Opinnäytetyö (AMK)

Tieto- ja viestintätekniikka | Terveysteknologia

2018

Saija Kaitio

ROBOT FRAMEWORKIN KÄYTTÖÖNOTTO JA MALLITESTIEN LUOMINEN

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto – ja viestintätekniikka | Terveysteknologia

2018 | 30 sivua

Saija Kaitio

ROBOT FRAMEWORKIN KÄYTTÖÖNOTTO JA MALLITESTIEN LUOMINEN

Työn tavoitteena oli kartoittaa ja pohjustaa toimeksiantajayritykselle toimivaa testiautomaatioympäristöä, joka vapauttaa testaajien resursseja muihin tehtäviin sekä nopeuttaisi testien ajoa.

Testiautomaatiotyökaluna käytettiin Robot Frameworkia, johon liitettiin Selenium2-kirjasto.

Opinnäytetyö jaettiin kahteen osaan. Teoriaosassa selvitettiin ohjelmistotestausta yleisesti, testauksen eri tasoja ja analysoitiin automaatiotestauksen etuja sekä haasteita.

Toisessa osassa kartoitettiin yrityksen tämän hetkistä testauskäytäntöä sekä selvitettiin, miten Robot Frameworkin, Selenium2-kirjaston sekä Pythonin asentaminen ja käyttöönotto onnistuu Windows 10-käyttöjärjestelmälle. Lisäksi opinnäytetyössä toteutettiin esimerkkitestejä yrityksen hoitajakutsujärjestelmän uudelle käyttöliittymälle.

Lopputuloksena Robot Frameworkia käytettiin onnistuneesti rajattuihin testitapauksiin. Laajempi käyttöönotto vaatisi enemmän aikaa ja resursseja joita tämän opinnäytetyön kirjoittamisen aikana ei onnistuttu järjestämään.

ASIASANAT:

Ohjelmistotestaus, testiautomaatio, Robot Framework, RIDE

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology | Health Informatics

2018 | 30 pages

Saija Kaitio

ROBOT FRAMEWORK INITIALIZATION AND CREATION OF MODEL TEST CASES

The goal of this thesis was to plan and underlay a working automation testing environment for the commissioning company's needs to free tester resources for other projects and speed up the testing process.

The Robot Framework was used for implementing automated tests together with the Selenium2Library.

Thesis is divided into two sections. The theoretical section introduces software testing and its different test levels. It also discusses the advantages and challenges of automated testing.

The practical part of this thesis examines the commissioning company’s existing testing processes and investigates the installation and initialization of the Robot Framework, Python and Selenium2Library to a Windows 10 operating system. In addition, a few test cases were written for the company's new user interface for nurse call system.

In conclusion, the Robot Framework can be used to write successful test cases for the company's needs. However, wider deployment requires more time and resources which could not be organized during this thesis’ timeframe.

KEYWORDS:

Software testing, test automation, Robot Framework, RIDE

SISÄLTÖ

KUVAT

KÄYTETTY SANASTO

IronPython	Pythonin .NET-toteutus
Jython	Pythonin Java-toteutus
Pip	Sovellus, jolla pystytään asentamaan Python-paketteja suoraan Python Package Indexistä.
Python	Tulkattava ohjelmointikieli
RIDE	Editori joka on suunniteltu Robot Frameworkin testien kirjoittamiseen ja ajamiseen.
Robot Framework	Python-pohjainen, avainsanoihin perustuva testiautomaatiokehys
Selenium2Library	Robot Frameworkin avainsanakirjasto

1 JOHDANTO

Opinnäytetyön idea syntyi suorittaessani harjoittelua toimeksiantajayrityksessä. Työn tavoitteena oli kartoittaa testiautomaation mahdollisuuksia yritykselle sekä pohjustaa, miten automaatioprosessin saisi alkuun yrityksessä.

Työn toimeksiantaja on suomalainen yksityinen terveysteknologiaan erikoistunut yritys. Yritys toimii kansainvälisesti useassa maassa ja sen ratkaisuja käyttävät niin yksityiset toimijat kuin julkisyhteisötkin.

Yrityksessä työskentelee erillinen pieni testausryhmä, joka testaa uudet ohjelmistoversiot ennen niiden julkaisemista. Tämä tapahtuu suurimmaksi osaksi manuaalisesti, joka vie paljon testaajien aikaa sekä hidastaa yrityksen tuotantotehoa.

Testiryhmä suunnittelee uusille ominaisuuksille ja tuotteille testisuunnitelman ja testitapaukset. Kun uusi ohjelmistoversio tulee testattavaksi, suoritetaan testitapaukset ja kirjataan testitulokset ylös testiraporttiin. Testituloksia verrataan aikaisemman version tuloksiin. Mikäli testitulokset ovat halutun kaltaiset, laitetaan ohjelmisto tuotantoon, muuten takaisin kehitysosastolle. Jokainen löydetty virhe raportoidaan kehittäjille. Testiryhmä pyrkii ajattelemaan käyttöä loppukäyttäjän kannalta ja raportoi myös havaitut puuteet kehittäjille.

Yritys haluaisi alkaa automatisoimaan testejä, mutta sen tutkimiseen ja käyttöönottoon ei ole työntekijöillä ollut juuri aikaa. Yrityksen testauskohteita ovat muun muassa web-käyttöliittymät eri järjestelmiin, mobiilisovellus, eri järjestelmien palvelimet, sulautetut laitteet ja näiden toimiminen kokonaisuutena.

Testiryhmässä oli alustavasti päätetty Robot Frameworkista testityökaluna, sillä sen käytöstä oli hieman jo kokemusta. Tarkemman tutustumisen johdosta selvisi, että Robot Frameworkissa on erittäin kattavat ja selkeästi dokumentoidut testikirjastot, jotka tukevat yleisempiä selaimia. Robot Frameworkissa on avoin lähdekoodi ja sen käyttöönottoon oli selkeät aloitusohjeet sekä saatavilla laajasti opetusmateriaalia. Robot Framework on myös saanut laajasti kannatusta Suomessa, sillä useat yritykset ovat valinneet sen testausautomaatiotyökaluksi. (Robot Framework / Users 2018)

Edellä mainituista syistä opinnäytetyössä rajattiin testiautomaatiotyökalujen tutkiminen Robot Frameworkiin.

Lisäksi testauskohde rajattiin web-käyttöliittymään, koska sille oli olemassa valmis kirjasto. Muita mahdollisia testauskohteita olisivat olleet mm. sulautetut laitteet, mutta ne jätettiin pois aikarajoitteen takia. Ne vaatisivat todennäköisesti omien kirjastojen kirjoittamista.

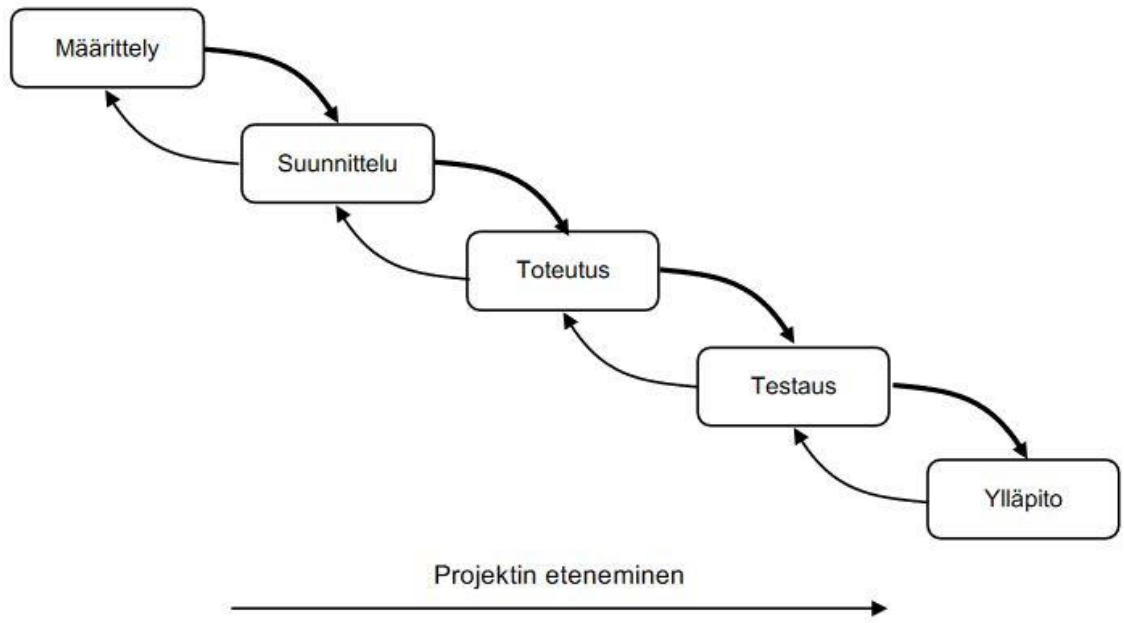
Opinnäytetyön ensimmäisenä vaiheena oli tutustua Robot Frameworkiin ja sen opetusmateriaaleihin. Tämän jälkeen suoritettiin vaadittavat lataukset ja asennukset. Seuraavaksi perehdyttiin sen käyttöönottoon ja toimintoihin sekä kirjoitettiin muutama testitapaus.

Lisäksi opinnäytetyössä perehdytään ohjelmistotestaukseen yleisesti, sen hyötyihin yrityksen näkökulmasta sekä ohjelmistotestauksen eri tasoihin. Työssä myös analysoidaan automaatiotestauksen hyötyjä sekä sen mahdollisia haasteita.

2 OHJELMISTOTESTAUS

Ohjelmistotestaus on aktiviteetti, jossa varmistetaan, että tuotettava ohjelmistotuote on toivotun kaltainen ja siihen valmiiksi saadut ominaisuudet toimivat oikealla tavalla. Ohjelmistotestaus on vain yksi osa ohjelmistotuotannon kokonaisuudesta ja siihen liittyvät työvaiheet ovat testauksen suunnittelu, testiympäristön luonti, testin suorittaminen ja tulosten tarkastelu. (Haikala & Mikkonen 2011, 205)

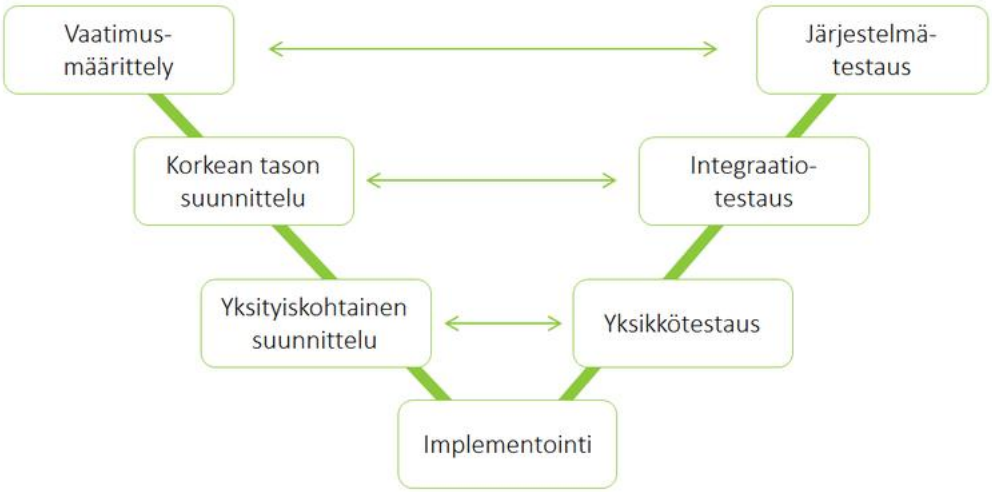
Perinteisesti ajatellaan, että testaus on vain yksi työvaihe ohjelmistotuotannon vesiputousmallissa. Vesiputousmallin ajatuksena on siirtyä aina askel eteenpäin seuraavaan vaiheeseen, kun edellinen päättyy (Kuva 1.) ja tavoitteena on, ettei palata enää taaksepäin. Tämä malli on epäkäytännöllinen, sillä testausta suoritetaan vain yhdessä vaiheessa ja silloin on jo pääosin kaikki suunnittelu- ja kehitystyö tehty. (Kasurinen 2013, 12-13)



Kuva 1. Perinteinen vesiputousmalli. (Kasurinen 2013)

V-mallissa testaus alkaa jo toteutusvaiheen aikana ja kestää projektin loppuun asti. Siinä testaus ei ole erillinen työvaihe, vaan jokaiselle rakennusvaiheelle on merkitty oma testaustaso, jonka Kuva 2 hyvin havainnollistaa. Testaustasot ovat yksikkötestaus,

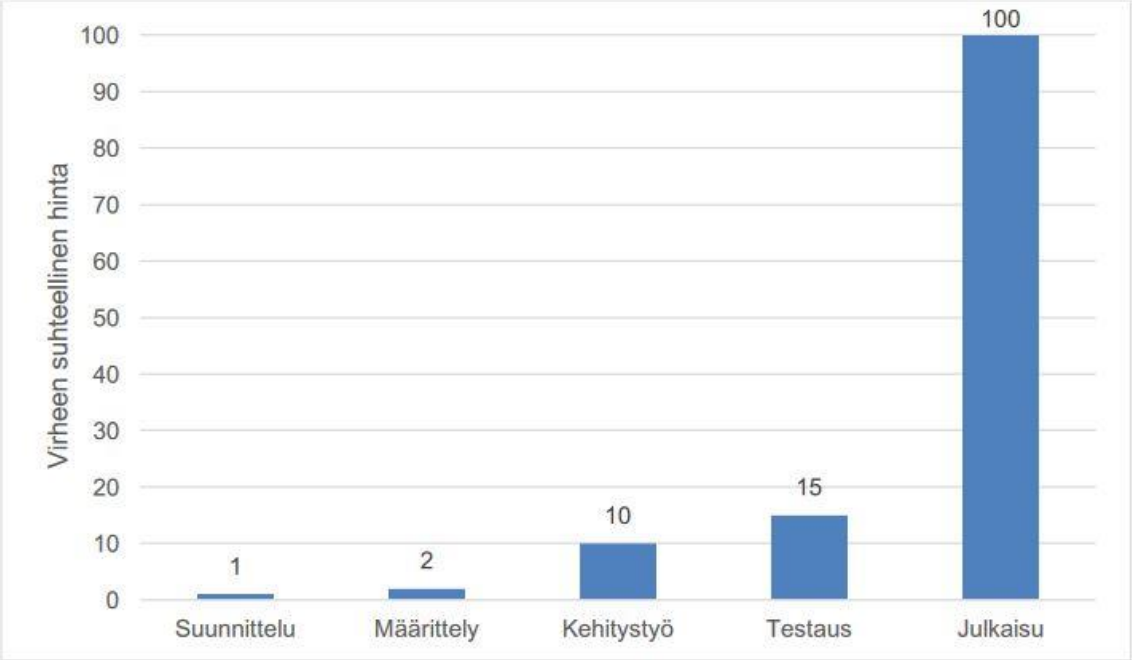
integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. Kun ohjelmisto on läpäissyt kaikki testaustasot, se on valmis käyttöönottoon. (Kasurinen 2013, 13-14)



Kuva 2. Testauksen V-malli. (Jyväskylän yliopisto 2017)

2.1 Testauksen hyödyt

Ohjelmistotestaus on tarpeen, jotta voidaan ilmaista kehitysvaiheiden aikana esiintyvät virheet. Testaus varmistaa, että tuotteen suorituskyky on riittävä ja asiakkaiden tarpeiden mukainen. Kun tuote on laadukas ja toimiva, se auttaa asiakkaiden luottamuksen saamista. Koska ohjelmistotestaus auttaa tunnistamaan ja korjaamaan vikoja ennen ohjelmiston käyttöönottoa, epäonnistumisen riskiä voidaan huomattavasti pienentää. Ohjelmiston käyttöönoton jälkeen virheiden korjaus voi vahingoittaa koko ohjelmistoa ja korjauskustannukset voivat nousta korkealle. (Kuvio 1.) Testaus myös paljastaa ohjelmiston mahdolliset kehittämisalueet. Kun testaus on täsmällistä ja tarkkaa, takaa se yritykselle alhaisemmat ylläpitokustannukset. (Advanto 2016)



Kuva 3. Löydetyn virheen suhteellinen hinta eri kehitysvaiheissa. (Kasurinen 2013)

2.2 Ohjelmistotestauksen tasot

Ohjelmistotestauksella on V-mallin mukaan yleisesti neljä eri testaustasoa. Ja mitä korkeammalle V-mallissa mennään, sitä suuremmiksi testattavat kokonaisuudet muuttuvat. Nämä neljä testaustasoa ovat yksikkötestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. (Kasurinen 2013, 78)

2.2.1 Yksikkötestaus

Yksikkötestaus on yleisin testausmenetelmä ja se tarkoittaa testaustyötä, jossa testataan yksittäisiä moduuleita, komponentteja tai olioiden toimintoja. Sen ideana on tarkistaa, että jokainen muutos tai toiminto ohjelmistossa toimii suunnitellulla tavalla. Testauksen toteuttaa yleensä ohjelmoija itse, jolloin hän voi korjata ongelman heti. Usein ongelmana on, ettei yksittäinen elementti voi suorittaa itsenäisesti mitään. Näihin tapauksiin pitää rakentaa testikomponentteja, joiden tarkoitus on mallintaa varsinaisen järjestelmän toimivuutta. (Kasurinen 2013, 79-84)

2.2.2 Integraatiotestaus

Integraatiotestaus tulee yksikkötestauksen jälkeen ja siinä yksittäiset elementit yhdistetään ja testataan ryhmänä. Tämän testaustason tarkoituksena on paljastaa vikoja integroitujen osien vuorovaikutuksessa sekä testata, että osat toimivat yhdessä. Integraatiotestauksessa toimivaan kokonaisuuteen liitetään joka kerta yksi elementti lisää ja testataan lopputuloksen toimivuus. Kun jokainen elementti on integroitu järjestelmään, mennään eteenpäin järjestelmätestaukseen. (Kasurinen 2013, 84-87)

2.2.3 Järjestelmätestaus

Järjestelmätestaus on ohjelmistotestauksen taso, johon siirrytään, kun elementit on ensin yksikkötestattu ja sen jälkeen rakennettu toimivaksi ohjelmistoksi integraatiotestauksessa. Järjestelmätestaus tehdään yhtenäiselle järjestelmälle. Testauksen tarkoituksena on verifioida, että järjestelmä toimii niin kuin oli suunniteltu ja toteuttaa kaikki sille asetetut vaatimukset. Järjestelmää ei kuitenkaan vielä testata lopullisessa käyttöympäristössä vaan testaus tapahtuu sille tarkoitetussa testiympäristössä. Järjestelmätestauksessa etsitään vielä vikoja myös yksittäisistä elementeistä ja muutokset järjestelmään ovat tavallisia. (Kasurinen 2013, 87-90)

2.2.4 Hyväksymistestaus

Hyväksymistestaus on viimeinen ohjelmistotestauksentaso ja sen päätarkoitus on todistaa ohjelmiston olevan riittävän korkealaatuinen ja toteuttaa vaatimusmäärittelyn. Hyväksymistestauksella luonnehditaan järjestelmän virallista katselmusta ja onnistuneeksi havaittu ohjelmisto siirtyy pois kehitysvaiheesta. Hyväksymistestauksessa on tavanomaista, että testaus tapahtuu järjestelmän lopullisessa käyttöympäristössä. Hyväksymistestauksessa, toisin kuin järjestelmätestauksessa, järjestelmälle ei enää tehdä merkittäviä muutoksia. (Kasurinen 2013, 91-92)

3 AUTOMAATIOTESTAUS

Automaatiotestauksella tarkoitetaan testaustoimintaa, jossa rakennetaan automaatiotyövälineitä ohjelman testaamista varten. Tavoite on testata toistuvia testitapauksia helposti ja tehdä niiden tarkistamiseen nopea ohjelmisto tai laite. Näin vältetään ihmisten tekemät virheet ja pystytään toistamaan testit useampaan kertaan samanlaisina. Päätavoitteena on testaajien työn nopeuttaminen ja heidän vapauttamisensa muihin tehtäviin.

Automaatiotestauksen käyttö ei silti poista testaajien tarvetta. Sen tarkoituksena on ainoastaan vähentää käsiin tehtävää testauksen määrää. Automaation tarkoituksena ei ole hakea ohjelmistosta uusia ongelmia ja vikoja vaan varmistaa, että aiemmin toimineet osat toimivat edelleen eivätkä rikkoutuneet kehitysprosessin aikana. (Kasurinen 2013, 76-78)

Automaatiotestaus kehittyy nopeasti ja sitä käytetään koko ajan enemmän, sillä automaatio-ohjelmistoa voidaan päivittää ja käyttää uudelleen helposti. Hyvin toteutetulla automaatiotestauksella voidaan vähentää epäonnistumisen riskiä. (Advanto 2016)

3.1 Automaatiotestauksen edut

Automaatiotestauksen etuna on testien suorittamisen nopeus verrattuna manuaaliseen testaukseen. Suorittaminen kestää lyhyemmän ajan, jolloin testejä voidaan suorittaa useammin. Testejä on mahdollista suorittaa myös rinnakkain, mikä mahdollistaa useamman testin samanaikaisen suorittamisen. (Laapas 2014) Testien toistettavuus täysin samanlaisina sekä testien uudelleenkäyttö ovat automaatiotestauksen etuja. Huolella toteutetut automaatiotestit lyhentävät tuotteen testausaikaa, jolloin tuote saadaan aikaisemmin markkinoille. (Murtiosalo 2015)

3.2 Automaatiotestauksen haasteet

Testiautomaatio edellyttää paljon etukäteen suunnittelua, ennakointia sekä osaamista testitiimiltä. Automaatio tukeutuu aikaisempiin testituloksiin, joten testitiimiltä vaaditaan yhteistyötä ja kommunikointia. (Yang 2018, Osuch 2018)

Myös oikean testausmenetelmän valinta on haastavaa. Pitää tietää, miten suunnitellaan ja toteutetaan automaatiotestit, jotta ne on helppo pitää ajan tasalla kohtuullisella ylläpidolla. (Vo 2017)

Lähes kaikissa tapauksissa testiautomaation alkuvaiheessa on joitain kustannuksia, johtuen sen edellyttämästä analyysista, suunnittelusta ja toteutuksien hankinnoista. Vaikka automaatiotyökalu olisi avoimen lähdekoodin ansiosta ilmainen tai edullinen, tulee kustannuksia automaation ylläpidosta sekä koulutuksista. Kun testiautomaatio on tehty kunnolla ja oikein, voi se tuottaa merkittäviä parannuksia laatuun, tarkkuuteen sekä tuottavuuteen. (Yang 2018)

Vaikka testiautomaatio on erinomainen järjestelmän virheiden tarkistamiseen, tarvitaan silti ihmisiä täydentämään ei-automatisoidut testausosat. Sillä ei ole mahdollista automatisoida jokaista testitapausta, esimerkiksi käytettävyyttä tai ulkoasua. Ja nämä testit tuovat tärkeää tietoa ohjelmiston parantamiseksi – ei kehittäjien vaan loppukäyttäjien puolesta. (Osuch 2018)

4 ROBOT FRAMEWORKIN KÄYTTÖÖNOTTO

Yrityksessä on käytössä Windows käyttöjärjestelmä, joten Robot Framework asennettaan Windows 10:lle. Robot Frameworkin lisäksi asennetaan Python, Pip, Selenium2-kirjasto sekä wxpython.

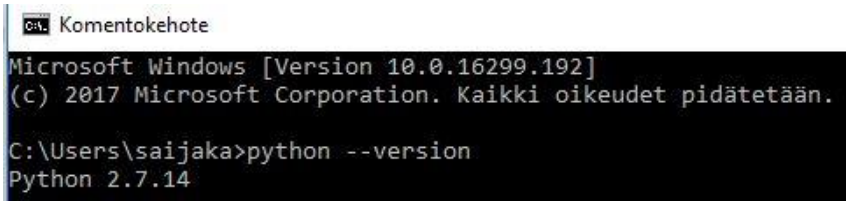
Python asennus

Robot Framework vaatii toimiakseen joko Pythonin, Jythonin tai IronPythonin. Mutta koska testien kirjoittamiseen ja suorittamiseen suunniteltiin käytettäväksi RIDE-editoria, joka tukee ainoastaan Pythonia, päädyttiin tässä työssä Pythoniin. (Robot Framework User Guide 2017) Pythonia ei ole Windowsissa valmiina asennettuna, joten asennetaan ensimmäiseksi.

RIDE-editori tukee ainoastaan Pythonin versiota 2.6 ja sitä uudempia muttei kuitenkaan Python 3:a, joten Pythonista ladattiin versio 2.7.14. (GitHub 2016) Pythonia asentaessa kustomointi-ikkunassa suositeltiin tarkastamaan, että Pip valitaan asennettavaksi. Pip on Python-pakettien hallintaohjelma, jolla pystytään asentamaan paketteja suoraan Python Package Indexistä. Tuoreimmat Python-asennusohjelmat mahdollistavat Pythonin lisäämisen suoraan ympäristömuuttujiin osana asennusta. Oletusarvoisesti tämä on poissa käytöstä, mutta valitsemalla kustomointi-ikkunasta ”Add python.exe to Path”, asennusohjelma lisää sen automaattisesti (Kuva 4.). (Robot Framework User Guide)

Kuva 4. Pip:n asentaminen ympäristömuuttujiin.

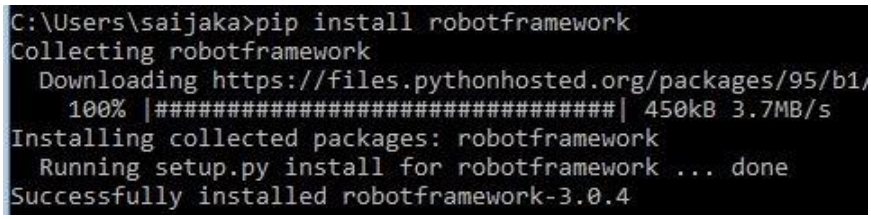
Pythonin asentamisen jälkeen tarkistetaan asennus kirjoittamalla komentoriville *Python --version* ja asennuksen onnistuessa, komentokehotteeseen tulostuu Pythonin versio. (Kuva 5.)



Kuva 5. Tieto Pythonin versiosta komentokehotteessa.

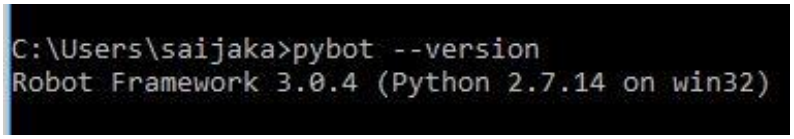
Robot Frameworkin asennus

Onnistuneen Pythonin asennuksen jälkeen voidaan asentaa Robot Framework. Asennus suoritetaan käyttämällä Pip: ä (Kuva 6.). Komentokehotteessa näkyi viesti asennuksesta sekä tämän jälkeen tieto sen onnistumisesta.



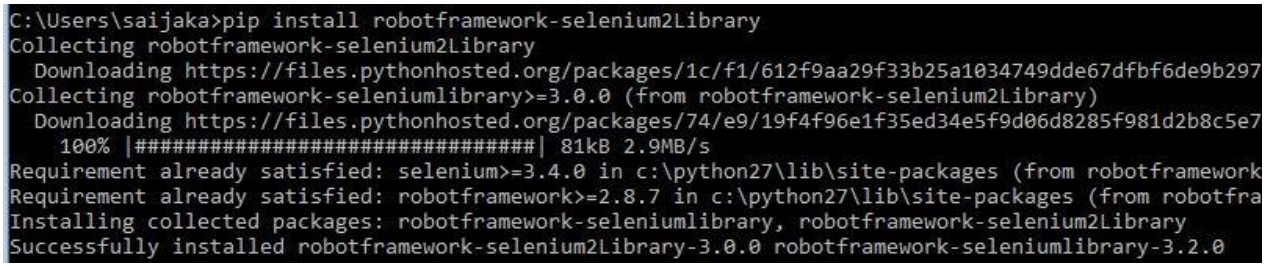
Kuva 6. Robot Frameworkin asennus.

Asennuksen onnistumisen voi vielä tarkistaa kirjoittamalla komentoriville *pybot --version*, tämä komento tulostaa tiedon sekä Pythonin, että Robot Frameworkin versionumerosta (Kuva 7.).



Kuva 7. Tieto Pythonin ja Robot Frameworkin versiosta.

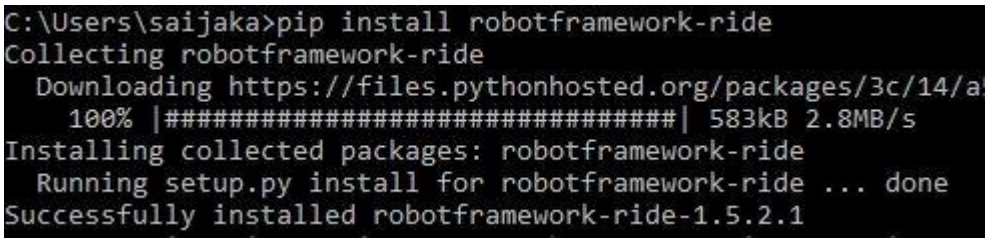
Opinnäytetyössä käytettiin Selenium2-kirjastoa ja sen asentaminen suoritettiin myös Pipiä käyttämällä (Kuva 8.). (Robot Framework User Guide 2017)



Kuva 8. Selenium2-kirjaston asennus.

RIDE asennus

RIDE-käyttöliittymä käyttää wxPython pakettia toimiakseen, joten asennusta varten ladataan ensin wxPython ja siitä ainoa yhteensopiva versio on 2.8.12.1. Kuten kirjastot, RIDE lataus onnistuu Pipin kanssa (Kuva 9.). RIDE saa käyntiin komennolla *ride.py*. (GitHub 2016)



Kuva 9. RIDE asennus.

5 TESTIEN SUORITTAMINEN

Toimeksiantajayrityksellä on tuotekehitysvaiheessa oleva uusi hoitajakutsujärjestelmän käyttöliittymä, jonka toiminnallisuuksia testataan. 3 kappaletta automaatiotestiä toteutetaan uudelle käyttöliittymälle. Testejä varten luodaan käyttäjätunnukset hoitajakutsujärjestelmän testitilille.

Kaikki avainsanat Selenium2-kirjastossa, jotka kohdistuvat tiettyyn elementtiin, tarvitsevat argumenttina paikantimen. Paikannin on merkkijono, joka kertoo, miten elementti paikannetaan käyttämällä määriteltyä syntaksia. (Robot Framework User Guide 2017) Näissä testeissä käytetään paikantimena id-attribuuttia, Xpath-polkua ja CSS-selektoria.

5.1 Onnistunut kirjautuminen

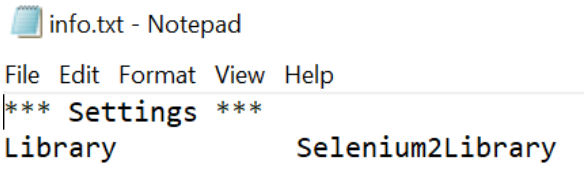
Ensimmäisessä testissä avataan määritelty selain ja siirrytään käyttöliittymän kirjautumissivulle. Syötetään kenttiin käyttäjätunnus ja salasana, jonka jälkeen painetaan kirjaudu nappia. Testi tarkistaa vielä, että kirjautuminen onnistui ja sulkee selaimen.

Testi kirjoitetaan Notepad-tekstieditorilla, jossa avainsanat ja argumentit erotetaan neljällä välilyönnillä.

5.1.1 Testin kirjoitus

Testin kirjoitus aloitetaan luomalla erillinen resurssitiedosto, johon pystytään määrittelemään avainsanoja ja muuttujia. Näin itse testitiedosto pysyy siistinä sekä helppolukuisena. Resurssitiedostoa voidaan käyttää myös muissa testitapauksissa ja näin uusien testien kirjoittaminen on nopeampaa.

Resurssitiedostolle annettiin nimi info.txt. Testissä käytetään Selenium2-kirjaston avainsanoja, joten kirjasto pitää lisätä info.txt-tiedoston alkuun Settings-osioon (Kuva 10.).



Kuva 10. Kirjaston käyttöönotto.

Seuraavaksi määritellään muuttujat resurssitiedostoon kirjaston alapuolelle. Tämän testin muuttujissa määritellään käytettävä selain, oikeat käyttäjätunnukset sekä kirjautumissivu ja onnistuneen kirjautumisen jälkeinen verkkosivu (Kuva 11.).

```
***Variables ***
${BROWSER}          Firefox
${VALID USER}       test
${VALID PASSWORD}     Robot
${LOGIN URL}        http://172.29.1.49:4200/login
${WELCOME URL}      http://172.29.1.49:4200/dashboard
```

Kuva 11. Muuttujat infotiedostossa.

Tämän jälkeen lisätään avainsanat ja niiden argumentit. Resurssitiedostoon on kirjoitettu avainsanoja, jotka on koottu yhdistämällä useita avainsanoja (Kuva 12.).

```
*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Login Page Should Be Open

Login Page Should Be Open
    Title Should Be    Company

Go To Login Page
    Go To    ${LOGIN URL}
    Login Page Should Be Open

Input Username
    [Arguments]    ${username}
    Input Text    username    ${username}

Input Password
    [Arguments]    ${password}
    Input Text    loginPassword    ${password}

Submit Credentials
    Click Button    loginSubmit

Welcome Page Should Be Open
    Location Should Be    ${WELCOME URL}
    Title Should Be    Company
```

Kuva 12. Avainsanat resurssitiedostossa.

Seuraavaksi kirjoitetaan itse testitiedosto. Testitiedosto nimettään login.txt ja sen asetuksiin määritetään luotu resurssitiedosto info.txt. Tämän jälkeen lisätään ensimmäisen testitapauksen otsikko, tässä testissä se on `Valid Login`. Otsikot suositellaan nimeämään yksiselitteisesti, jotta jo nimen perusteella tiedetään, mitä testitapauksessa on tarkoitus tapahtua. (Robot Framework User Guide 2017) Tämän alle kirjoitetaan kutsuttavan avainsanan nimi ja sen perään seuraaviin sarakkeisiin avainsanalle syötettävät parametrit. Lopussa on määritelty `Teardown`-avainsana. Se ajetaan testitapauksen jälkeen. Siinä on määritelty parametriksi `Close Browser`, mikä sulkee selaimen testitapauksen jälkeen. (Kuva 13.).

```
*** Settings ***
Resource          info.txt

*** Test Cases ***
Valid Login
    Open Browser To Login Page
    Input Username    test
    Input Password    Robot
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]    Close Browser
```

Kuva 13. Kirjautumistesti.

5.1.2 Testin tulokset

Testi käynnistetään komentorivillä komennolla *robot login.txt* ja komentorivin tulosteesta nähdään testiajon tulokset. Testiraportit ovat luettavissa samassa kansiossa, jossa testitiedostot ovat (Kuva 14.).

```
C:\Robot\Project2>robot login.txt
==============================================================================
Login
==============================================================================
Valid Login                                                           | PASS |
------------------------------------------------------------------------------
Login                                                                 | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Robot\Project2\output.xml
Log:     C:\Robot\Project2\log.html
Report:  C:\Robot\Project2\report.html

C:\Robot\Project2>
```

Kuva 14. Onnistunut testiajo.

5.2 Epäonnistunut kirjautuminen

Toisessa testissä tarkoituksena on yrittää kirjautua kuudella erilaisella virheellisellä käyttäjätunnus-salasana yhdistelmällä sisälle käyttöliittymään ja tarkistaa, että sivulle tulee virheilmoitus sekä verkko-osoite pysyy kirjautumissivulla.

5.2.1 Testin kirjoitus

Kirjoitetaan uusi testitiedosto (Kuva 15.) ja nimetään se InvalidLogin.txt. Määritellään sen asetuksiin aikaisemmin luotu resurssitiedosto info.txt, `Setup` -ja `Teardown`-komennot, jotka jokaisen testitapauksen alussa avaavat selaimen, siirtyvät kirjautumissivulle ja lopuksi sulkevat selaimen. Lisätään testitapaukset ja tarvittavat avainsanat sekä parametrit.

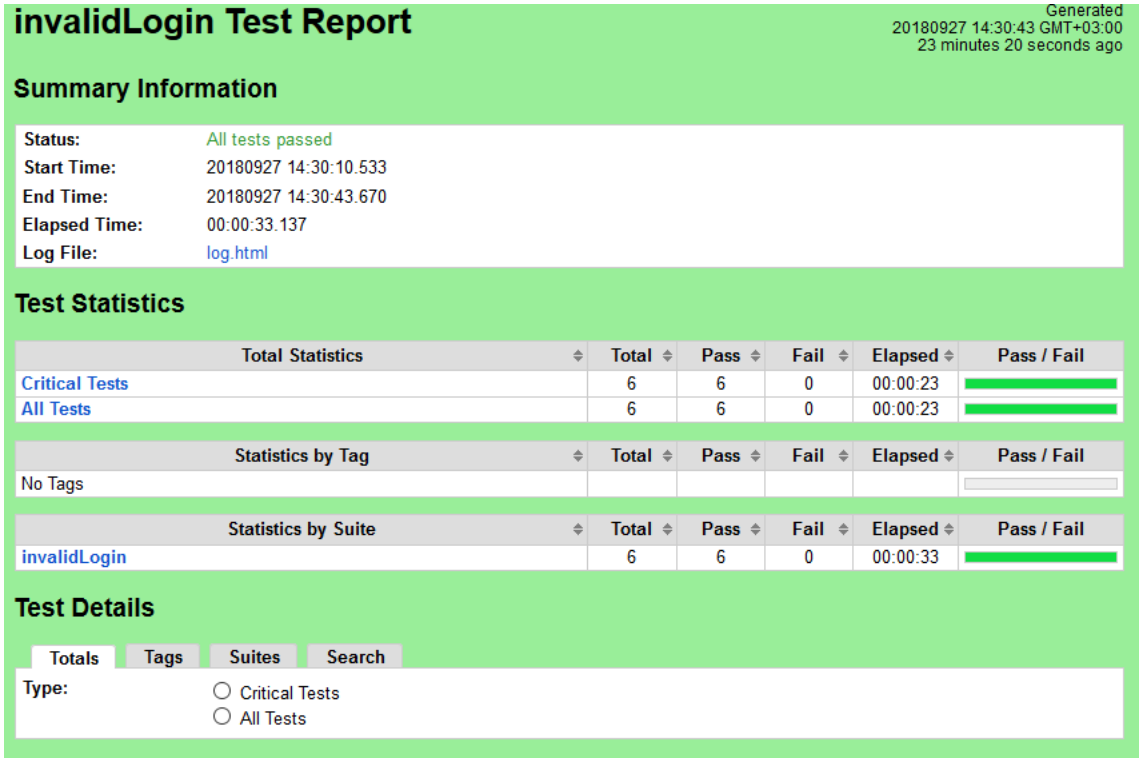
```
*** Settings ***
Suite Setup       Open Browser To Login Page
Suite Teardown    Close Browser
Test Setup        Go To Login Page
Test Template     Login With Invalid Credentials Should Fail
Resource          info.txt

*** Test Cases ***               USER NAME        PASSWORD
Invalid Username                 invalid          ${VALID PASSWORD}
Invalid Password                 ${VALID USER}    invalid
Invalid Username And Password    invalid          whatever
Empty Username                   ${EMPTY}         ${VALID PASSWORD}
Empty Password                   ${VALID USER}    ${EMPTY}
Empty Username And Password      ${EMPTY}         ${EMPTY}

*** Keywords ***
Login With Invalid Credentials Should Fail
    [Arguments]    ${username}    ${password}
    Input Username    ${username}
    Input Password    ${password}
    Submit Credentials
    Wait Until Keyword Succeeds    5s    1s    Login Should Have Failed

Login Should Have Failed
    Location Should Be    ${ERROR URL}
    Element Should Be Visible    ${ALERT}
    Title Should Be    Company
```

Kuva 15. InvalidLogin-testitiedosto.

5.2.2 Testin tulokset

Kun testi on ajettu onnistuneesti, voidaan tuloksia käydä katsomassa Robot Frameworkin testiraportissa (Kuva 16.) tai testilokissa. Testiraportissa näkyy nopeasti ja selkeästi testitulokset, sillä sen tausta on vihreä, jos testit ovat menneet onnistuneesti läpi. Jos

yksikin testi on epäonnistunut, on tausta punainen. Raportissa on myös helposti luettavissa kaikkien testitapausten ajoaika sekä yksittäisen testitapauksen ajoaika.



invalidLogin Test Report

Generated
20180927 14:30:43 GMT+03:00
23 minutes 20 seconds ago

Summary Information

Status:	All tests passed
Start Time:	20180927 14:30:10.533
End Time:	20180927 14:30:43.670
Elapsed Time:	00:00:33.137
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	6	6	0	00:00:23	
All Tests	6	6	0	00:00:23	

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

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
invalidLogin	6	6	0	00:00:33	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests

Kuva 16. Robot Frameworkin testiraportti.

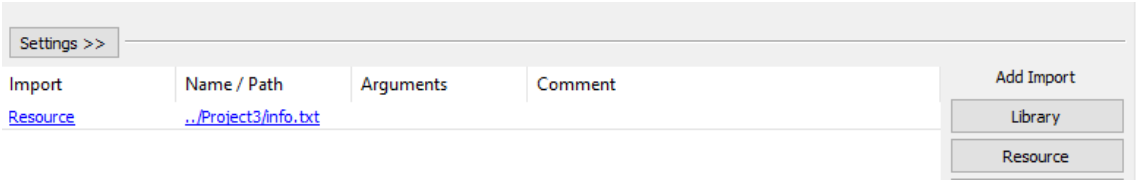
5.3 Laitteen lisäys

Kolmantena ja viimeisenä testinä tehtiin uuden laitteen lisäys testiasiakkuuteen käyttöliittymästä. Kirjaudutaan oikeilla tunnuksilla testitilille, painetaan Laitehallintalinkkiä ja sen jälkeen sivun yläreunasta ”+ lisää uusi laite”-nappia. Oikeaan reunaan ilmestyy ikkuna johon pitää kirjoittaa uuden laitteen id ja painaa tämän jälkeen tarkista-nappia. Jos tarkistus menee läpi ja laitetta ei vielä ole käyttöliittymässä alapuolelle ilmestyy kentät ”Nimi” ja ”Sijainti”. Täytetään niihin oikeat tiedot ja painetaan tallenna-nappia. Laite ilmestyy näkyviin testitilin laitelistaan.

5.3.1 Testin kirjoitus

Testi kirjoitettiin käyttämällä RIDE-editoria. RIDE:n saa käyntiin kirjoittamalla komentoriville *ride.py*. RIDE käynnistyttyä luodaan uusi projekti, jonka alapuolelle lisätään uusi

testisarja. RIDE:ssa voidaan myös käyttää resurssitiedostoa, joten lisätään jo aiemmin luotu resurssitiedosto testisarjan asetuksiin (Kuva 17.).



Kuva 17. Resurssitiedoston lisäys asetuksiin.

Tämän jälkeen lisätään testisarjaan uusi testitapaus ja nimetään se AddDevice. Koska resurssitiedosto lisättiin jo asetuksiin löytyvät sen avainsanat valmiina listasta. Avataan resurssitiedosto text edit-välilehdellä ja lisätään loppuun tarvittavat avainsanat (Kuva 18.).

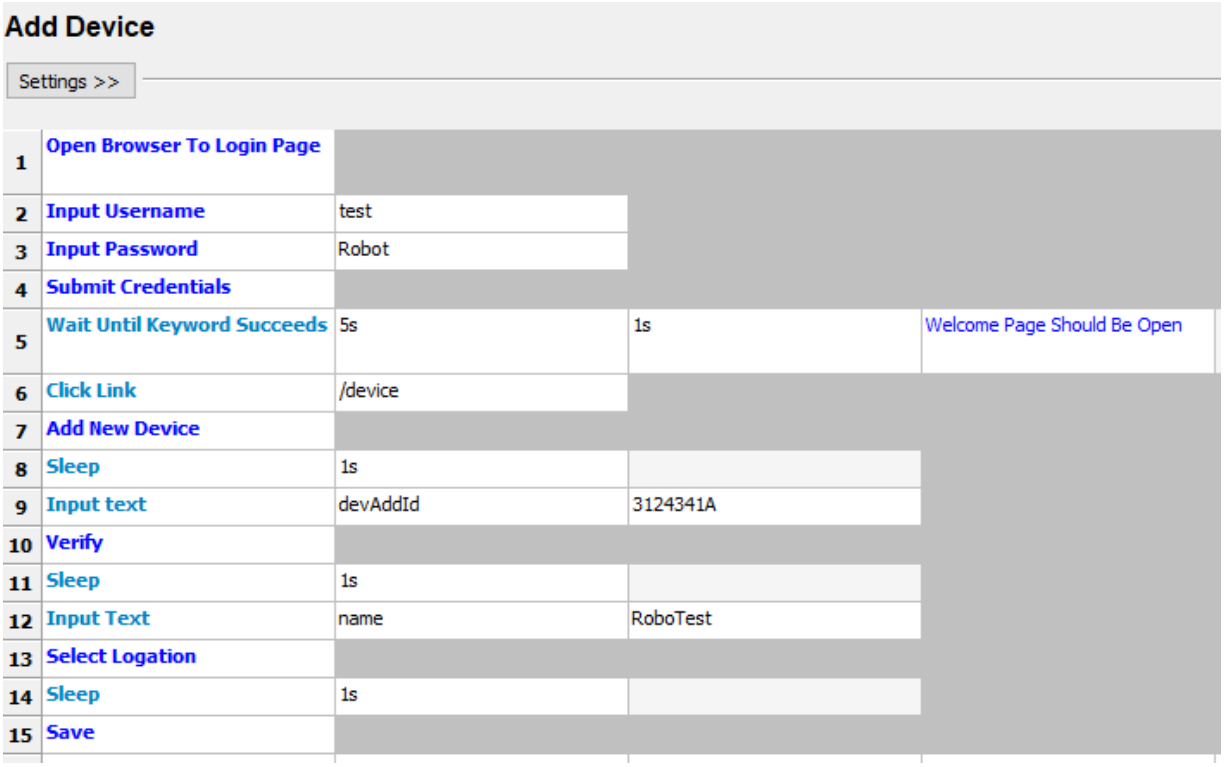
```
39
40  Add New Device
41     Click Button    css=button.btn.btn-outline-primary[_ngcontent-c6=""]
42
43  Verify
44     Click Button    css=button[_ngcontent-c11=""]
45
46  Select Logation
47     Click Element    css=select
48     Click Element    css=option.ng-star-inserted:nth-child(1)
49
50  Save
51     click Button    css=button.btn.btn-outline-primary.mr-1
52
```

Kuva 18. Tarvittavat avainsanat resurssitiedostoon.

Tämän jälkeen ruvetaan kirjoittamaan testin avainsanoja ja tarvittavia parametreja RIDE:n taulukkoon. Ensimmäiseen sarakkeeseen kirjoitetaan avainsana ja seuraaviin avainsanan parametrit. RIDE näyttää avainsanalle täytettävät pakolliset parametrisarakkeet punaisena, vapaavalintaiset vaaleanharmaana ja loput tummana. Painamalla ctrl + shift yhdistelmää saa auki avainsanalistan josta löytyy selitys ja esimerkki avainsanoille. Listasta löytyvät myös resurssitiedoston avainsanat (Kuva 19.).

Kuva 19. Avainsanalista RIDE:ssa.

Kirjoitetaan testin vaatimat avainsanat ja parametrit sarakkeisiin (Kuva 20.) ja ajetaan testi RIDE:n Run-välilehdeltä painamalla start-nappulaa.

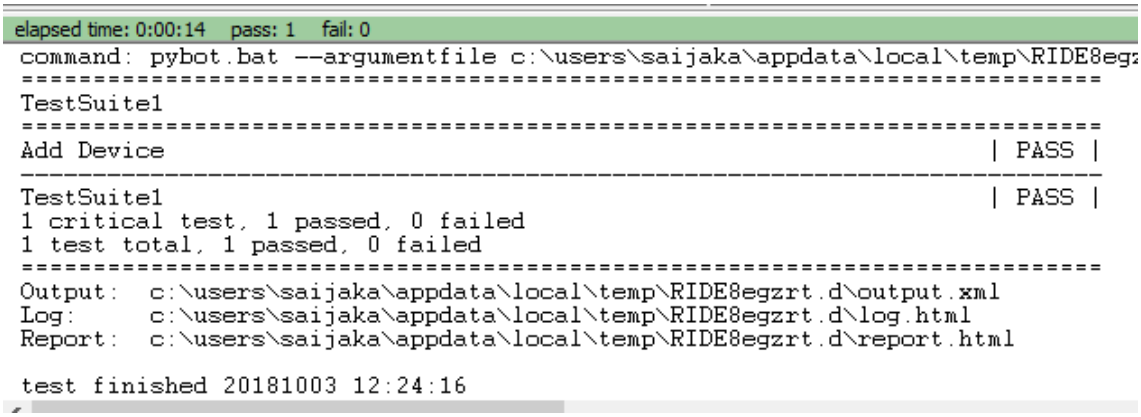


Kuva 20. AddDevice- testi

5.3.2 Testin tulokset

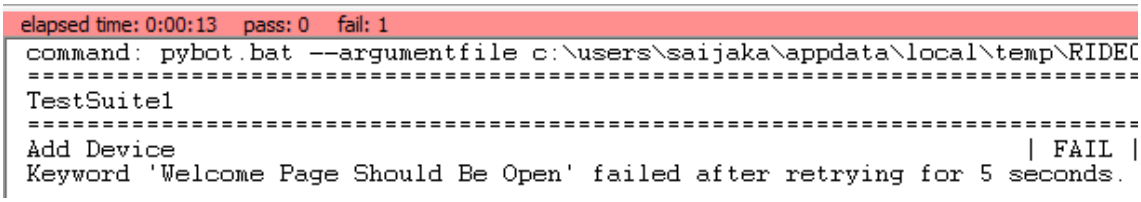
Kun testi on ajettu Robot Frameworkilla, jää verkkosivu auki ja voidaan itse nähdä uusi laite laitelistassa. Samalta run-välilehdeltä nähdään myös testitulos. Jos testi on

onnistunut, tulee yläosaan vihreä palkki, jossa on kerrottu testin suoritusaika ja kuinka monta testitapausta onnistui ja kuinka monta epäonnistui (Kuva 21.).



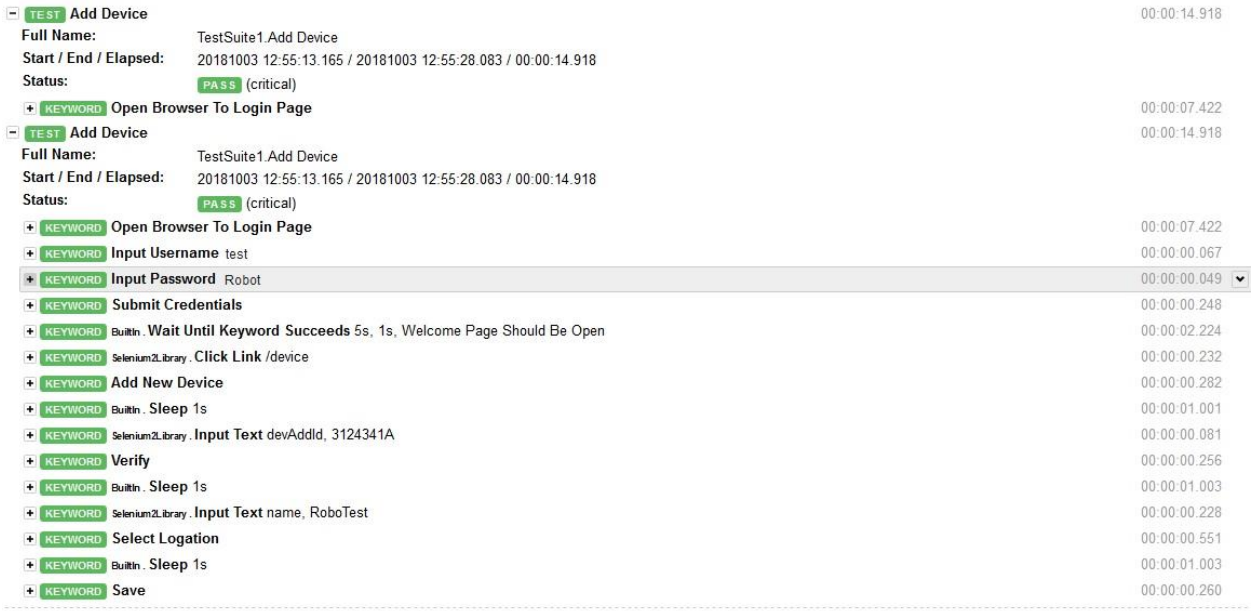
Kuva 21. Onnistunut testiajo RIDE:lla.

Jos testi epäonnistuu, on yläosan palkki punainen. Myös testitapauksen perään tulee teksti FAIL ja alapuolelle selitys miksi testi epäonnistui (Kuva 22).



Kuva 22. Epäonnistunut testiajo RIDE:lla.

Run-välilehdellä on Report- ja Log-nappulat josta päästään lukemaan testin tuloksia tarkemmin. Report-sivulla nähdään testin onnistunut ajo vihreästä taustaväristä sekä testin aloitus- ja lopetusaika. Log-sivulla nähdään testintulos, aloitus- ja lopetusaika sekä pystytään katsomaan erikseen jokaisen avainsanan suorittamiseen kulunut aika (Kuva 23.) Jos testin ajo kestää kauemmin kuin on toivottua, niin tästä listasta voidaan katsoa minkä avainsanan ajo aiheuttaa viiveen.



Kuva 23. Robot Frameworkin Log-sivu.

6 TULOKSET

Työn tavoitteena oli saada automaatiotestaus alkuun toimeksiantajayrityksessä. Automaatiotyökaluksi valikoitui Robot Framework, sillä siinä oli avoin lähdekoodi, monipuoliset kirjastot sekä yrityksen testitiimillä oli jo hieman kokemusta kyseisestä työkalusta. Testien kirjoittamiseen käytettiin osaksi RIDE-editoria, jonka käyttö oli uutta kaikille.

Robot Frameworkin asennus oli selväpiirteistä mutta jos kokemus ohjelmoinnista on vähäistä, vaati käyttöönotto hieman perehtymistä. Suurimpina ongelmina testejä kirjoittaessa vastaan tuli joidenkin elementtien paikantaminen. Robot Frameworkin esimerkeissä paikantamiseen käytettiin aina id-attribuuttia mutta yrityksen käyttöliittymän verkkosivuilla tämä ei ollut aina mahdollista. Tarkemman perehtymisen ja useiden kokeilujen jälkeen ratkaisuksi löytyivät CSS-selektori ja Xpath-polku, joilla pystyi paikantamaan elementin ilman nimeä. Toisena ongelmana oli, ettei verkkosivu ehtinyt latautua aina riittävän nopeasti ennen seuraavan avainsanan suorittamista. Tällöin testistä tuli virheilmoitus ja testiajon suorittaminen pysähtyi siihen. Ratkaisuksi tähän käytettiin `sleep`-avainsanaa, jolla pystytään asettamaan viivettä suoritettavien avainsanojen väleihin.

RIDE-editorin käyttö testien kirjoittamisessa osoittautui mainioksi työkaluksi sen sisältämän avainsanalistan ansiosta. RIDE-editoriin saa kätevästi asennettua asetuksiin kirjastoja sekä tallennettua valmiita resurssitiedostoja. Testit saa ajettua joko komentokehotteesta tai RIDE-editorista ja tulokset tulevat heti selkeästi näkyviin. Tarkempia testituloksia pääsee tarkastelemaan Robot Frameworkin luomasta Report - ja Log-tiedostoista.

Työssä olisi ollut tarpeen perehtyä tarkemmin muihinkin automaatiotyökaluihin, jotta olisi pystytty varmistumaan parhaasta mahdollisesta työkalusta yritykselle. Mutta rajallisen ajan sekä testiryhmän alustavan päätöksen vuoksi päätettiin rajata työ vain Robot Frameworkin tutkimiseen. Kiinnostavaa myös olisi ollut selvittää miten testien ajo voidaan käynnistää automaattisesti, esimerkiksi tiettyyn kellon aikaan mutta tämän työn puitteissa sitä ei lähdetty tekemään.

Työstä saadun kokemuksen perusteella on hyvä lähteä kehittämään uusia automaatiotestejä käyttöliittymälle sekä myöhemmin laajentamaan testejä yrityksen muihin ympäristöihin muun muassa sulautetuille järjestelmille. Työ osoitti automaatiotestauksen tarpeellisuuden yritykselle, sillä testattavaa on paljon. Automaatiotestaus varmistaisi

tuotteen laadukkuuden sekä takaisi asiakkaille nopeammat tuotepäivitykset. Se antaisi myös nopeampaa palautetta tuotekehitykselle, miten muutokset vaikuttivat tuotteen laatuun.

LÄHTEET

Advanto software 2016. Benefits of software testing. Viitattu 23.2.2018 https://www.advantosoftware.com/blogs/benefits-of-software-testing/.

GitHub 2016. robot framework/RIDE. Viitattu 12.03.2018 https://github.com/robotframework/RIDE/wiki/Installation-Instructions.

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. 12., uudistettu painos. Helsinki: Talentum.

Jyväskylän yliopisto 2017. Taloudellinen näkökulma. Viitattu 23.2.2018 http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/taloudellinen-nakokulma.

Jyväskylän yliopisto 2017. Testaus lyhyesti. Viitattu 5.3.2018 http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testaus-lyhyesti.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo.

Laapas,A. 2014. Cost-benefit analysis if using test automation in the development of embedded software. Diplomityö. Lappeenrannan teknillinen yliopisto, tuotantotalous, School of Industrial Engineering and Management. Viitattu 8.9.2018 http://www.doria.fi/bitstream/handle/10024/97101/masters_thesis_cost_benefit_analysis_of_test_automation.pdf?sequence=2&isAllowed=y.

Murtiosalo, J. 2015. Automaatiotestaus. Opinnäytetyö, AMK. Laurea-ammattikorkeakoulu, tietojenkäsittelyn koulutusohjelma, degree programme in Business Information Technology. Viitattu 8.9.2018 http://www.theseus.fi/bitstream/handle/10024/101637/Automaatiotestaus.pdf?sequence=1&isAllowed=y.

Osuch, B. 2018. Top 8 test automation challenges and how to solve them. Viitattu 22.9.2018 https://testflo.com/en/top-8-test-automation-challenges-and-how-to-solve-them/.

Robot Framework User Guide 2017. Viitattu 12.3.2018 http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#getting-started.

Robot Framework / Users 2018. Viitattu 1.11.2018 http://robotframework.org/#users.

Vo, T. 2017. Top 5 Challenges in Test Automation. Viitattu 22.9.2018 https://dzone.com/articles/top-five-challenges-in-test-automation-1.

Yang C. 2018. Top 5 Challenges of Testing Automation. Viitattu 22.9.2018 https://dzone.com/articles/top-5-challenges-of-testing-automation.