TAMPEREEN
AMMATTIKORKEAKOULU

WEB-KÄYTTÖLIITTYMÄ JA ROBOT FRAMEWORK: AUTOMAATIOTESTIEN RAKENTAMINEN OSANA TUOTEKEHITYSTÄ

Juuso Hämäläinen

Opinnäytetyö
Joulukuu 2017
Tietojenkäsittely
Web-palvelut

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Web-palvelut

HÄMÄLÄINEN JUUSO:
Web-käyttöliittymä ja Robot Framework: Automaatiotestien rakentaminen osana tuotekehitystä

Opinnäytetyö 33 sivua, joista liitteitä 3 sivua
Joulukuu 2017

Opinnäytetyön tavoitteena oli selvittää, kuinka automaatiotestien kehittäminen tulisi toteuttaa web-käyttöliittymälle osana tuotekehitystä. Automaatiotestien rakentamiseen käytettiin Robot Framework -kehystä, johon liitettiin Selenium2-kirjasto, jolla mahdollistettiin web-käyttöliittymän testaus. Robot Framework osoittautui toimivaksi työkaluksi automaatiotestien rakentamiseen web-käyttöliittymälle. Opinnäytetyön toimeksiantajana toimi Loikka Design Oy, joka kehittää hoitoyrityksille Nursebuddy-nimistä ohjelmistoa kotihoitoa varten.

Opinnäytetyössä selvitettiin, kuinka Pythonin, Robot Frameworkin ja Selenium2-kirjaston asentaminen ja käyttöönotto tapahtuvat Windows 7 ja Mac- käyttöjärjestelmille. Opinnäytetyössä tutkittiin, miten Robot Framework toimii, kuinka testien rakentaminen sen avulla onnistuu ja mitä käyttöliittymää tehdessä tulee muistaa, jotta testien rakentaminen olisi sujuvaa.

Testaus on tärkeä osa onnistunutta tuotekehitystä. Opinnäytetyössä perehdyttiin tarkemmin siihen mitä testaus on, sekä sen yleisimpiin tyyppeihin. Työssä tutustuttiin tarkemmin myös testauksen periaatteisiin, käytäntöihin ja asioihin, joita tulee huomioida, jotta testauksesta saadaan kattavaa ja luotettavaa.

Asiasanat: testaus, automaatio, tuotekehitys, robot framework, käyttöliittymä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Business Information Systems
Web-services

HÄMÄLÄINEN JUUSO:
Web Interface and Robot Framework: Building Automation Tests as a Part of Product Development

Bachelor's thesis 33 pages, appendices 3 pages
December 2017

The aim of this thesis was to examine how automated tests should be implemented for a web interface as a part of product development. Robot Framework was used for implementing automated tests together with the Selenium 2 library to enable testing of web interfaces. Robot Framework turned out to be a useful tool for creating automation tests for a web interface. The thesis was commissioned by Loikka Design Oy, a company that develops home care application.

Discussion is provided on the installing of Python, Robot Framework and the Selenium 2 library for the Windows 7 and Mac operating systems. The working principles of the Robot Framework are also examined in this thesis, including a walk-through on the steps needed in creating tests. Furthermore, these principles are also inspected in relation to the interface to make the process of creating tests easier.

Testing is an important part of product development and successful projects. A closer look is taken on what testing is and the most common test types are introduced. The principles as well as practices and issues related to testing that need to be considered are also discussed to ensure adequate reliability and extensiveness of tests.

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

SISÄLLYS

TERMIT

Automaatio	Toiminto, joka suoritetaan ilman ihmisen väliintuloa koneen toimesta
Elementti	Web-käyttöliittymän yksittäinen pala
HTML	Hypertext Markup Language
IronPython	Pythonin C#-kielinen toteutus
Jython	Pythonin Java-kielinen toteutus
Pip	Pythonilla kirjoitettujen pakettien hallintasovellus
Python	Ohjelmointikieli
Robot Framework	Avoimen lähdekoodin testiautomaatio -kehys
Selenium2Library	Verkkosivujen testaukseen käytetty Robot Framework kirjasto
Testi	Toimenpide, joka varmistaa järjestelmän toimivuutta
XML	Extensible Markup Language

1 TYÖN LÄHTÖKOHDAT

1.1 Toimeksiantaja

Opinnäytetyön toimeksiantajana oli Tampereella toimiva yritys Loikka Design Oy, joka kehittää hoitoyrityksille Nursebuddy -nimistä ohjelmistoa kotihoitoa varten. Loikka Design Oy perustettiin Turussa vuonna 2012. Yritys panostaa jatkuvasti tuotteen kehitykseen ja tuotteen asiakkaita löytyy Suomen lisäksi monista muista Euroopan maista. Jatkuvan tuotekehityksen takia ja myös tuotteen luotettavuuden takaamiseksi yritys on automatisoinut testausta. Vuonna 2013 NurseBuddy valittiin kansainväliseen HealthXL-kiihdytysohjelmaan, johon pääsivät 8 maailmanlaajuisesti lupaavinta terveydenhoitoalan startup-yritystä. (Nursebuddy)

1.2 Tavoitteet

Opinnäytetyön tavoitteena oli selvittää, kuinka web-käyttöliittymän automaatiotestaus tulisi toteuttaa osana yrityksen tuotekehitystä. Automaatiotestien työkaluksi valittiin Robot Framework. Työssä käydään aluksi läpi automaatiota tuotekehityksen näkökulmasta. Tämän jälkeen selvitetään mikä on Robot Framework, kuinka sen asennus ja käyttöönotto onnistuvat Windows ja Mac -käyttöjärjestelmillä, sekä kuinka sillä voidaan kirjoittaa testejä web-käyttöliittymää varten, ja mitä on hyvä ottaa huomioon. Opinnäytetyössä perehdytään myös siihen mitä testaus on, sekä sen yleisimpiin tyyppeihin. Työssä tutustutaan tarkemmin myös testauksen periaatteisiin, käytäntöihin ja asioihin, joita tulee huomioida, jotta testauksesta saadaan kattavaa ja luotettavaa.

2 TESTAUKSEN AUTOMATISOINTI

2.1 Automaation määrittely

Testauksen automatisointi tarkoittaa ohjelmiston työkalujen käyttöä suunnittelemaan ja ohjelmoimaan testejä, joita voidaan jatkuvasti toistaa tietokoneella. Automatisointi siirtää manuaalisen työn ohjelmiston työkalujen varaan. (Spillner, A. 2014, 286)

Manuaalinen tuotetestaus on ihmisen tekemää työtä, jossa hän käy varovasti läpi tuotteen eri näkymiä, eri käyttötapauksia ja yhdistelmiä, ja vertaa niitä oletettuun käyttäytymiseen. Manuaaliset testit toistetaan usein kehityksen varrella lähdekoodin muuttuessa tai kehitysympäristön vaihtuessa tai laitteiston muutoksissa. (Why automated testing?)

Automaattinen testaustyökalu pystyy toistamaan ennalta määrättyjä tapahtumia ja verrata niitä oletettuun tapahtumaan sekä raportoimaan onnistumisista tai epäonnistumisista. Kun automaatiotesti on määritelty, sitä voidaan helposti toistaa lukuisia kertoja. Näin voidaan luoda paljon monimutkaisempia käyttötapauksia, joita olisi mahdotonta testata manuaalisesti. (Why automated testing?)

2.2 Automaatio yrityksissä

Jokainen ohjelmistokehitysryhmä testaa tuotettaan, mutta silti ohjelmistoon jää vikoja. Testaajat pyrkivät löytämään kaikki viat ennen tuotteen julkaisua, mutta silti niitä päätyy tuotteeseen, vaikka se olisi manuaalisesti huolellisesti testattu. (Why automated testing?) Testaamiseen ja arviointiin arvellaan kuluvan 25% - 50% ohjelmiston kehitysajasta (Koomen, T. 1999). Automaatiotestaus on paras tapa lisätä tehokkuutta ja testauksen kattavuutta. (Why automated testing?)

Yritykset ovat ottaneet käyttöön automaatiotestit osaksi tuotekehitystä ja ymmärtäneet, että ne ovat olennainen osa onnistunutta yrityshanketta. Automaatiotestausta on pitkään pidetty kriittisenä osana ison tuotteen kehittämistä, mutta sitä on usein pidetty kalliina tai vaikeana vaihtoehtona pienemmille yrityksille ottaa mukaan. (Why automated testing?)

3 ROBOT FRAMEWORK

3.1 Johdanto

Robot Framework on testiautomaatio -kehys. Se on tarkoitettu hyväksymistestaukseen ja hyväksymistestivetoiseen kehitykseen. (Robot Framework) Hyväksymistestaus tarkoittaa sitä, että testattavalle kohteelle asetetaan tietyt vaatimukset, jotka testin tarvitsee läpäistä ollakseen hyväksytty. Hyväksymistestivetoinen ohjelmistokehitys toimii samalla periaatteella kuin hyväksymistestaus, mutta siinä asiakas, kehittäjä ja testaajat keskustelevat yhdessä asiakkaiden tarpeista, joista muodostuvat vaatimukset ohjelmistolle.

Sumit Bisht (2014, 8) kirjoittaa, että hyväksymistestivetoisen kehityksen tarkoitus nopeuttaa ja tehostaa kehitystä sekä lisätä luotettavuutta ohjelmiston eri ominaisuuksille. Hän jakaa hyväksymistestivetoisen kehityksen tuomat hyödyt neljään kategoriaan:

1. Ohjelmiston häiriöiden paikannus
   Testauksen kautta voidaan tunnistaa ohjelmiston erilaisia virheitä ja tehokkuuteen liittyviä häiriöitä, jotka ovat saattaneet kehittäessä jäädä huomaamatta
2. Vähentynyt virheiden määrä
   Automaation avulla, suunniteltujen vaiheiden testaaminen voidaan määritellä, kuten halutaan, eikä siihen tarvita käyttäjän vuorovaikutteisuutta
3. Automaatio ja uudelleenkäytettävyys
   Testaajat tai muut ihmislähtöiset resurssit ovat kalliimpia kuin tietokoneen suorittamat tehtävät. Tästä syystä on järkevää automatisoida toistuva työ, mikä vähentää aikaa mikä muuten menisi kirjoitukseen, klikkauksiin ja käyttöliittymän opetteluun testaajalta.
4. Testien tulostiedot
   Pitämällä tulostietoja testeistä, voidaan kerätä ja löytää mielenkiintoisia tietoja esimerkiksi siitä, kuinka paljon järjestelmä rasittuu testauksesta ja kuinka paljon virheitä tapahtui. Näitä tietoja voidaan käyttää järjestelmän kehittämiseen.

Robot Framework hyödyntää avainsanapohjaista (keyword) lähestymistä testeihin (Robot Framework). Robot Framework on toteutettu Pythonilla ja sitä voidaan myös käyttää Jythonilla ja IronPythonilla. Robot Framework toimii Pythonin versiolla 2 ja 3. (Robot Framework User Guide, 2017)

Robot Framework on Pekka Klärckin (Klärck, P. Eliga Oy) aloittama avoimen lähdekoodin projekti, joka on alun perin kehitetty Nokia Networksin toimesta ja on nykyisin Robot Framework Foundationin sponsoroima. Robot Frameworkissa on selkokielinen syntaksi ja sen toiminnollisuuksia on mahdollista laajentaa Pythonilla tai Javalla kirjoitetuilla testikirjastoilla. (Robot Framework)

3.2 Käyttöönotto

Robot Frameworkin käyttöönotto onnistuu kaikilla käyttöjärjestelmillä (User Guide, 2017), mutta opinnäytetyössä käydään läpi sen asentaminen Windows 7 ja Mac -käyttöjärjestelmille. Asennuksen tavoitteena on, että molemmilla käyttöjärjestelmillä voidaan onnistuneesti ajaa testejä sekä Google Chrome - että Firefox -selaimissa.

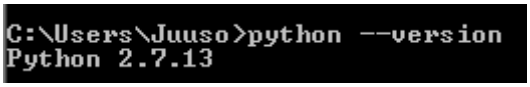
Opinnäytetyössä Robot Framework asennetaan Pythonin versiolle 2 sen hyvän tuettavuuden johdosta. Vaikka Robot Framework tukee Pythonin versiota 3, eivät kaikki kirjastot ole vielä siirtyneet käyttämään sitä. Mac-käyttöjärjestelmästä Python pitäisi löytyä valmiiksi asennettuna, mutta Windowsissa se tarvitsee asentaa itse. (Robot Framework User Guide, 2017)

3.2.1 Asentaminen Windows 7-käyttöjärjestelmälle

Robot Framework vaatii Pythonin version 2.6, 2.7, 3.3 tai uudemman toimiakseen. Aloitetaan käyttöönotto lataamalla ja asentamalla Python tietokoneelle. Pythonin saa ladattua Pythonin kotisivulta (http://www.python.org/).

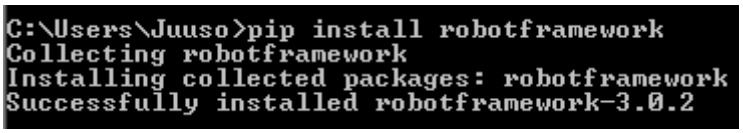
Asentamisen jälkeen tarvitsee asettaa Pythonin polku (path) (Robot Framework User Guide, 2017). Python tarvitsee asettaa tietokoneen ympäristömuuttujaksi, jotta sen komentoja voidaan käyttää kaikkialla komentoriviltä (Robot Framework User Guide, 2017). Polun asettaminen on nähtävissä liitteessä 1.

Polun (path) asettamisen jälkeen tarkistetaan, että Python asentui oikein. Mikäli polun asettaminen meni oikein, pitäisi komentoriville tulla tieto Pythonin versiosta, kun syöttää komennon *python –version* (KUVA 1).



KUVA 1. Python version tarkastaminen Windowsilla

Pythonin asentamisen jälkeen voidaan asentaa Robot Framework. Pythonin versiossa 2.7.9 ja sitä myöhemmissä versioissa, asennuksen mukana tulee ja aktivoituu Pip - pakетinhallintasovellus, jota voidaan käyttää asentamaan tarvittavia paketteja komentorivin kautta. Robot Frameworkin saa ladattua komennolla *pip install robotframework* (KUVA 2). (Robot Framework User Guide, 2017)



KUVA 2. Robot Frameworkin asentaminen Windowsille

3.2.2 Asentaminen Mac-käyttöjärjestelmälle

Mac-käyttöjärjestelmässä pitäisi olla valmiiksi asennettuna Python, joten aluksi varmistetaan, että se on asennettuna. Pythonin asennuksen voi varmistaa kirjoittamalla komennon *python -V* terminaaliin. Jos terminaaliin ei tule tekstiä komennon kirjoittamisen jälkeen, ei Python ole asennettuna. Lataa ja asenna Pythonin versio 2,6, 2.7, 3.3 tai uudempi osoitteesta (http://www.python.org/). Pythonin asentamisen jälkeen ei pitäisi tarvita erikseen syöttää Pythonin polkua ympäristömuuttujaksi, jos Pythonia ei ole aikaisemmin asennettu. (Robot Framework User Guide, 2017)

Robot Frameworkin asentaminen on mahdollista käyttäen Mac-käyttöjärjestelmästä löytyvää easy_install -paketinhallintasovellusta, mutta opinnäytetyössä Robot Frameworkin asentamiseen käytetään Pip -paketinhallintasovellusta, koska sitä suositellaan asennuksen tekemiseen virallisissa ohjeissa. (Robot Framework User Guide, 2017)

Pipin asentaminen tapahtuu terminaalissa komennolla *sudo easy_install pip* (KUVA 3).

```
sudo easy_install pip
```

KUVA 3. Pip -paketinhallintasovelluksen asennus Mac -käyttöjärjestelmällä

Asennuksen jälkeen tarkistetaan, että Pip asentui oikein kirjoittamalla *pip –version* terminaaliin (KUVA 4). Terminaaliin pitäisi tulla Pipin versiotiedot, jos se asentui oikein.

```
pip --version
```

KUVA 4. Pipin version tarkistaminen Mac -käyttöjärjestelmällä

Pipin asentamisen jälkeen voidaan asentaa Robot Framework käyttämällä komentoa *pip install robotframework* (KUVA 5).

```
Juuso$ pip install robotframework
Collecting robotframework
  Downloading robotframework-3.0.2 tar.gz (434kB)
    100% |████████████████████████████████| 434kB 589kB/s
Building wheels for collected packages: robotframework
  Running setup.py bdist_wheel for robotframework
  Stored in directory: /Users/Juuso/Library/Caches/pip/wheels/5c/52/06/c52eb8
Successfully built robotframework
Installing collected packages: robotframework
Successfully installed robotframework-3.0.2
```

KUVA 5. Robot Frameworkin asentaminen Mac -käyttöjärjestelmälle

3.2.3 Selenium 2 Library

Selenium on Robot Framework- kirjasto, joka mahdollistaa web-käyttöliittymien testauksen. Se on suosittu ja yksi ensimmäisistä avoimen lähdekoodin ratkaisuista selainpohjaiseen testiautomaatioon (Simon Stewart, 2009). Selenium WebDriver käyttää hyväksi selaimien tarjoamia ajureita (driver). Selenium WebDriver ohjaa selaimien ajureita, jotka taas ohjaavat selaimissa tapahtuvia toimenpiteitä kuten esimerkiksi hiiren painalluksia. (SeleniumHq)

Seleniumin asentaminen onnistuu Windows ja Mac -käyttöjärjestelmillä käyttämällä paketinhallintasovellus Pipiä. Asennus tapahtuu syöttämällä komennon *pip install robotframework-Selenium2Library* (KUVA 6).

```
C:\Users\Juuso>pip install robotframework-selenium2Library
Collecting robotframework-selenium2Library
Requirement already satisfied: decorator>=3.3.2 in c:\python27\lib\site-packages
 (from robotframework-selenium2Library)
Requirement already satisfied: selenium>=2.32.0 in c:\python27\lib\site-packages
 (from robotframework-selenium2Library)
Requirement already satisfied: robotframework>=2.6.0 in c:\python27\lib\site-pac
kages (from robotframework-selenium2Library)
Installing collected packages: robotframework-selenium2Library
Successfully installed robotframework-selenium2Library-1.8.0
```

KUVA 6. Selenium2 kirjaston asentaminen Windows -käyttöjärjestelmälle

Jotta testejä voidaan suorittaa Google Chrome - ja Firefox -selaimilla, tarvitaan ajurit niitä varten. Ajurien lataamiseen ja käyttöönottoon löytyvät ohjeet liitteestä 2.

3.3 Esimerkkitesti

Asennusten jälkeen voidaan luoda ensimmäinen testi, jotta nähdään, että kaikki osat toimivat yhdessä. Esimerkissä luodaan testi, jonka tavoite on avata Googlen haku ja hakea siitä käyttäjän määrittelemä teksti.

Aloitetaan luomalla *test_suite_1.robot* -niminen tiedosto halutun kansion sisään. Esimerkissä testitiedosto on laitettu työpöydälle luotuun kansioon nimeltä *testsuites*. Testitiedostoon tuodaan aluksi web-käyttöliittymän testaukseen vaadittava kirjasto *Selenium2Library*. Sen jälkeen asetetaan neljä muuttujaa *variable* -osion alle (palvelimen osoite, selaimen nimi, viittaus hakukenttään ja url-osoitteen kehykset), jotta niitä voidaan minkä tahansa avainsanan kanssa. Muuttujien jälkeen määritellään kolme avainsanaa *keywords*-

osioon, jotka pitävät sisällään testin toiminnallisuudet sekä yksi testitapaus *test cases* -osioon, joka määrittää testin nimen ja rakenteen (KUVA 7).

```
*** Settings ***
Library           Selenium2Library

*** Variables ***
${SERVER}         google.fi
${BROWSER}        Chrome
${SEARCHFIELD}    lst-ib
${URL}      http://${SERVER}/


*** Keywords ***
Open Browser To Google
    Open Browser    ${URL}    ${BROWSER}

Wait For Search Field
    Sleep     5s
    Page Should Contain Element       ${SEARCHFIELD}
    Element Should Be Visible         ${SEARCHFIELD}

Search From Google
    [arguments]     ${text}
    Input Text    ${SEARCHFIELD}    ${text}
    Press Key     ${SEARCHFIELD}    \\13

***Test Cases***
Google Search Test
    Open Browser To Google
    Wait For Search Field
    Search From Google    Robot Framework
```

KUVA 7. Testitapaus, Google haku

Kun testi on saatu kirjoitettua, siirrytään komentorivillä kansioon, jossa testitiedosto sijaitsee ja ajetaan komento *robot test_suite_1.robot*. Jos käyttöönotto on mennyt oikein pitäisi testin onnistua (KUVA 8).

```
C:\Users\Juuso\Desktop\testsuites>robot test_case_1.robot
==============================================================================
Test Case 1
==============================================================================
Google Search Test                                                    | PASS |
------------------------------------------------------------------------------
Test Case 1                                                           | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\Juuso\Desktop\testsuites\output.xml
Log:     C:\Users\Juuso\Desktop\testsuites\log.html
Report:  C:\Users\Juuso\Desktop\testsuites\report.html
```

KUVA 8. Onnistuneen testitapauksen tiedot komentorivillä

4 TESTIEN KIRJOITTAMINEN

4.1 Johdanto

Testit organisoidaan Robot Frameworkissa kolmionmalliseen hierarkiaan, joka koostuu testisarjasta (test suite), testitapauksesta (test case) ja testitapahtumasta (test action). Tämä on yleinen käytäntö, jota suurin osa jäsennellyistä testeistä noudattaa. Testisarja kokoaa testitapaukset kokonaisuudeksi ja näin kaikki testit saa ajettua kutsumalla testisarjaa. Testitapaus on kokonainen testi yhteen asiaan. Testitapahtuma taas on pienin osa testistä ja sen tehtävänä on vahvistaa sille annettu ehto. (Bisht 2013, 22) Testitapahtuma voi olla esimerkiksi kahden arvon laskeminen yhteen ja saadun tuloksen palauttaminen.

4.2 Testitiedoston rakenne

Robot Frameworkin testitiedostot kirjoitetaan taulukkomuodossa, tarkoittaen, että eri toiminnot erotetaan sarakkeiden avulla. Robot Framework tukee useita eri formaatteja sarakkeiden erottamiseen. Tuettuja formaatteja ovat HTML, TSV, Plaintext, Piped text, RestructuredText. (Bisht. 2013, 26)

Opinnäytetyön esimerkeissä tullaan käyttämään selkotekstiä (plaintext) sen suosion, yksinkertaisuuden ja nopean muokattavuuden johdosta. Selkoteksti on formaateista kaikista helppolukuisin ja se sisältää vähiten merkkejä (Bisht 2013, 26).

Toiminnot erotetaan selkotekstissä toisistaan kahdella tai useammalla välilyönnillä. (Bisht 2013, 26). Formaatti voi tuntua aluksi hieman erilaiselta ja kömpelöltä henkilölle, jolla on kokemusta koodauksesta, mutta formaattiin tottuu kyllä nopeasti. Selkotekstin helppokäyttöisyys ja muokattavuus näkyy siten, että testejä voi rakentaa ja muokata millä tahansa tekstieditorilla. Moniin tekstieditoreihin on myös mahdollista saada Robot Frameworkin syntaksia tukevia lisäosia (Klärck 2009), joilla saa värityksen ja automaattisen sanojen täydennyksen, mikä helpottaa testien kehitystä.

4.3 Testien nimeäminen

Testien nimeäminen on tärkeä osa standardointia ja johdonmukaisuutta. Hyvin nimetty testi myös viestii sen laadusta, sillä hyvin nimetyt testit osoittavat selkeästi niiden käyttötapauksen ja järjestyksen. Tämä auttaa testien hallinnassa tulevaisuudessa. (Bisht 2013, 22).

4.4 Muuttujat

Muuttujat (variable) ovat olennainen osa Robot Frameworkia. Muuttujia käytetään testeissä argumentteina avainsanoille (keyword) ja avainsanatauluille. Muuttujia on mahdollista määritellä myös kaikkialla asetuksissa. (Robot Framework User Guide, 2017).

Robot Frameworkista löytyy useampia muuttujia: skalaarit, listat, kirjastot ja ympäristömuuttujat. Muuttuja määritellään syntaksilla ${skalaari}, @{lista}, &{kirjasto} ja %{ympäristömuuttuja}. (Robot Framework User Guide, 2017). Käytännössä siis vain aaltosulkeiden edessä oleva merkki muuttuu.

4.5 Avainsanat

Robot Framework hyödyntää avainsanapohjaista lähestymistä testeihin, mikä tarkoittaa sitä, että testit koostuvat eri avainsanoista (keyword) ja niille syötettävistä argumenteista ja muuttujista (variable). Avainsanat pitävät siis sisällään erilaisia toimintoja ja niille voidaan syöttää eri arvoja, jolloin ne palauttavat eri arvoja (Robot Framework User Guide, 2017).

Otetaan esimerkkinä Fail-avainsana (keyword), joka on yksi Robot Frameworkiin sisäänrakennetuista avainsanoista. Fail-avainsanan avulla saa testin epäonnistumaan, jos testissä tapahtuu jotain, mitä ei pitäisi. Fail-avainsanalle voidaan määritellä tietty teksti yhtenä argumenttina (KUVA 9), joka tulee näkyviin, kun tarkastellaan testituloksia (KUVA 10).

```
***Test Cases***

Failed Test Action
    Fail    This is a failed test
```

KUVA 9. Esimerkki Fail-avainsanan käytöstä

```
C:\Users\Juuso\Desktop\testsuites>robot test_case_1.robot
==============================================================================
Test Case 1
==============================================================================
Failed Test Action                                                    | FAIL |
This is a failed test
------------------------------------------------------------------------------
Test Case 1                                                           | FAIL |
1 critical test, 0 passed, 1 failed
1 test total, 0 passed, 1 failed
==============================================================================
Output:  C:\Users\Juuso\Desktop\testsuites\output.xml
Log:     C:\Users\Juuso\Desktop\testsuites\log.html
Report:  C:\Users\Juuso\Desktop\testsuites\report.html
```

KUVA 10. Fail-avainsanan syöte

Yhteisten avainsanojen (keyword) luominen on erittäin tärkeää ylimääräisen toistamisen välttämiseksi. Hyvä periaate on, jos tietyt toimenpiteet mahdollisesti tarvitsee toistaa tulevaisuudessa, kannattaa niistä tehdä oma avainsanansa. Esimerkiksi tietyn sivun avaaminen, joka vaatii vain kaksi avainsanaa, on järkevää luoda omaksi hyvin nimetyksi avainsanakseen, koska sitä tullaan käyttämään todennäköisesti usein, ja muuttujan nimen muuttuessa, tarvitsee sen nimi muuttaa vain yhdestä paikasta.

4.6 Kirjastot

Robot Framework kirjastot ovat Pythonilla kirjoitettuja ulkoisia ja sisäänrakennettuja tiedostoja, joissa on erilaisia operaatioita, kuten vertailuja ja muokkauksia. Kaikki avainsanat (keyword) sijaitsevat kirjastoissa. (Robot Framework User Guide, 2017). Kirjastot tarjoavat testaajille helpon tavan tuoda monimutkaisiakin avainsanoja käytettäväksi ilman, että testaajan tarvitsee ymmärtää todella teknisiä asioita tai osata välttämättä ohjelmoida. Avainsanat pyritään aina nimeämään siten, että ei-tekninenkin ihminen ymmärtää mitä ne tekevät. Tämä on tärkeää, kun suunnittelussa on mukana sidoshenkilöitä.

Kirjastojen käyttöönotto on helppoa. Testitiedoston Settings-osioon kirjoitetaan Library ja sen perään kirjaston nimi. Jos kyseessä on Robot Frameworkin mukana tullut kirjasto, voidaan kirjasto ottaa käyttöön asettamalla sen nimi Library-asetuksen perään ilman viittausta kirjaston osoitteeseen. (Robot Framework User Guide, 2017). Mikäli kyseessä on

ulkoinen kirjasto, täytyy kirjasto siirtää haluttuun kansioon, ja sen jälkeen tiedoston osoite asettaa kirjaston nimen perään (KUVA 11).

```
*** Settings ***
Library     Collections
Library     lib/ExternalLibrary.py
```

KUVA 11. Robot Framework kirjaston asettaminen

Robot Frameworkin mukana tulee kahdeksan kirjastoa: Collections, Dialogs, Operating System, Process, Screenshot, String, Telnet; XML. (Bisht 2013, 47) Kaikista kirjastoista löytyy eri tarkoituksiin käytettäviä avainsanoja. Esimerkiksi Collections pitää sisällään listojen ja kirjaston käsittelyssä käytettäviä avainsanoja, kun taas Operating System käyttöjärjestelmään liittyviä. Kirjastoja ja avainsanoja on tarjolla ilmaiseksi myös verkossa.

4.7 Pystytys ja alasajo

Kaksi oleellista tekijää testien ajojen kannalta ovat pystytys (setup) ja alasajo (teardown). Pystytys pitää sisällään testin käynnistyessä ensimmäisenä suoritettavat asiat. Alasajo taas pitää sisällään testin lopussa tai sen epäonnistuessa suoritettavat asiat. Pystytyksessä määritellään yleensä ehtoja kuten, mihin osoitteeseen halutaan mennä, millä käyttäjäroolilla kirjaudutaan käyttöjärjestelmään tai tiettyjen merkkien (tag) asettaminen testille. Määritellään siis toimenpiteitä, jotka halutaan tehdä testien alussa tai lopussa. Toimenpiteet voivat olla sekä testikohtaisia, että useammin käytettäviä. Käyttötapauksesta riippumatta pystytystä ja alasajoa varten kannattaa luoda omia avainsanoja.

Alasajoon (teardown) halutaan laittaa pääsääntöisesti kahdenlaisia toimenpiteitä, epäonnistumisesta tietoja keräämiä toimenpiteitä ja testiympäristön sulkemiseen liittyviä toimenpiteitä. Sulkemiseen liittyvät toimenpiteet voivat olla esimerkiksi järjestelmästä ulos kirjautuminen, selaimen sulkeminen tai etusivulle palaaminen. Näillä toimenpiteillä pyritään varmistamaan se, että seuraavat suoritettavat testit käynnistyvät oikein, jos edellinen testi on epäonnistunut, eikä ei ole suoriutunut normaalisti loppuun.

Myös pystytyksessä on hyvä huomioida tilanteet, joissa edellisen testin ajo ei ole suoriutunut loppuun. Voidaan esimerkiksi tarkastaa selaimen sijainti ja tehdä toimenpiteitä, mikäli uuden testin käynnistyessä sijainti ei olekaan kotisivu.

4.8 Web-käyttöliittymä

Yksi tärkeimmistä asioista onnistuneen testin kirjoittamisessa web-käyttöliittymälle on, että saa eri HTML-elementeistä kiinni. Elementeistä kiinniottaminen tarkoittaa siihen viittaamista (locator) Selenium-kirjaston tukemalla tavalla. Selenium-kirjasto tukee lukuisia eri tapoja viitata elementteihin. Kuvassa 12 näkyy kaikki Selenium-kirjaston tukemat menetelmät viitata elementteihin.

Strategy	Example	Description
identifier	Click Element | identifier=my_element	Matches by @id or @name attribute
id	Click Element | id=my_element	Matches by @id attribute
name	Click Element | name=my_element	Matches by @name attribute
xpath	Click Element | xpath=//div[@id='my_element']	Matches with arbitrary XPath expression
dom	Click Element | dom=document.images[56]	Matches with arbitrary DOM express
link	Click Element | link=My Link	Matches anchor elements by their link text
partial link	Click Element | partial link=y Lin	Matches anchor elements by their partial link text
css	Click Element | css=div.my_class	Matches by CSS selector
class	Click Element | class=my_class	Matches by class name selector
jquery	Click Element | jquery=div.my_class	Matches by jQuery/sizzle selector
sizzle	Click Element | sizzle=div.my_class	Matches by jQuery/sizzle selector
tag	Click Element | tag=div	Matches by HTML tag name
default*	Click Link | default=page?a=b	Matches key attributes with value after first '='

KUVA 12. Selenium-kirjaston tukemat viittaukset (Robot Framework)

Yksinkertaisin tapa viitata elementtiin on id. Id:llä määritelty elementti ei tarvitse mitään muita määrittelyjä kuin nimen, jotta siihen voidaan tehdä viittaus (locator) (KUVA 13). Kuvan esimerkissä suoritetaan painallus elementtiin, jonka id on etusivu.

```
Yksinkertainen viittaus
    Click Element    etusivu
```

KUVA 13. Id:llä määriteltyyn elementtiin viittaus

Tärkeää muistaa ohjelmistoa kehittäessä, että noudattaa HTML-elementtien nimeämisen perusteita. Id on aina uniikki nimitys yhdelle tietylle HTML-elementille. Class eli luokka voi olla nimitys useammalle HMTL-elementille, joille halutaan samat ominaisuudet.

Selenium-kirjaston viittauksilla (locator) on mahdollista viitata lähes mihin tahansa elementtiin, vaikka se olisi huonosti nimetty, mutta oikean viittauksen löytäminen voi olla tuskallista ja aikaa vievää.

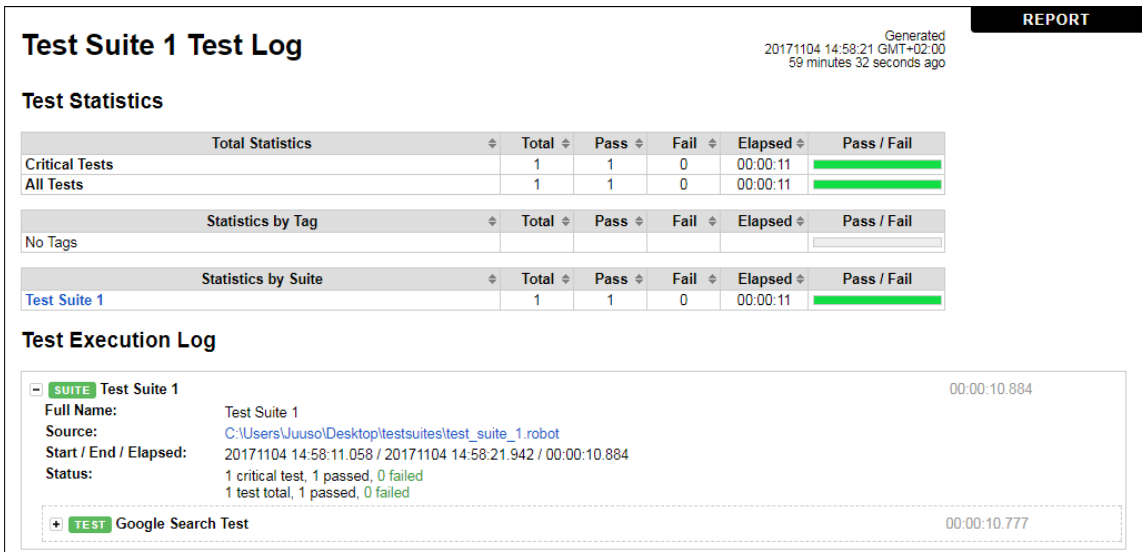
Jos elementtejä ei ole nimetty käyttöliittymässä, on mahdollista viitata elementteihin käyttäen Xpath-viittausta (locator). Xpath-viittaus ottaa huomioon käyttöliittymän elementtien rakennepuun. Tämä on kuitenkin joissakin tilanteissa huono tapa, sillä jos käyttöliittymään tehdään muutoksia ja rakennepuu vaihtuu, ei viittaukset enää välttämättä viittaa oikeaan elementtiin. Kun käyttöliittymän elementit nimetään hyvin, tekee se testien kirjoittamisesta paljon helpompaa ja testeistä tulee luotettavampia sekä ymmärrettävimpiä eikä rakenteen muuttuessa synny ongelmia. Se on yksi painavimmista syistä kehittää testit ja ohjelmisto yhdessä. Kun testien kehittäjällä on helppo tapa viitata elementteihin, ja tieto siitä miten ohjelmisto toimii, testeistä tulee paremmat vähemmällä vaivalla ja samalla ohjelmisto tulee testattua paremmin.

Kun testejä tehdään jo olemassa olevaan tuotteeseen, saattaa olla, että elementeillä ei ole tarpeellisia nimiä tai nimeäminen on puutteellista. Tässä tilanteessa olisi kannattavampaa luoda tuotteeseen muutokset kuin tehdä testiä rajallisilla työkaluilla.

4.9 Testitulosten analysointi

Robot Framework luo testien ajamisen jälkeen kolme tiedostoa. XML-tiedoston, joka on tarkoitettu laitteiston luettavaksi sekä loki- ja raporttitiedostot, jotka ovat HTML-muodossa ja tarkoitettu suoraan testitulosten tarkasteluun. (Robot Framework User Guide, 2017)

Lokitiedostot pitävät sisällään yksityiskohtaiset tiedot suoritetusta testistä. Siitä näkee testisarjan (test suite), testitapauksen (test case) ja avainsanakohtaiset (keyword) tiedot (KUVA 14). Kun testejä kehitetään, lokitiedostoa tarvitaan lähes aina. Raporttitiedostot ovat parempia antamaan kokonaiskuvan suoritetuista testeistä. Niissä on statistiikkaa kategorisoituna suoritettujen testisarjojen ja merkkien (tag) mukaan. (KUVA 15) (Robot Framework User Guide, 2017)



REPORT

Test Suite 1 Test Log

Generated
20171104 14:58:21 GMT+02:00
59 minutes 32 seconds ago

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:11	
All Tests	1	1	0	00:00:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Test Suite 1	1	1	0	00:00:11	

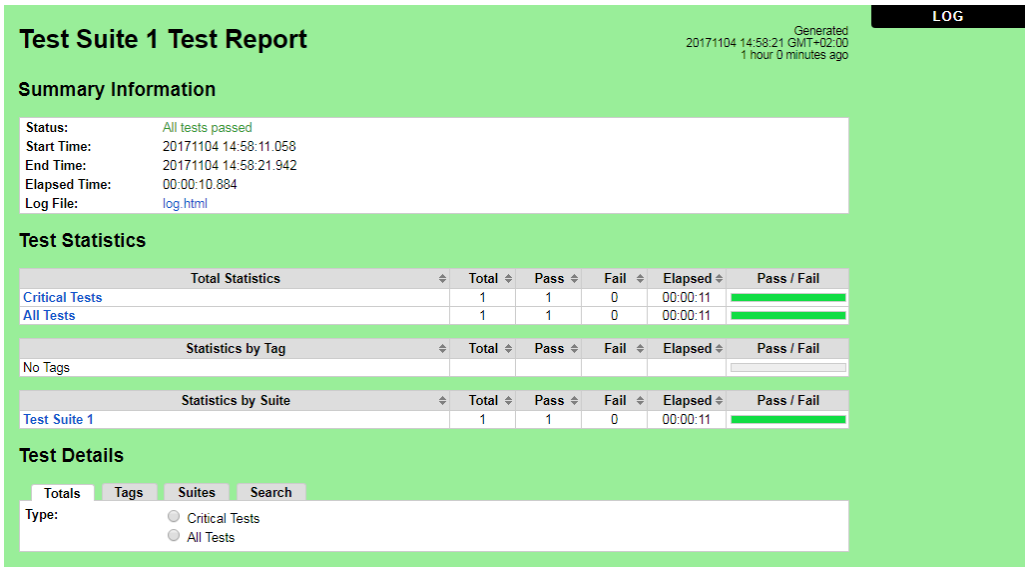
Test Execution Log

SUITE Test Suite 1 — 00:00:10.884

Full Name: Test Suite 1
Source: C:\Users\Juuso\Desktop\testsuites\test_suite_1.robot
Start / End / Elapsed: 20171104 14:58:11.058 / 20171104 14:58:21.942 / 00:00:10.884
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST Google Search Test — 00:00:10.777

KUVA 14. Onnistuneen testin lokitiedosto



LOG

Test Suite 1 Test Report

Generated
20171104 14:58:21 GMT+02:00
1 hour 0 minutes ago

Summary Information

Status: All tests passed
Start Time: 20171104 14:58:11.058
End Time: 20171104 14:58:21.942
Elapsed Time: 00:00:10.884
Log File: log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:11	
All Tests	1	1	0	00:00:11	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Test Suite 1	1	1	0	00:00:11	

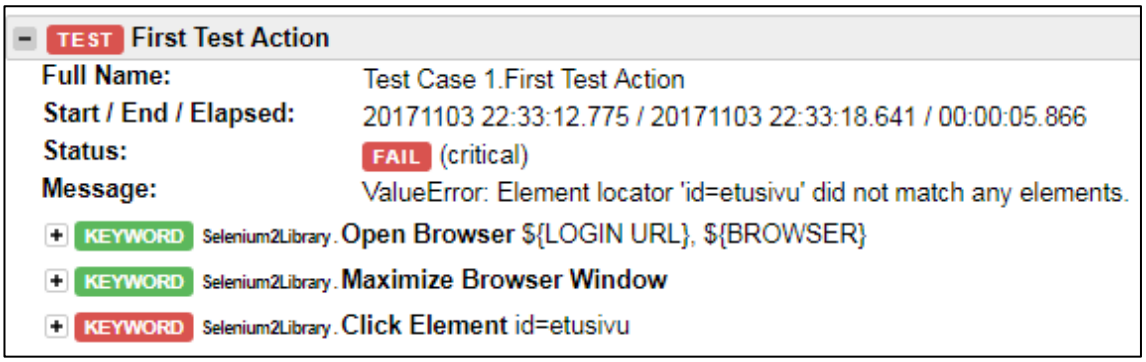
Test Details

Totals | Tags | Suites | Search

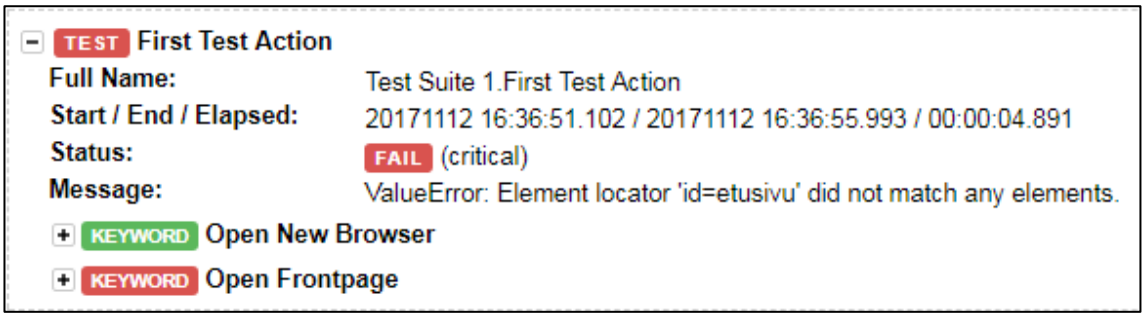
Type: Critical Tests / All Tests

KUVA 15. Onnistuneen testin raporttitiedosto

Hyvin nimetyt avainsanat auttavat myös virheiden analysoimisessa. Otetaan esimerkkinä kaksi tapaa tehdä sama asia. Ensin tehdään samat toiminnallisuudet ilman omia avainsanoja ja sitten omilla avainsanoilla. Molemmissa esimerkeissä etusivun avaaminen on epäonnistunut. Ilman avainsanoja toteutetun testin tuloksista nähdään, että elementin painaminen on epäonnistunut, mutta ei nähdä, että missä yhteydessä se on tapahtunut (KUVA 16). Kun taas omilla avainsanoilla toteutetusta testistä, näkee nopeasti, että epäonnistunut painallus on tapahtunut etusivulle mennessä (KUVA 17).



KUVA 16. Testi toteutettuna ilman omia avainsanoja



KUVA 17. Testi toteutettuna omilla avainsanoilla

5 TESTAUS JA TUOTEKEHITYS

5.1 Johdanto

Ohjelmiston testauksella on monia syitä. Virheiden löytäminen, laadun mittaaminen, luottamuksen nostaminen ja ohjelmiston analysointi virheiden välttämiseksi. (Spillner 2014, 9.) Ohjelmistojen mittakaava ja monimutkaisuus on kasvanut ja näin myös tarve laadun ja luotettavuuden takaamiselle. Manuaaliset testit ovat usein helppoja tehdä, mutta mittakaavan kasvaessa niiden soveltuvuus testaukseen laskee, koska testaaja pystyy manuaalisella työllä vain tiettyyn pisteeseen löytämään vikoja ja virheitä vaikuttamatta testin lopputulokseen. (Bisht 2014, 8.) Koska manuaalinen testaus vaatii äärimmäistä tarkkuutta niin jo yhdenkin asian tarkastamisen unohtaminen vaikuttaa lopputulokseen. Siitä syystä sovelluksen mittakaavan kasvaessa manuaalisessa testauksessa jää helpommin asioita testaamatta.

Luotettavuus määrittää ohjelmiston kyvyn pysyä toiminnassa tietynlaisen käytön alla. Luotettavuuden määrittelee ohjelmiston kypsyys, joka jaetaan kahteen osaan, viansietokykyyn ja palautumiseen. Viansietokyky tarkoittaa ohjelmiston kykyä ylläpitää tietyn toiminnan tason, kun tapahtuu ympäristövirheitä, käyttöliittymän väärinkäyttöä tai käyttäjät antavat vääriä syötteitä. Palautuminen tarkoittaa kykyä palautua normaaliin toiminnan tasoon sen jälkeen, kun virhe on tapahtunut. (Spillner 2014, 12.)

Vaikka testaus on todella tärkeää, kuitenkaan kaikkia ominaisuuksia ei ole välttämätöntä testata. On tapauksia, joissa uusi ominaisuus on niin pieni, että sille ei ole kannattavaa tai tarpeellista tehdä testiä. Pieniä ominaisuuksia ovat esimerkiksi kosmeettiset muutokset tuotteessa, jotka eivät vaikuta suoraan sen toiminnollisuuksiin. Voi olla myös, että uusi ominaisuus ei ole kriittinen järjestelmän käytön ja toiminnan kannalta, joten yksinomaan sille testin rakentaminen ei ole kannattavaa.

Testaus ei todista virheiden olemattomuutta. Jotta voitaisiin testata kaikki mahdolliset virheet, pitäisi tietää kaikki mahdolliset tilanteet, kaikki mahdolliset käyttäjän antamat syötteet ja ottaa huomioon kaikki eri olosuhteet. (Spillner 2014, 13.)

Tästä syystä testaukseen käytetty vaiva pitää suhteuttaa saavutettavissa oleviin tuloksiin.

Testausta pitää jatkaa niin kauan, kun vian etsimisen ja korjauksen hinta on pienempi kuin vikatilanteen syntymisestä seuraavat kulut (Koomen 1999). Testaukseen käytettävä aika on aina riippuvainen ohjelmiston riskiarviosta. Ohjelmisto, jonka vioista koituu suuria tappioita, tulee testata huolellisemmin kuin ohjelmisto, jonka vioista ei synny isoja tappioita.

Kaikkien firmojen ei siis tarvitse panostaa testaamiseen suhteessa yhtä paljon kuin muiden, jos riskiarvion tekemisen jälkeen on käynyt selväksi, että tuotteen mahdollisista vioista ei koidu niin suuria tappioita, että sen hetkiseen testauskäytäntöön kannattaisi laittaa enemmän rahaa.

5.2 Testauksen tyypit

Testaamista on useaa eri tyyppiä. Opinnäytetyössä käydään läpi yleisimmät testauksen tyypit, jotka ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus, hyväksymistestaus ja regressiotestaus.

Yksikkötestaus on ensimmäinen osa ohjelmiston yksiköiden eli lähdekoodin osien testaamista. Yksikkötestauksen tärkein tarkoitus on testata yksittäisen testikohteen toiminnallisuus. Testataan, että se toimii oikein ja täysin niin kuin määrittelyssä on sanottu. Toiminnallisuus tarkoittaa testikohteen syötteen ja ulostulon käyttäytymistä. (Spillner 2014, 42-48.)

Integraatiotestaus on testaamisen toinen vaihe. Tässä vaiheessa komponentit ovat valmiita ja on aika yhdistää ne. Testataan, että komponentit toimivat yhdessä oikein. Tavoitteena on löytää mahdollisia virheitä, joita tapahtuu, kun komponentin toimivat yhdessä. (Spillner 2014, 50-55.)

Kun integraatiotestaus on saatu suoritettua, siirrytään järjestelmätestaukseen. Järjestelmätestauksen tavoite on varmistaa, että järjestelmä toimii loppukäyttäjälle niin kuin on haluttu. Vaikka integraatiotestaus kattaa komponenttien toimivuuden keskenään, saattaa vuorovaikutus järjestelmän kaikkien komponenttien kanssa tuoda esiin vikoja ja tästä syystä järjestelmätestaus on tärkeää. (Spillner 2014, 58-60.)

Hyväksymistestaus on viimeinen vaihe ennen kuin ohjelmisto julkaistaan asiakkaalle. Sen tavoite on testata ohjelmistoa asiakkaan näkökulmasta. Testin pohjana voidaan käyttää mitä tahansa dokumenttia, joka kuvaa ohjelmiston käyttöä asiakkaan näkökulmasta. Hyväksymistestaus on mahdollista suorittaa myös oikeilla asiakkailla, jolloin asiakkaat käyttävät ohjelmistoa ja validoivat sen toimivuuden. (Spillner 2014, 61-64.)

Ohjelmistoon tullessa muutoksia, halutaan varmistaa, että mikään olemassa oleva toiminnallisuus ei hajoa. Tällöin toistetaan muutokseen suoraan liittymättömiä testejä, jotta mahdolliset viat löydettäisiin. Regressiotestaukseen käytettävien testien tulee olla hyvin dokumentoituja ja uudelleenkäytettäviä. Tämän johdosta ne ovat hyviä vaihtoehtoja automaatiotesteiksi.

5.3 Testien suunnittelu

Testien suunnitteleminen käyttöliittymälle lähtee aina testattavan ominaisuuden toiminnollisuuksien selvittämisestä. Toiminnallisuutta kuvaamaan on hyvä luoda aitoja käyttötapauksia eli skenaarioita siitä, kuinka lopullinen käyttäjä tulee sitä käyttämään. Niiden avulla on helpompi hahmottaa, miten ja mihin ominaisuutta käytetään. Kun tiedetään tapa, jolla ominaisuutta käytetään, on helppo rakentaa käyttötapausta mukaileva runko testille. Kun testin runko on rakennettu, on siihen helppo lisätä tarkistuksia, jotka eivät ole niin kriittisiä käyttötapauksen kannalta, mutta parantavat testin laatua.

Testattavan ominaisuuden monimutkaisuudesta ja koosta riippuen testaukseen voi olla hyvä rakentaa useampi testi. Jos ominaisuudesta syntyy useita erilaisia käyttötapauksia, saattaa olla kannattavaa tehdä useampia testejä, joista jokainen käy tietyn käyttötapauksen toiminnallisuudet läpi. Tämä helpottaa testin epäonnistuessa vian paikantamista ja ominaisuuden muuttuessa tiettyä käyttötapausta on helppo muuttaa. On myös hyvä muistaa, että usean testin tekeminen samasta ominaisuudesta voi myös johtaa siihen, että kun ominaisuutta muokataan paljon, saattaa joutua muokkaamaan useampaa testiä erikseen ja siihen kuluu enemmän aikaa. Tästä syystä yhteistyö kehittäjän kanssa on äärimmäisen tärkeää. Se, että ohjelmiston kehittäjä on suunnitellut ja toteuttanut uuden ominaisuuden helposti testattavaksi, säästää paljon aikaa ja vaivaa.

5.4 Testien kirjoittaminen

Testien kirjoittaminen voidaan aloittaa, kun ohjelmiston kehittäjällä ja testien rakentajalla on yhteinen ymmärrys siitä, miten asioiden tulee toimia. Yhteisymmärrys saadaan, kun osapuolet käyvät kehitettävän toiminnallisuuden käyttötapaukset läpi yhdessä.

Robot Frameworkilla testejä tehtäessä suositellaan käytettävän ylhäältä-alaspäin-menetelmää, mikä tarkoittaa sitä, että testin rakenne luodaan ensin ja sen jälkeen konkreettinen käytäntöön pano. Vaikka tämä ei ole pakollista, helpottaa se isojen testien kirjoittamista ja on sen johdosta suositeltu tapa testien kirjoittamiseen. (Bisht 2013, 21.) Rakenteen luomisella tarkoitetaan sitä, että esimerkiksi avainsanat (keyword) kirjoitetaan järjestyksessä ja merkityksessä valmiiksi testirunkoon ja sen valmistuttua, kirjoitetaan sisältö avainsanoihin. Merkitys tulee avainsanan tehtävästä testissä. Sen tarkoitus on kuvata mahdollisimman selkeästi omaa toiminnallisuuttaan.

Testin rakentaminen on hyvä aloittaa siitä tilanteesta, kun käyttäjä kirjautuu sisälle tuotteeseen. Tällä varmistetaan, että käyttötilanteesta riippumatta testi pystyy tekemään tarpeelliset asiat. Toisistaan riippuvaisiakin testejä on mahdollista tehdä. Niissä ensimmäinen testi vastaa tietystä osuudesta ja seuraava toisesta, mutta jommankumman testin epäonnistuessa jää testauksen ulkopuolelle asioita ja mahdollisia virheitä löytämättä, jotka olisi voitu erillisillä testeillä huomata.

Testi kannattaa toteuttaa järjestelmällisesti alusta loppuun tuotteen toiminnollisuuksien ja käyttötapauksien mukaan. Kehityksessä oleva testi kannattaa suorittaa säännöllisin välein ja tarkistaa, että se toimii halutulla tavalla. Jälkeenpäin löydettyjen ongelmien syiden paikantaminen ja ratkaisu vie enemmän aikaa, kuin testin säännöllinen suorittaminen testejä kirjoittaessa. Palaaminen takaisinpäin hidastaa kehitystä huomattavasti ja pahimmassa tapauksessa ongelman takia myös testin rakennetta tarvitsee muuttaa.

5.5 Tuotteen testaus

Kun testauksessa löytyy virheitä, tulee testaajan tarkastella testituloksia ja päätellä, johtuvatko virheet testistä vai onko vika ohjelmistossa. Jos vika on ohjelmistossa, se tulee

dokumentoida ja arvioida nopeasti, mistä se saattaa johtua. Vian löytyminen vaatii testaajalta tarkennusta testiin ja mahdollisesti testaaja joutuu ajamaan useampia testejä virheen todentamiseksi. (Spillner 2014, 26.)

Testaajaan tulee voida luottaa. Jos testaaja ilmoittaa ohjelmiston viasta, mutta vika onkin hänen testissään, laskee muiden ihmisten luottamus häneen. Testaajan ei kuitenkaan tule sensuroida tuloksia eikä pelätä mahdollisista vioista ilmoittamista. (Spillner 2014, 27.)

Kun tuotteesta on löytynyt vika, siitä on tehty vikailmoitus ja se on korjattu, tulee testaajan tarkastaa, että vika on oikeasti korjaantunut eikä uusia vikoja ole syntynyt. (Spillner 2014, 27). Tarkastaakseen, että vika on korjaantunut, testaaja toistaa alkuperäisen käyttötapauksen, jolla vika löytyi. Jos käyttötapauksesta ei enää löydy virhettä, voidaan vikailmoitus sulkea. (Quality Control, 39.) Tarkastukseen voidaan myös käyttää testiä, joka alun perin löysi vian, kun testi on muokattu vastaamaan korjattua ohjelmistoa. Tarpeen mukaan myös uusia testejä tarvitaan, jos ohjelmiston lähdekoodi on muuttunut. (Spillner 2014, 27.)

Yrityksen ja ohjelmiston koosta riippuen jokaisen vian ilmoittaminen erikseen kehittäjille ei ole välttämättä järkevä tapa. Yksittäisistä virheistä ilmoittaminen veisi todella paljon aikaa molemmilta osapuolilta. Jos yrityksessä on erillinen testaaja, tulee kalliiksi odottaa, että kehittäjä korjaa vian ja vasta tämän jälkeen testaaja jatkaa testausta. Ominaisuus, josta vika löytyy, tulisi testata huolellisesti ja mikäli siitä löytyy useampi vika, tulisi vioista ilmoittaa kehittäjille samalla kerralla. (Spillner 2014, 27.)

Uutta ohjelmiston ominaisuutta ei voida aina testata riittävästi jokaisessa projektissa. Aika ei välttämättä riitä tai rahat voivat olla vähissä. Tällaisessa tapauksessa valitaan kohtuullinen määrä testejä, joiden tarkoituksena on varmistaa, että kaikkein kriittisimmät viat, joita voi syntyä, huomataan. Tilannetta kutsutaan riskipohjaiseksi testaamiseksi. Riskipohjaisella testaamisella on hyötyjäkin. Koska kriittisimmät mahdolliset viat arvioidaan vakavuusjärjestykseen, huomataan ne ensiksi. (Spillner 2014, 27.)

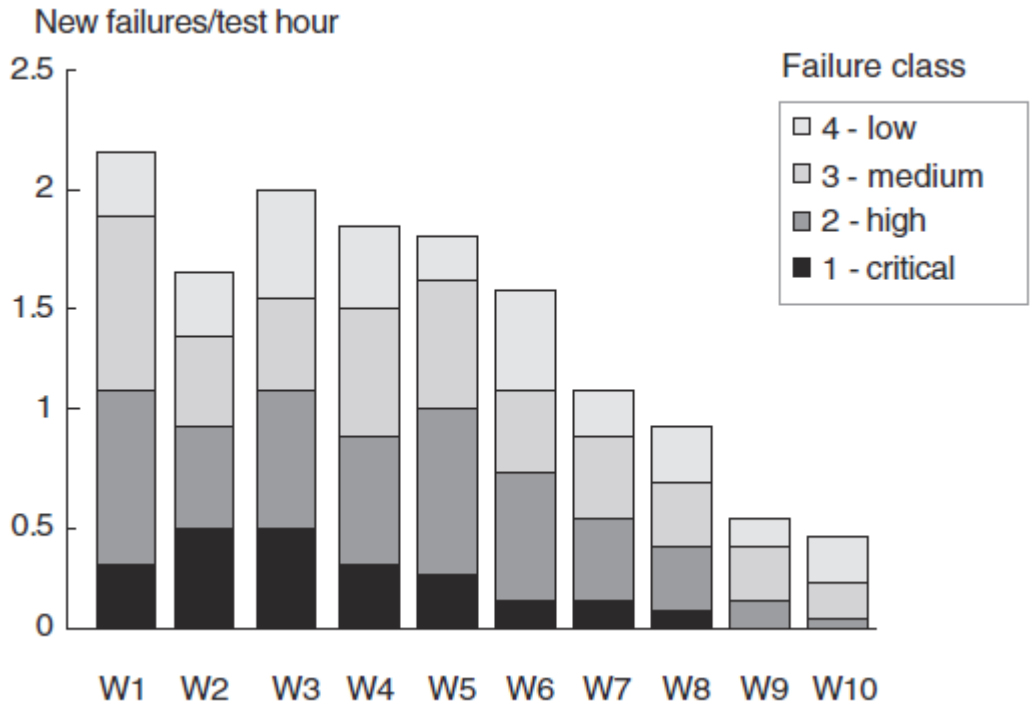
5.6 Hyväksytty testaus

Jokaiselle testattavalle ominaisuudelle tulee määritellä vaatimukset, joiden täytyttyä se on hyväksytysti testattu. Ohjelmisto tai sen ominaisuus on epäonnistunut, jos käyttäjän odotuksia ei täytetä. Esimerkki tällaisesta epäonnistumisesta on tuote, jota on liian vaikea käyttää tai se on liian hidas, mutta se täyttää toiminnalliset vaatimukset. (Spillner. 2014, 28.)

Mikäli jokin vaatimus ei ole toteutunut, se tulee testata, jotta voidaan varmistua vaatimusten täyttymisestä. Tällaisessa tilanteessa tulee pitää huolta, että uusi toteutettava testi varmasti testaa juuri vaaditun asian, koska muuten aikaa on käytetty ylimääräiseen työhön, joka ei ole tuonut mitään parannusta aikaisempaan. (Spillner 2014, 28.)

On mahdollista, että testaamattoman vaatimuksen testaaminen todetaan arvioinnin jälkeen liian työlääksi. Näin voi käydä, jos mahdollisista vioista koituvat kulut, eivät ylitä testaukseen tarvittavia kuluja. Tässä tilanteessa lisätestaus lopetetaan ja vaatimukset todetaan täytetyiksi. Testaamatta jättäminen on aina riski, mutta riittäviä työkaluja ja testiympäristöä ei ole mahdollista aina toteuttaa järkevällä kustannuksella. Joissakin tapauksissa päätetään testata lisää ja aloittaa testaaminen uudestaan. Silloin voi olla paikallaan päivittää testikriteereitä, jotta vaaditut kriteerit täyttyvät. (Spillner 2014, 28-29)

Testauksen hyväksyttämiselle voidaan asettaa muitakin kriteereitä kuin tiettyjen vaatimusten täyttäminen. Hyväksyttämiseen voidaan käyttää löydettyjen virheiden määrää. Kuvio 18 näyttää keskimääräisen virheiden määrän tuntia kohden kymmenen viikon aikana. Ensimmäisen viikon aikana on löytynyt keskimäärin kaksi virhettä testaamisen kulunutta tuntia kohden. Kymmenennellä viikolla virheitä on löytynyt vähemmän kuin yksi virhe kulunutta kahta tuntia kohti. Jos virheiden määrä putoaa näin alas, voidaan todeta, että lisätestaaminen ei ole enää rahallisesti perusteltavissa ja testaus voidaan lopettaa. (Spillner 2014, 29.)



KUVIO 18. Keskimääräinen virheiden määrä per tunti (Spillner 2014, 29)

5.7 Testien ylläpito

Kun tuotetta kehitetään jatkuvasti, myös testauksen pitää pysyä mukana. Jos tuote muuttuu, niin myös sitä arvioivat testit tarvitsee muuttaa sopimaan päivitettyyn tuotteeseen. Eri testeille pitäisi löytyä uudelleen käytettäviä komponentteja, joita tulisi muokata testien mukaan. Muutosten tarve on hyvä selvittää jo ennen kuin tuotetta muutetaan, koska tällöin se voidaan muokata vastaamaan uutta tuotetta mahdollisimman nopeasti. Jos testejä ei päivitetä jatkuvasti tuotteen mukana, tuotteen luotettavuus laskee ajan myötä, eikä se kehity niin kuin pitäisi (Spillner 2014, 69).

Testisuunnitelma eli riskiarvioinnin ja testien vaatimuksien pohjalta luotu toimintapa, vaatii myös säännöllistä päivittämistä. Testisuunnitelman tulee muuttua, kun projektin riskit muuttuvat ja testauksesta tulee palautetta. Testien määrän kasvaessa, voi ylläpidosta tulla ongelmia, kun tuotetta kehitetään ja joudutaan muokkaamaan lukuisia testejä. Voi käydä niin, että kehitettävän asian työmäärään ei osata ottaa tarpeeksi tarkasti huomioon useiden testien muokkaamiseen menevää aikaa ja näin myöhästytään alkuperäisestä tavoitteesta tai tuotetta ei ehditä testata riittävän huolellisesti.

6 POHDINTA

Opinnäytetyön tavoite oli selvittää, kuinka automaatiotestien rakentaminen tulisi toteuttaa web-käyttöliittymälle osana yrityksen tuotekehitystä. Robot Framework osoittautui toimivaksi työkaluksi automaatiotestien rakentamiseen ja web-käyttöliittymän testaukseen. Sen käyttöönottaminen voi olla hieman työlästä, jos ei ole aikaisempaa kokemusta komentorivin käytöstä tai pakettien asentamisesta. Kuitenkin käyttöönottamisen jälkeen, kun pääsee kirjoittamaan testejä, on sillä testien kirjoittaminen melko suoraviivaista ja selkeää. Hyviä puolia on, että Robot Framework on avoimen lähdekoodin projekti eikä se ole ohjelmistoriippuvainen. Siitä löytyy myös kattava valikoima eri kirjastoja ilmaiseksi ja se on laajennettavissa Pythonin avulla.

Testaus on tärkeä osa onnistunutta tuotekehitystä ja se nostaa tuotteen laatua. Testaus pitää sisällään paljon muitakin vaiheita kuin pelkän fyysisen toimenpiteen. Automaatiotestauksella voidaan vähentää manuaalisen työn määrä ja siten säästää kuluissa. Vaikka automaatiotestit vähentävät manuaalisen työn määrää, eivät ne kuitenkaan poista sitä kokonaan ja testien ylläpitoon on syytä varata aikaa. Testien suunnittelu ja toteutus on syytä tehdä huolella, jotta niiden korjaamiseen ja ylläpitoon ei kulu niin paljon aikaa. Testien kehittäminen ja päivittäminen on hyvä tehdä tuotekehityksen rinnalla ja yhteistyössä kehittäjien kanssa.

Opinnäytetyössä ei ollut mahdollista paneutua kaikkiin haluamiini asioihin riittävän kattavalla tasolla. Olisi ollut mielenkiintoista käydä läpi, kuinka automaatiotestit saisi käyttöliittymälle säännölliseen ajoon, ja kuinka testien tulosten keräämisen voisi toteuttaa. Esimerkiksi Travis CI ja Jenkins ovat työkaluja, joilla on mahdollista toteuttaa testien säännöllinen suoritus käyttöliittymälle. Myös Pythonilla avainsanojen luomisen prosessiin olisi voitu perehtyä, ellei se olisi itsessään niin laaja alue. Edellä mainittuihin työkaluihin ja Pythoniin kannattaa ehdottomasti tutustua tarkemmin.

LÄHTEET

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

Klärck, P. Eliga Oy. Luettu 15.11.2017.
http://eliga.fi/index.html

Klärck, P. Robot Framework Introduction. 2009.
https://www.slideshare.net/pekkaklarck/robot-framework-introduction

Koomen, T. Pol, M. 1999. Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing. Addison-Wesley.

Nursebuddy. Luettu 15.11.2017.
https://nursebuddy.co/fi/

Quality Control, Segue Technologies. 2014

Robot Framework. Luettu 20.11.2017.
http://robotframework.org/

Robot Framework User Guide. 2017. Luettu 20.11.2017.
http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Selenium WebDriver. SeleniumHQ, browser automation. Luettu 18.11.2017.
http://www.seleniumhq.org/docs/03_webdriver.jsp

Simon Stewart, Google Open Source Blog. 2009. Luettu 26.11.2017.
https://opensource.googleblog.com/2009/05/introducing-webdriver.html

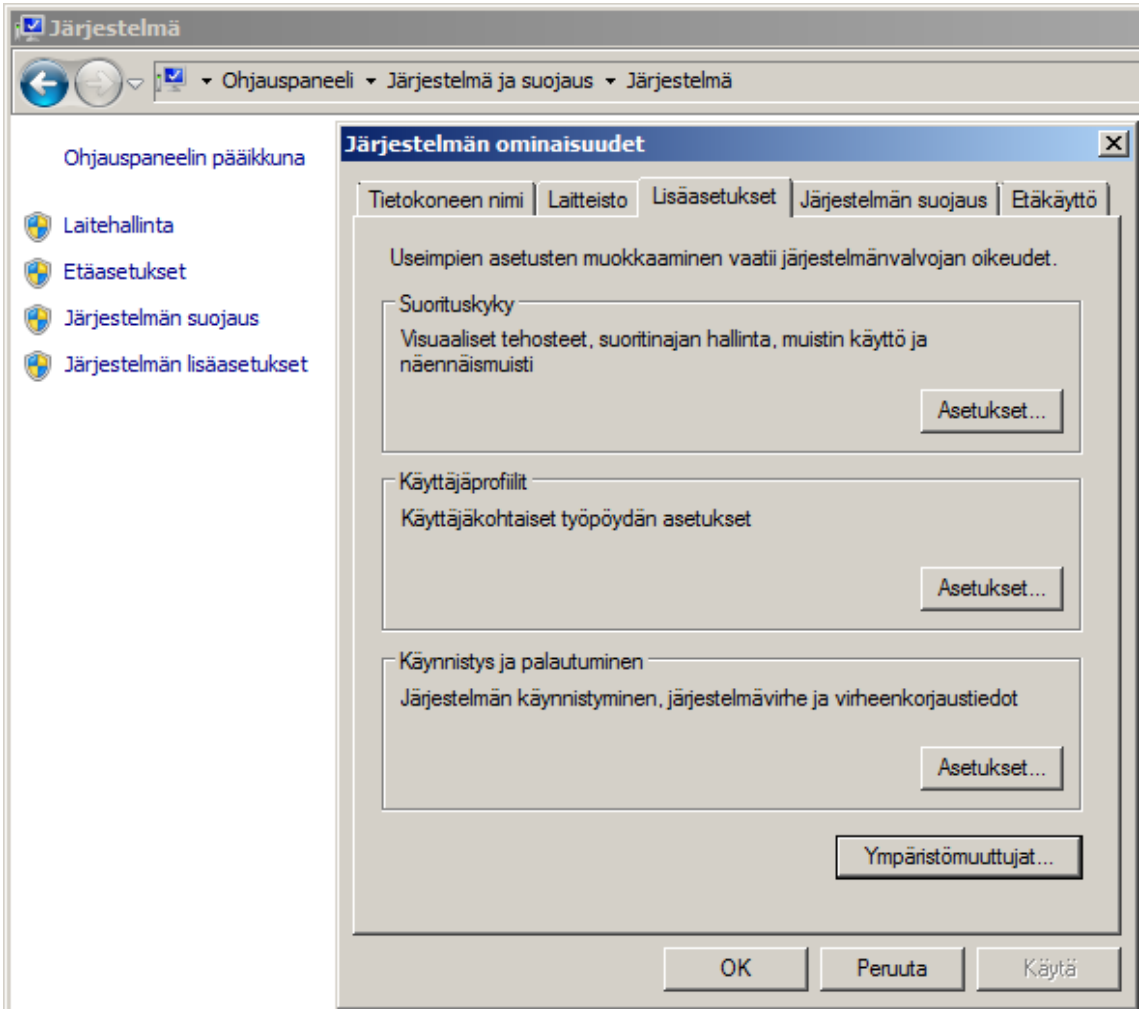
Spillner, A. Linz, T. Schaefer, H. 2014. Software Testing Foundations - 4th Edition. San-ta Barbara: Rocky Nook Inc.

Why automated testing? Smartbear. Luettu 23.11.2017.
https://support.smartbear.com/articles/testcomplete/manager-overview/

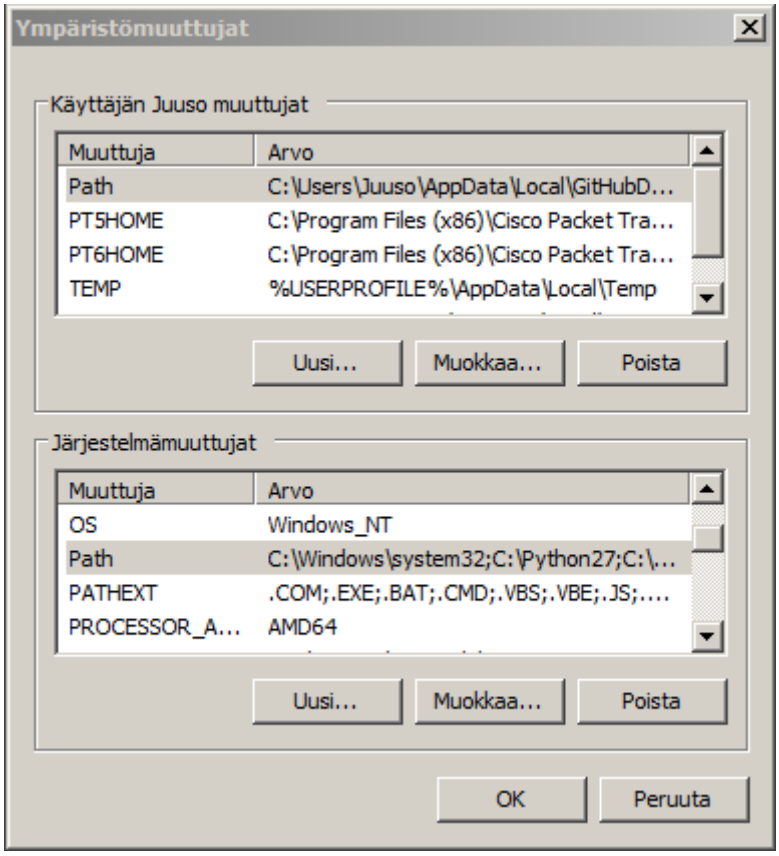
LIITTEET

Liite 1. Ympäristömuuttujan asettaminen Windows -käyttöjärjestelmällä

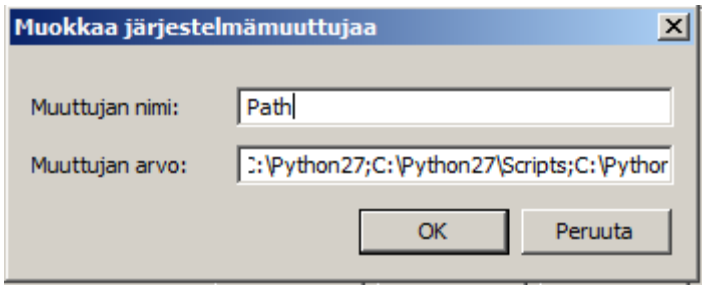
Ympäristömuuttujan asettaminen Windowsissa tapahtuu ohjauspaneelin kautta menemällä järjestelmä ja suojaus -osiolle, josta avataan järjestelmä. Järjestelmän auettua sivun vasemmasta laidasta löytyy järjestelmän lisäasetukset. Järjestelmän lisäasetuksista aukeaa alla näkyvä ikkuna:



Ympäristömuuttujia pääsee muokkaamaan ympäristömuuttujat -painikkeesta, josta aukeaa ympäristömuuttujat ikkuna. Valikosta löytyy kaksi eri listaa muuttujista, oman käyttäjän muuttuja ja järjestelmän muuttuja. Muokataan järjestelmämuuttujat-listasta löytyvää muuttujaa nimeltä *Path*, eli polku.



Polut erotetaan Windowsissa puolipisteellä. Laita polun perään puolipiste ja sen perään kopioi Pythonin asennuskansion polku (path) sekä Pythonin scriptien oma kansio. Jos Python asennettiin oletuskansioon, pitäisi polun olla *C:\Python27;C:\Python27\Scripts;*.



Liite 2. Google Chrome - ja Firefox -selaimien ajurien käyttöönotto

Turvallisin vaihtoehto on ladata ajureista viimeisimmät versiot, jos ei ole tietoa, että tarvitsee tietyn version. Viimeisimmät versiot löytyvät seuraavista osoitteista.
Firefox-ajuri (driver):
https://github.com/mozilla/geckodriver/releases
Google Chrome-ajuri:
https://sites.google.com/a/chromium.org/chromedriver/downloads

Ladatut ajurit siirretään haluttuun kansioon tai hakemistoon ja niiden sijaintien polut asetetaan järjestelmän ympäristömuuttujiksi. Opinnäytetyötyössä ajurit on laitettu python27 kansioon.
C:\Python27\geckodriver.exe;
C:\Python27\chromedriver.exe;

Ympäristömuuttujien asettaminen tapahtuu Windows 7-käyttöjärjestelmälle liitteen 1. ohjeiden mukaisesti.

Mac -käyttöjärjestelmällä ympäristömuuttujiksi asettaminen tapahtuu seuraavasti. Siirrä ladatut ajurit (driver) haluttuun kansioon tai hakemistoon ja lisää tiedostojen sijaintien polut tiedostoon *~/.bash_profile* sisään seuraavalla formaatilla: *export PATH=$PATH:oma/chromedriverpath/chromedriver:oma/geckodriverpath/geckodriver*. Esimerkissä oma tarkoittaa hakemistoa, johon tiedosto on siirretty.

Mac -käyttöjärjestelmässä tiedostoja voi muokata terminaalin kautta komennolla *nano tiedoston_nimi*. Mac -käyttöjärjestelmän ympäristömuuttujat on asetettu *~/.bash_profile* -tiedostoon.