselkeitä ja sen takia myös testitapaukset pidentyvät turhaan (kuva 7).



KUVA 6. Testitapaukset omilla avainsanoilla.



KUVA 7. Testitapaukset kirjastojen avainsanoilla.

4 TESTIT TUOTTEESSA

4.1 Johdanto

Startup-yrityksillä on usein kädet täynnä töitä ja tuotteita yritetään saada mahdollisimman nopeasti tuotantoon rajoitetun budjetin takia. Tuotteeseen saatetaan haluta jokin uusi ominaisuus nopeasti valmiiksi ja siinä vaiheessa Robot Frameworkin kaltaiset automaatiotestit saattavat tuntua turhilta. Päällepäin niiden tekeminen saattaa näyttää joillekin siltä, että kehittäjä käyttää kallisarvoista aikaa johonkin muuhun kuin uusien ominaisuuksien tekemiseen tai bugien korjaukseen. Tosiasiassa automaatiotestit auttavat ohjelmistokehitystä vähentämällä manuaaliseen testaamiseen menevää aikaa ja antamalla varmuutta tuotteelle. Manuaalisessa testaamisessa jää enemmän aikaa sellaisille asioille, joita automaatiotesteillä ei voida testata.

Tuote, johon Robot Framework automaatiotestejä opinnäytetyössä tehdään, on toimeksiantajan Ticker sovellus. Ticker on Pörssiyhtiöille suunnattu sisäpiiriluetteloiden hallinnointityökalu. Sisäpiiriluetteloiden ylläpitämiseen on paljon säännöksiä Suomen ja EU:n puolesta, joilla ehkäistään sisäpiiritietojen väärinkäyttöä. Palvelu mahdollistaa sisäpiiriluettelon ylläpitämisen Pörssin ja Finanssivalvonnan ohjeistuksien mukaisina ja pitää asiakkaan sisäpiiriluetteloiden tiedot helposti saatavilla yhdessä paikassa. Palvelussa asiakkaan pääkäyttäjät lisäävät yrityksen sisäpiiriin kuuluvat henkilöt sisäpiiriluetteloon. Palvelun kautta pystyy lähettämään sähköpostiviestejä luetteloon kuuluville henkilöille, sisäpiiritietoihin ja luetteloon liittyvistä asioista. Sisäpiiriläiset voivat käydä tarkistamassa tai täyttämässä henkilötietojaan sähköpostiin tulevalla linkillä.

4.2 Testi kohde

Tickerissä on kolme sisäpiiriluetteloversiota, joita asiakas käyttää tarpeidensa mukaan, eli johto ja johdon lähipiiri, taloudellista tietoa saavat henkilöt ja hankekohtaiset sisäpiiriluettelot. Johto ja johdon lähipiiri sisäpiiriluetteloon kerätään yhtiön johtohenkilöt ja heidän lähipiirinsä tiedot. Hankekohtaiseen luetteloon kerätään yhtiössä tapahtuvien yksittäisten projektien sisäpiiritiedon saaneet henkilöt. Taloudellista tietoa saaviin taas ke-

rätään henkilöt, joilla on säännöllinen pääsy yhtiön julkistamattomaan taloudelliseen tietoon. Opinnäytetyössä keskitytään tekemään automaatiotestejä hankekohtaisien sisäpiiriluetteloiden ominaisuuksiin.

Tickerin johdon sisäpiiriluettelossa ja hankekohtaisten sisäpiiriluetteloissa on samat perusominaisuudet pienillä vaihteluilla. Taloudellista tietoa saavien lista on riisuttu versio näistä kahdesta. Projektikohtaisessa luettelossa ominaisuudet ovat kaikista geneerisimmät, joten sen testit toimivat hyvänä pohjana muille luetteloille. Perusominaisuuksiin kuuluu henkilön lisääminen sisäpiiriluetteloon, sähköpostiviestin lähettäminen luetteloon kuuluville henkilöille, velvoitteet tekstin muokkaus ja listan aikaisemman tai nykytilanteen vienti Excel tiedostoon. Jokaisella listalla on myös tapahtuma- ja viestiloki. Hankekohtaisissa luetteloissa henkilöiden lisääminen eroaa johdon luettelosta vain tietojen määrällä, jota henkilöstä annetaan lisäämishetkellä. Viestien lähettämisessä vain viestipohjat eroavat listojen välillä.

4.3 Testitapaukset

Tickerissä on monta eri käyttäjätasoa, joilla on erilaisia oikeuksia ohjelmistossa. Tällä hetkellä testit on tehty käyttäen käyttäjää, jolla on eniten oikeuksia ohjelmistoon asiakkaan näkökulmasta. Muitakin käyttäjiä eri oikeuksilla voidaan tietysti kokeilla vaihtamalla kirjautumistietoja. Muille käyttäjille testit eivät ole kuitenkaan optimoituja ja testit eivät todennäköisesti toimi kunnolla. Opinnäytetyöhön tehdyt testit pohjautuvat toimeksiantajan toimesta tehtyihin testitapauksiin, jotka ovat tulleet esille tuotetta kehitettäessä.

Hankekohtaisen luettelon testitapaukset on jaettu kolmeen testitapausryhmään: hankekohtaisen luettelon käsittelyyn, henkilöiden tietojen käsittelyyn ja viestien lähettelyyn henkilöille. Kaikkien testien *Suite Setup* eli testien alustaminen on Firefox-selaimen avaaminen, Ticker-sivulle meno ja kirjautuminen palveluun. Testien ajamisen jälkeen tapahtuva *Suite Teardow* eli testien alasajo sulkee selaimen.

4.3.1 Listan käsittely

Ensimmäinen testitapaus listan käsittelyssä on itse listan luonti. Robotti menee listan luontikohtaan ja täyttää tarvittavat tiedot. Tässä testitapauksessa käytettyjä avainsanoja käytetään myöhemmin myös henkilöiden käsittelyssä ja sähköpostiviestien lähettely -testeissä pohjana. Listalle lisätään yksi testihenkilö ja täytetään hänen tietonsa kokonaan, jotta lista ei ole täysin tyhjä myöhemmissä testitapauksissa.

Kun lista on luotu ja muut alun testitapaukset ovat tehty, testataan listan tietojen muokkausta. Ensimmäisessä tietojen muokkaus -testitapauksessa vaihdetaan kaikki listan luonnissa annetut tiedot. Toisessa tietojen muokkaus -testitapauksessa testataan listassa olevan velvollisuudet -tekstin muokkausta. Siinä vaihdetaan ensin englanninkielinen teksti, jonka jälkeen vaihdetaan kieli suomeksi ja muutetaan myös suomenkielinen teksti. Tämän jälkeen testissä tallennetaan vaihdetut tekstit.

Listan henkilöistä on mahdollista ladata Excel-raportti, jota seuraavissa testitapauksissa testataan ja jossa alussa lisättyä henkilöä tarvitaan. Ensimmäisessä tapauksessa ladataan nykyhetken tilanne listasta ja toisessa valitaan päivä, jolloin saadaan listan tilanne menneisyydestä ja pystytään testaamaan listan historiaa.

Viimeinen testitapaus on listan päättäminen eli testataan, että lista voidaan sulkea ja siirtää suljettujen listojen joukkoon. Tässä ja aiemmissa testitapauksissa, joissa ladataan tai tallennetaan jotain, käytetään tarkistukseen ohjelmistoon tehtyä ilmoitusviestiominaisuutta. Jos ilmoituksessa oleva teksti vastaa testissä olevaa muuttujaa, testi menee läpi. Tätä tarkistusta käytetään melkein kaikissa opinnäytetyöhön tehdyistä testeistä.

4.3.2 Henkilöiden käsittely

Henkilöiden käsittely -testit alkavat uuden listan luonnilla, jossa käytetään samoja avainsanoja kuin aikaisemmissa testeissä. Kun lista on luotu, testataan seuraavaksi henkilöiden lisäämistä ja luontia listaan neljällä eri testitapauksella. Käyttäjiä voi luoda tai niitä voidaan importoida muista organisaation listoista ja käyttäjiä voidaan lisätä useampia kerralla.

Ensimmäinen testitapaus testaa muihin listoihin tehtyjen käyttäjien importointia tähän uuteen listaan. Importointi tapahtuu samasta paikasta, jossa lisätään myös uusia käyttäjiä, joten seuraavassa testitapauksessa testataan käyttäjien importointia ja luontia samaan aikaan. Kolmannessa taas testataan pelkästään yhden käyttäjän luontia, kun taas neljännessä puolestaan monen uuden käyttäjän luontia. Henkilöiden lisääminen tarkistetaan ilmoitusviestin avulla sekä *Wait Until Page Contais* -avainsanalla, jonka avulla tarkistetaan, että luotujen henkilöiden sähköpostit tulevat näkyviin listaan.

Viimeisenä testataan henkilön kaikkien tietojen täyttämistä ja muokkausta, koska henkilöiden luonnissa annetaan vain etu- ja sukunimi sekä sähköpostiosoite. Testitapaus täyttää yhden henkilön kaikki tiedot ja tarkistaa tietojen lisäyksen onnistumisen ilmoitusviestistä ja henkilölistasta löytyvästä statuskolumnista. Jos tiedot ovat täytetty, status muuttuu punaisesta pending-sanasta vihreäksi ok-sanaksi. Lopuksi testataan saman henkilön tietojen muokkausta vaihtamalla aiemmin tallennettuja tietoja.

4.3.3 Sähköpostiviestien lähettäminen

Sähköpostien lähettelyä testataan monella tavalla, sillä niitä on mahdollista lähettää usealla eri tavalla. Ennen sähköpostiin liittyviä testitapauksia alustetaan uusi lista ja sinne muutamia käyttäjiä, joille voidaan lähettää sähköpostia. Sähköposteja voidaan lähettää yhdelle tai monelle henkilölle ja niistä lähtee englannin- ja suomenkielinen versio sillä perusteella, mikä kieli henkilölle on valittu listalle lisäämisvaiheessa. Niitä voidaan myös ajastaa lähtemään myöhemmin, jolloin niitä voidaan vielä muokata tai poistaa ennen niiden lähtöä.

Viestin lähettämisessä testataan ensimmäisenä kutsuviestin lähettämistä kaikille listan jäsenille sekä saman kutsun lähettämistä tietylle henkilölle. Viesteistä voidaan myös tehdä täysin muokattuja, joita seuraava testitapaus testaa. Kun testi on muokattu ja lähetetty, käydään muokatut viestit tarkastamassa viestilokista. Sieltä tarkistetaan, että viesti on varmasti muokattu molemmilla kielillä.

Viestien lähettämisessä viimeiseksi testataan normaalin ja muokatun viestin ajastamista sekä ajastetun viestin muokkausta ja poistoa. Ajastetun viestin teko ei eroa normaalin viestin lähettämisestä muuten kuin lähtemispäivän ja kellonajan valinnalla. Ajastetut

viestit menevät lähtevien viestien listaan, josta niitä voidaan muokata vielä tai poistaa kokonaan. Testitapaus, joka muokkaa ajastettua viestiä, myös tarkistaa, että muokkaus pysyy voimassa tallennuksen jälkeen. Viimeinen testitapaus poistaa yhden ajastetun viestin.

4.4 Ilmenneet ongelmat ja ratkaisut

Testejä tehtäessä pitää ottaa huomioon se, että robotti tekee asiat vähän eri tavalla, kuin normaali ihminen. Robotti tekee kaiken paljon nopeammin kuin mitä ihminen pystyy tekemään ja se ei odota, että sivu tai elementti on latautunut kokonaan, jos ei sitä erikseen käske. Robotin nopeus saattaakin aiheuttaa ongelmia, jos ohjelmiston Backend ei pysy sen mukana tai jokin elementti ei ehdi päivittyä. Yleensä näihin ongelmiin toimii *Wait Until* -avainsanat, mutta välillä ne eivät toimi halutulla tavalla tai ne aiheuttavat testien keston pitenemistä.

Ohjelmistossa käytettävät ilmoitusviestit aiheuttivat hieman ongelmia robotin nopeuden takia. Ilmoitusviestit ovat esillä noin neljä sekuntia, minkä takia robotti kerkeää tekemään jotain muuta, mikä taas laukaisee uuden viestin. Toinen ongelma oli, että ilmoitusviesti menee elementin päälle ja robotti ei pysty klikkaamaan kyseistä elementtiä. Robotin olisi voinut laittaa odottamaan, että viesti katoaa, mutta tämä olisi pitkittänyt testausaikaa huomattavasti. Ilmoitusviesteissä on ominaisuutena se, että niitä voidaan klikata, jolloin ne katoavat. Joten ratkaisuksi tehtiin avainsana, joka tarkistaa ilmoitusviestin tekstin, että se on oikea, ja klikkaa sitä aikaa säästääkseen (kuva 8). Avainsana ottaa argumenttina viestin, joka ilmoitusviestissä pitäisi olla.

```
23  Check Notification Bar
24      [Arguments]  ${message}
25      Wait Until Element Is Visible  id=${NOTIFICATION_ID}
26      Element Should Contain  id=${NOTIFICATION_ID}  ${message}
27      Click  ${NOTIFICATION_ID}
28      Wait Until Page Does not Contain  id=${NOTIFICATION_ID}
```

KUVA 8. Ilmoitusviestin avainsana.

Toinen Robotin nopeudesta johtuva ongelma oli sähköpostienviestien lähettämisessä, kun robotti lähettää sähköpostin ja menee tarkistamaan sen sisällön viestilokista. Ohjelmiston

Backend ei ollut kerennyt päivittämään sähköposteja, joten testit kaatuivat siihen, että robotti ei löytänyt viestejä viestilokista. Koska viestiloki ei päivity automaattisesti, ei testeissä voitu käyttää *Wait Until* -avainsanoja järkevästi. Ratkaisuksi käytettiin *Sleep*-avainsanaa, joka pysäyttää testin tässä tapauksessa viideksi sekunniksi. Tämä riitti siihen, että sähköpostit kerkesivät tulla näkyviin viestilokiin. Vaikka *Sleep*-avainsanaa ei suositella käytettäväksi, tuntui se tässä tilanteessa parhaimmaksi vaihtoehdoksi ja se oli helppo ratkaisu ongelmaan.

4.5 Selain

Ohjelmiston testaamista on suositeltavaa tehdä kaikista käytetyimmillä selaimilla. Joulukuussa 2018 käytetyimmät työpöytäselaimet olivat Chrome 70.95%, Firefox 10.05%, IE 5.4% ja Safari 5.06% (Statcounter GlobalStats, 2019). Chrome on ylivoimaisesti käytetyin selain. Toisena oleva selain ei ole kuitenkaan niin selkeä. Statcounterin mukaan Firefox on selkeästi toisena käyttäjämäärältään, mutta muissa lähteissä IE tai Safari on Firefoxia edellä. Tietojen perusteella voidaan olettaa, että IE, Safari ja Firefox ovat aika lähellä toisiaan käyttäjämääriltään, joten testejä siis kannattaa tehdä ainakin Chromella, Firefoxilla ja Safarilla.

Jotta Robot Framework testejä voidaan ajaa, täytyy ladata ja asentaa erilliset ajurit (WebDriver). Firefoxin ajuri on nimeltään GeckoDriver ja Chromen taas on ChromeDriver. Safarin ajuria ei tarvitse erikseen ladata, mutta se pitää käydä laittamassa päälle Safarin kehittäjä asetuksista, sillä se on pois päältä oletuksena. Kun ajurit on asennettu, Robot Framework pystyy käyttämään selaimia testien ajamiseen. Kun testit ajetaan normaalisti, tavallisen näköinen selain aukeaa ja siitä voidaan tarkastella, mitä robotti tekee.

Chromella ja Firefoxilla on mahdollisuus normaalin ajon lisäksi ajaa testejä selaimella headless-tilassa, jolloin selaimen graafinen käyttöliittymä ei aukea. Headless-selain on nopeampi kuin normaali selain, sillä sen ei tarvitse ladata ja avata CSS-, Javascript- ja HTML-tiedostoja. Nopeus verrattuna normaaliin selaimeen vaihtelee kuitenkin aika paljon, mutta noin kaksinkertainen nopeus on normaalia. (Colantonio, J. 2017.) Headless selainta on järkevä käyttää, jos testien etenemistä ei tarvitse nähdä graafisessa käyttöliittymässä eikä bugien etsintää tarvitse tehdä paljoa.

Opinnäytetyötä alettiin tekemään Chromea käyttäen. Kyseinen ajuri toimii erittäin hyvin ja robotti liikkuu todella sulavan näköisesti graafisessa käyttöliittymässä. Firefoxissa robotin liikkuminen taas näyttää vähän tökkivältä verrattuna Chromeen. Tämä näkyy varsinkin, kun robotti kirjoittaa tekstikenttiin jotain. Chromella kirjoittaminen näyttää siltä kuin joku kirjoittaisi oikeasti siihen, kun taas Firefoxilla teksti vain ilmestyy tekstikenttään vähän ajan kuluttua. Tämä on toisaalta vain kosmeettinen ongelma eikä vaikuta itse toimivuuteen millään tavalla.

Opinnäytetyön testejä tehtäessä Chromen ajurissa ilmeni ongelma. Robot Frameworkin Selenium2Library -kirjaston *Input Text* -avainsana etsii parametrina annetun id:n perusteella tekstikentän ja laittaa sen arvoksi toisena parametrina annetun merkkijonon. Normaalisti tämä avainsana poistaa tekstikentässä valmiiksi olevan merkkijonon, jos siinä sellainen on, ja lisää sitten avainsanalle annetun merkkijonon. Chromessa ilmennyt ongelma oli se, että avainsana ei enää poistanut vanhaa tekstiä vaan lisäsi tekstin suoraan sen perään. Ongelmaa lähdettiin korjaamaan omalla pythonilla kirjoitetulla avainsanalla, mutta ongelma ei korjaantunut sillä. Koska ongelman ratkaisuun alkoi kulua sen verran paljon aikaa, päätettiin käyttää Firefoxin Geckodriveria testien ajamiseen. Firefoxin ajurilla testit toimivat normaalisti.

5 KÄYTTÖÖNOTTO

5.1 Jenkins

Jenkins on avoimen lähdekoodin automaatioserveri, jolla voidaan automatisoida ohjelmistokehitysprosessia ja siten nopeuttaa sitä. Sillä voidaan tehdä monia asioita kuten kontrolloida ohjelmiston koontiversiota, testausta, paketointia, käyttöönottoa ja monia muita asioita. Jenkins voidaan laittaa tarkistamaan koodimuutoksia esimerkiksi Githubista ja Jenkins voi automaattisesti aloittaa testauksen. (About Jenkins, 2019.) Black Woodpecker käyttää Jenkinsiä moneen tarkoitukseen kuten esimerkiksi koodin testaukseen, joten sinne oli hyvä laittaa opinnäytetyöstä syntyneet testit, mikä olikin yksi kriteeri työssä.

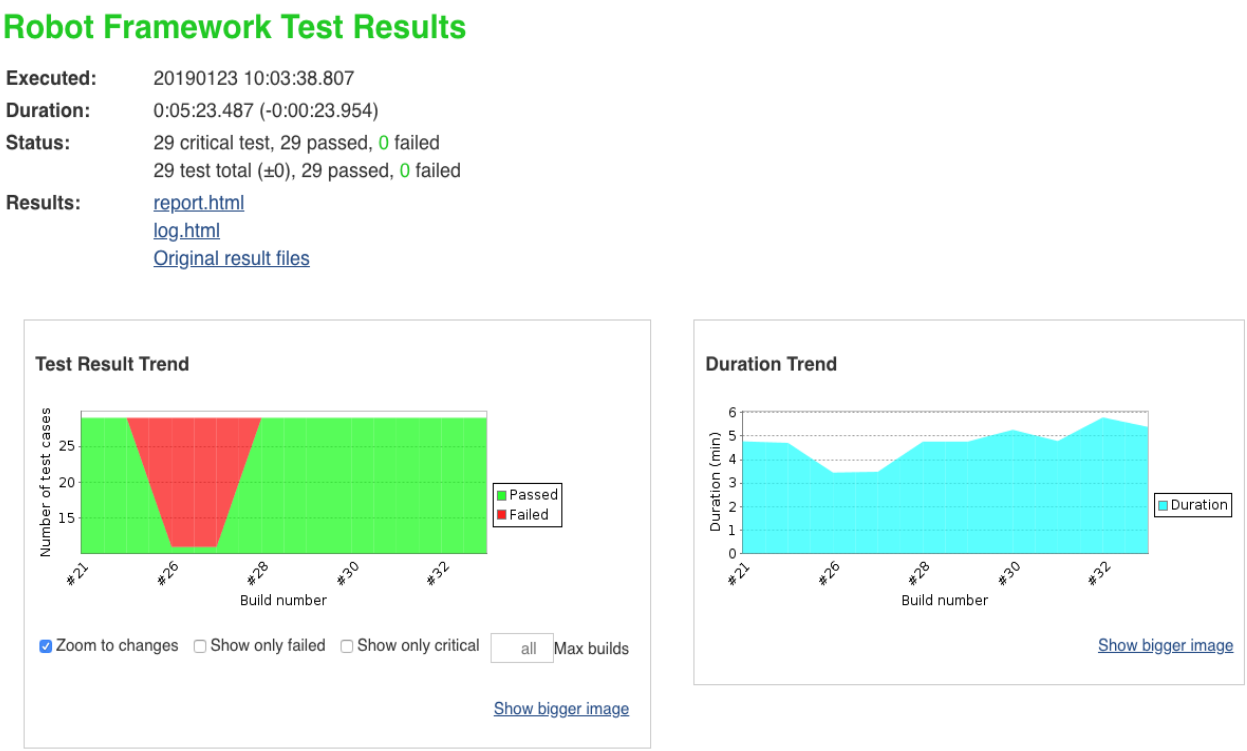
Jenkinsistä löytyy Robot Framework -liitännäinen, joka on todella hyödyllinen testien analysoimiseen ja tarkasteluun itse Jenkinsissä. Sen avulla saadaan Jenkinsissä pyörivien Robot Framework -testien tulokset näkyviin itse Jenkinsin sivuilla. Testeistä tulee Jenkinsiin näkyviin koosteita, joista näkyy, miten testit ovat sujuneet (kuva 9). Tuloksista on nähtävillä myös tarkempi seloste erilaisten graafien kanssa, joista näkyy, kuinka kauan testeihin on mennyt aikaa ja minkälainen onnistumisprosentti testeillä on aikaisemmin ollut (kuva 10). Niistä näkee nopeasti, jos testit ovat epäonnistuneet ja niistä pääsee nopeasti raportti- ja lokitiedostoihin tarkastamaan, miksi testit eivät ole menneet läpi.

Robot Test Summary:

	Total	Failed	Passed	Pass %
Critical tests	29	0	29	100.0
All tests	29	0	29	100.0

- Browse results
- Open report.html
- Open log.html

KUVA 9. Jenkins Robot Framework testi kooste.



KUVA 10. Jenkins Robot Framework testien tarkemmat tiedot.

Jotta testit saadaan pyörimään Jenkinsissä, pitää sinne tehdä uusi työ (job). Opinnäytetyössä käytetään Jenkins Pipeline -työtä. Kun Pipeline-työ tehdään, sille annetaan Githubista koodilähde, jonka muutoksia se tarkkailee. Näiden muutoksien perusteella Pipeline aloittaa sille annettujen toimintojen tekemisen. Toiminnot määritellään Pipeline script -skriptiin, joka voidaan laittaa työn asetuksiin Pipeline script kohtaan tai se voidaan tehdä jenkinsfile-tiedostoon, joka taas laitetaan lähdekoodin kanssa versionhallintaan. Asetuksissa pitää vain määritellä kumpaa vaihtoehtoa käytetään. (Jenkins Handbook 2019.)

Opinnäytetyössä käytetään jenkinsfile-tiedostoa (kuva 11). Kun työ käynnistetään, tekee se ensimmäisenä uuden kansion, johon Robot Frameworkin lokitiedostot menevät. Tämän jälkeen alkaa testaus vaihe, jossa käynnistetään Shell-skripti ja johon ollaan määritelty Jenkins Robot Framework -liitännäinen. Liitännäisen määrittelyssä on kerrottu,

mistä se hakee lokitiedostot ja minkä nimisiä kyseiset tiedostot ovat. Tässä tapauksessa se hakee ne aiemmin tehdystä robot-logs -kansiosta, johon robotti on käsketty laittamaan ne. Jos kansiota, johon lokitiedostot menevät, ei määritellä, ei Robot Framework liitännäinen osaa näyttää testien tuloksia Jenkinsissä.

```
properties properties: [
    disableConcurrentBuilds(),
    pipelineTriggers([pollSCM('H/2 * * * *')])
]

pipeline {
  agent any
  stages {
    stage('Create robot logs folder') {
      steps {
        sh 'mkdir robot-logs'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing..'
        sh './docker/setup.sh'
        step([
            $class : 'RobotPublisher',
            outputPath : "./robot-logs/",
            outputFileName : "output.xml",
            logFileName: 'log.html',
            reportFileName: 'report.html',
            disableArchiveOutput : false,
            passThreshold : 100,
            unstableThreshold: 95.0,
            otherFiles : "*.png",
        ])
```

KUVA 11. Testien Jenkinsfile.

5.2 Docker

Docker on avoin lähdekoodi työkalu, jonka avulla on helppo tehdä, ottaa käyttöön ja ajaa ohjelmia konttien (container) avulla. Niiden avulla ohjelmia voidaan pakata pakettiin, joka sisältää ohjelman tarvitsemat asiat. Kontin avulla varmistetaan, että ohjelma toimii kaikissa ympäristöissä samalla tavalla eristämällä se sen ympäristöstä. (Docker, 2019.)

Dockerin käyttö alkaa Docker -tekstitiedoston tekemisellä, jossa on komentoja, joita voidaan komentorivillä käyttää. Siitä tehdään image komennolla *docker build* ja kun image ajetaan komenolla *docker run,* syntyy siitä kontti.

Aiemmin esiintynyt Shell-skripti jenkinsfile-tiedostossa käynnistää testit Dockerin avulla. Ticker -ohjelmiston front- ja backend sekä itse robottitestit laitetaan kaikki eri Docker -kontteihin, jotka on määritelty Docker-tiedostoissa. Front- ja backendin konteissa asennetaan niiden tarvitsemat kirjastot ja käynnistetään kumpikin development-ympäristöön. Robotin kontissa asennetaan python, Robot Framework ja Selenium2Library-kirjasto (kuva 12). Koska kontteja on kolme, käytetään niiden ajamiseen Docker Compose -toimintoa, jonka avulla voidaan ajaa useita kontteja sisältäviä Docker -ohjelmistoja. Kontit kootaan docker-compose.yml -tiedostoon (kuva 13).

```
1   FROM alpine:latest
2
3   ADD ./docker/wait-for/wait-for-python ./wait-for
4
5   # PYTHON
6   RUN apk add --no-cache python py-pip
7
8   # ROBOT + LIBRARIES
9   COPY requirements.txt /tmp/requirements.txt
10  RUN  pip install -r /tmp/requirements.txt
11
```

KUVA 12. Robotin Docker -tiedosto.

```
24  ticker_frontend:
25    build:
26      context: ./ticker-web
27      dockerfile: ./Dockerfile_frontend
28    env_file:
29      - ./docker-envs/env_ticker_frontend
30    environment:
31      REACT_APP_API_BASE_URL: http://ticker_backend:3000/api/v1
32    expose:
33      - 5201
34    depends_on:
35      - ticker_backend
36
37  selenium_hub:
38    image: selenium/hub
39    expose:
40      - 4444
41
42  firefoxnode:
43    image: selenium/node-firefox
44    environment:
45      - HUB_PORT_4444_TCP_ADDR=selenium_hub
46      - HUB_PORT_4444_TCP_PORT=4444
47    expose:
48      - 5555
49
```

KUVA 13. Docker-compose -tiedosto.

Front- ja Backendin lisäksi pitäisi saada vielä selain toimimaan Dockerissa, mikä onnistuu helposti Docker Hubista löytyvillä Selenium grid -imageilla. Selenium grid toimii siten, että sen Selenium hub toimii keskuksena, johon liitetään eri nodeja (Guru99). Tässä tapauksessa hubiin liitetyt nodet ovat selaimia, joissa testejä ajetaan.

Dockerissa Selenium grid toimii niin, että docker-compose -tiedostoon lisätään Selenium gridiin valmiiksi tehdyt image *selenium/hub* ja vaikka firefox-selaimesta tehty image *selenium/node-firefox*. Firefox-node liitetään hubiin antamalla sille ympäristöksi hubin sijainti ja portti, joka on oletuksena 4444. Testeissä taas *Open Browser* -avainsanaan lisätään *remote_url*-argumentti, jotta se löytää selaimen ja sen ajurin, joka pyörii hubissa (kuva 14). Tässä tapauksessa osoite on ” http://selenium_hub:4444/wd/hub”.

```
Open Browser In Ticker Url
    Run Keyword If  ${REMOTE}  Open Browser  ${TICKER_URL}  ${DEFAULT_BROWSER}  remote_url=${FIREFOX_URL}
   ...  ELSE  Open Browser  ${TICKER_URL}    ${DEFAULT_BROWSER}
    Set Window Size    1920    1080
```

KUVA 14. Selaimen aukaisu Dockerissa.

6 POHDINTA

Tavoitteena opinnäytetyössä oli ottaa käyttöön Robot Framework kehys Ticker -ohjelmistossa. Robot Frameworkia ei aikaisemmin oltu käytetty Black Woodpeckerin ohjelmistoissa, mutta työntekijöillä kuitenkin oli kokemusta siitä. Koska testejä ei tehty koko Ticker-ohjelmistoon, on vaikea arvioida opinnäytetyöstä syntyvää hyötyä ohjelmiston kehityksessä. Iso osa ohjelmistosta täytyy testata vielä käsin, mutta jos tehtyjä testejä käytetään säännöllisesti, voidaan testien kattamalla alueella keskittyä testaamaan käsin sellaisia asioita, joita Robot Frameworkilla ei voida testata.

Tällä hetkellä tehdyt testit toimivat hyvänä pohjana testeille, jos niitä ruvetaan tekemään lopulle ohjelmistolle, sillä testeihin tehtyjä avainsanoja voidaan käyttää helposti ohjelmiston muissa paikoissa. Testien kehitys kuitenkin kulkee käsi kädessä ohjelmiston kehityksen kanssa ja testejä joudutaan tekemään lisää tai muuttamaan uusien ominaisuuksien takia, joita ohjelmistoon tehdään. Hyvin tehdyillä testeillä säästetään aikaa ja rahaa.

Opinnäytetyössä tehtyihin Robot Framework testeihin löytyy paljon kehitysmahdollisuuksia. Esimerkiksi tällä hetkellä Jenkinsissä pyörivät testit testaavat Ticker-ohjelmiston back- ja frontendia vain niiden master-haarojen pohjalta. Skripti, joka aloittaa testit Jenkinsissä olisi hyvä parametrisoida niin, että sen on mahdollista hakea back- ja frontendista tietyt haarat. Se mahdollistaa ominaisuuden testaamisen ennen master-haaraan lisäämistä. Testit testaavat ohjelmistoa tällä hetkellä vain Firefoxilla ja ne olisi hyvä saada pyörimään myös muilla selaimilla. Tämä onnistuu Selenium gridin avulla, mutta siihen pitäisi perehtyä vähän tarkemmin. Kehitys kohteita löytyy aina lisää ja niistä pitääkin priorisoida tärkeimmät asiat, sillä kaikkeen ei riitä aikaa.

Robot Framework on monipuolinen testauskehys, jonka avulla saadaan automatisoitua hyväksymistestausprosessi. Kaikkeen se ei tietenkään pysty, mutta testien avulla nostetaan tuotteen laatua ja varmistetaan, että asiat toimivat niin kuin niiden kuuluu. Jos hyväksymistestausta ei ole kokeiltu, on Robot Framework hyvä ja helppo vaihtoehto, josta kannattaa lähteä liikkeelle.

LÄHTEET

Bisht, S. 2013. Robot Framework Test Automation. Packt Publishing.

Cloudbees. About Jenkins. Luettu 1.4.2019. https://www.cloudbees.com/jenkins/about

Colantonio, J. 2017. Headless browser testing pros and cons. Luettu 27.1.2019 https://www.joecolantonio.com/headless-browser-testing-pros-cons/

Docker. What is a Container. Luettu 2.4.2019 https://www.docker.com/resources/what-container

Guru99. Introduction to selenium grid luetu 6.4.2019. https://www.guru99.com/introduction-to-selenium-grid.html

Hambling, B. ja Van Goethem, P. 2013. User acceptance testing: A step-by-step Guide. BCS Learning & Development Limited.

How to write good test cases. Luettu 17.1.2019. https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst

Installation instructions. Luettu 11.3.2019. https://github.com/robotframework/robotframework/blob/master/INSTALL.rst

Jenkins Handbook. Pipeline. Luettu 1.4.2019. https://jenkins.io/doc/book/pipeline/

Pip Installation. Luettu 12.3.2019. https://pip.pypa.io/en/stable/installing/

ProfessionalQA. V Model. Luettu 3.4.2019. http://www.professionalqa.com/v-model

Python setup and usage. Luettu 11.3.2019. https://docs.python.org/2.7/using/mac.html

Robot Framework. Luettu 22.11.2018. https://robotframework.org/

Robot Framework User Guide. Luettu 27.12.2018. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Robot Framework SeleniumLibrary. Luettu 16.1.2019. http://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

Sale, D. 2014. Appying unit testing, TDD, BDD and acceptance testing. John Wiley & Sons, Incorporated

Statcounter. GlobalStats. Luettu 21.1.2019. http://gs.statcounter.com/browser-market-share/desktop/worldwide/#monthly-201811-201811-bar

Tutorialspoint. SDLC. Luettu 3.4.2019. https://www.tutorialspoint.com/sdlc/sdlc_overview.htm