



SAVONIA

OPINNÄYTETYÖ - AMMATTIKORKEAKOULUTUTKINTO
TEKNIIKAN JA LIIKENTEEN ALA

FIMLAB LABORATORIOT OY:N TESTAUSAUTOMAA- TION KEHITTÄMINEN

TEKIJÄ: Toni Hynynen

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Toni Hynynen	
Työn nimi Fimlab Laboratoriot Oy:n testausautomaation kehittäminen	
Päiväys 24.4.2020	Sivumäärä/Liitteet 35
Toimeksiantaja/Yhteistyökumppani(t) Fimlab Laboratoriot Oy	
Tiivistelmä <p>Opinnäytetyön tavoitteena oli suunnitella ja kehittää testausautomaatiota Fimlab Laboratoriot Oy:lle hyödyntäen Robot Framework -testausautomaatiokehystä ja SeleniumLibrary -kirjastoa. Testausautomaation avulla pyrittiin automatisoimaan yrityksen Web-sovelluksien käyttöliittymän testausta ja parantamaan niiden testausprosessia manuaalitestauksen ohella. Automaatiotestaus tuo manuaalitestauksen lisäksi lisää luotettavuutta ja varmuutta sovelluksien ominaisuuksien toiminnan varmuuteen.</p> <p>Opinnäytetyössä hyödynnettiin Robot Framework -testausautomaatiokehystä, SeleniumLibrary -kirjastoa, Jiraa ja Jenkins teknologioita ja palveluita. Robot Frameworkilla luotiin automaatiotestit yrityksen Web-sovelluksien käyttöliittymien ja käytettävien järjestelmien yhteistoiminnan varmistamiseksi hyödyntäen SeleniumLibrary -kirjastoa. Jenkins palvelin mahdollisti automaatiotestien käynnistämisen automaattisesti ja niiden tuloksien synkronoinnin Jiran Xray testisuoritteelle.</p> <p>Opinnäytetyön lopputuloksena saatiin rakennettua yritykselle toimiva automaatiotestauskokonaisuus, jonka avulla voitiin suorittaa automaatiotestit ajastetusti, kun sille asetetut käynnistysehdot toteutuvat. Lopuksi voidaan todeta, että automaatiotestaus soveltuu hyvin yrityksen käyttämien Web-sovelluksien testaamiseen ja täten vapauttavat yrityksen ohjelmistotestaajien työaikaa ja tuovat varmuutta sovelluksien toiminnan varmistamiseen.</p>	
Avainsanat Automaatiotestaus, ohjelmistotestaus, Robot Framework, SeleniumLibrary	

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Toni Hynynen			
Title of Thesis Development of Test Automation for Fimlab Laboratories			
Date	24 April 2020	Pages/Appendices	35
Client Organisation /Partners Fimlab Laboratoriot Oy			
<p>Abstract</p> <p>The purpose of this thesis was to design and develop test automation for Fimlab Laboratories using the open source Robot Framework automation framework and SeleniumLibrary. The purpose of the developed test automation is to automate the testing of the user interface of the company's web applications and to improve their testing process alongside manual testing. Adding automated tests to the testing process improves the reliability of the whole process which ensures the correct functionality of the applications.</p> <p>The Robot Framework test automation framework, SeleniumLibrary, Jira and Jenkins technologies and services were utilized in the thesis. Robot Framework and its SeleniumLibrary were used to create automated tests that ensure the correct co-operation of the company's different web applications and systems. The Jenkins server allowed the automated tests to be executed automatically and their results to be synchronized to Jira Xray.</p> <p>As a result, a functional automation testing system was built for the company which could be used to execute automation tests in a timed manner when certain start-up conditions are fulfilled. It can be concluded that automated testing is suitable for testing the web applications used by the company, and it significantly decreases the manual testing workload of the company's software testers and brings more certainty to ensuring the correct function of the applications.</p>			
<p>Keywords test automation, software testing, Robot Framework, SeleniumLibrary</p>			

SISÄLTÖ

LYHENTEET JA TERMIT	6
1 JOHDANTO	7
2 JÄRJESTELMÄKUVAUS	8
2.1 Prosessi	8
2.2 Health Level Seven	9
2.2.1 Health Level Seven Version 2.x.....	9
2.2.2 Health Level Seven Version 3	9
2.3 Jira	10
2.3.1 Octopus.....	10
2.3.2 XRAY.....	11
3 OHJELMISTOTESTAUS	13
3.1 TESTAUSTASOT.....	14
3.1.1 Yksikkötestaus	14
3.1.2 Integrointitestaus	14
3.1.3 Järjestelmätestaus.....	15
3.1.4 Hyväksymistestaus	15
3.2 Testausmenetelmät.....	15
3.2.1 Musta laatikko.....	15
3.2.2 Lasilaatikko	16
3.2.3 Harmaa laatikko	16
3.2.4 Regressio.....	17
3.2.5 Tutkiva	17
3.3 Testausautomaatio.....	17
4 ROBOT FRAMEWORK.....	19
4.1 Robot Framework	19
4.2 Asentaminen	20
4.2.1 Windows	20
4.2.2 Ubuntu	22
4.3 Rakenne.....	24

4.4	Suorittaminen ja raportit.....	26
4.5	Jenkins	29
5	TESTITAPAUKSET.....	30
5.1	Testaussuorite	30
5.2	Automaatiotesti	31
6	TULOKSET JA POHDINTA.....	33
7	LÄHDELUETTELO.....	34

LYHENTEET JA TERMIT

HTML	Lyhenne sanoista Hypertext Markup Language. Verkkosivujen standardi merkintäkieli.
XML	Lyhenne sanoista Extensible Markup Language eli merkintäkieli.
Xpath	Lyhenne sanoista XML Path Language. Xpath mahdollistaa elementtien löytämisen web-sivuilta.
SeleniumLibrary	Verkkosivujen testaamiseen luotu kirjasto Robot Frameworkille.
Python	Korkean tason ohjelmointikieli.
PIP	Paketin hallintajärjestelmä Pythonille.
RIDE	Sovellus, jolla voi kirjoittaa Robot Framework testejä.
JVM	Lyhenne sanoista Java Virtual Machine. Virtuaalikone, joka suorittaa käännettyjä Java-ohjelmia.
.NET	Microsoftin kehittämä ohjelmistokomponenttikirjasto.
LIS	Lyhenne sanoista Laboratory Information Systems. Laboratorion tietojärjestelmä.

1 JOHDANTO

Automaatiotestaaminen on yksi ohjelmistotestauksen muodoista ja sillä tarkoitetaan sovelluksen testaamista varten kehitettäviä, automaattisesti toimivia työvälineitä, kuten robotilla haluttujen testien suorittamista. Oikein rakennetulla sekä ajoitetulla automaatiotestaamisella saavutetaan hyötyjä, kun esimerkiksi samoja regressiotestejä suoritetaan useita kertoja uudelleen. Automaatiotestaamisella mahdollistetaan ohjelmistotestaajien ja -kehittäjien ajan säästöä, mikä tarkoittaa henkilöresurssien säästöä (Kasurinen, 2013, ss. 76-79).

Tässä opinnäytetyössä suunniteltiin ja toteutettiin automaatiotestausta Fimlab Laboratoriot Oy:lle hyödyntäen Robot Framework -testausautomaatiokehystä, SeleniumLibrary -kirjastoa, Pythonia, Jiraa ja Jenkinsiä. Testitapauksien automatisoinnilla oli tarkoitus automatisoida useita erilaisia testitapauksia yrityksen käyttämien Web-sovelluksien käyttöliittymän ja palveluiden testaamiseen. Opinnäytetyön tavoitteena oli rakentaa automaatiotestauskokonaisuus siten, että automaatiotestit käynnistyvät, kun Jenkins -palvelimen työlle asetetut käynnistysehdot tai sille asetettu ajankohta täyttyy.

Työssä käydään yleisellä tasolla läpi ohjelmistotestauksen teoriaa. Kyseisessä teoriaosuudessa käsitellään ohjelmistotestauksen V-mallissa esiintyvät testaustasot sekä yleisimmät testausmenetelmät. Ohjelmistotestauksen teoriaosuuden jälkeen syvennytään tarkemmin Robot Frameworkin -teoriaan, automaatiotestien rakenteeseen, testien suorittamiseen, tulosraportteihin ja Jenkins -integraatioon.

Opinnäytetyön tilaajana toimi Fimlab Laboratoriot Oy, joka on suomalainen laboratorioalan yritys. Yritys toimii Pirkanmaalla, Päijät-Hämeessä, Kanta-Hämeessä ja Keski-Suomessa yli 100 toimipisteessä lähes 1100 työntekijän voimin ja on Suomen suurin laboratorioalan yritys. (Fimlab Laboratoriot Oy, 2020) Yrityksellä on oma tietohallinto, jossa työskentelee noin 30 IT-alan ammattilaista.

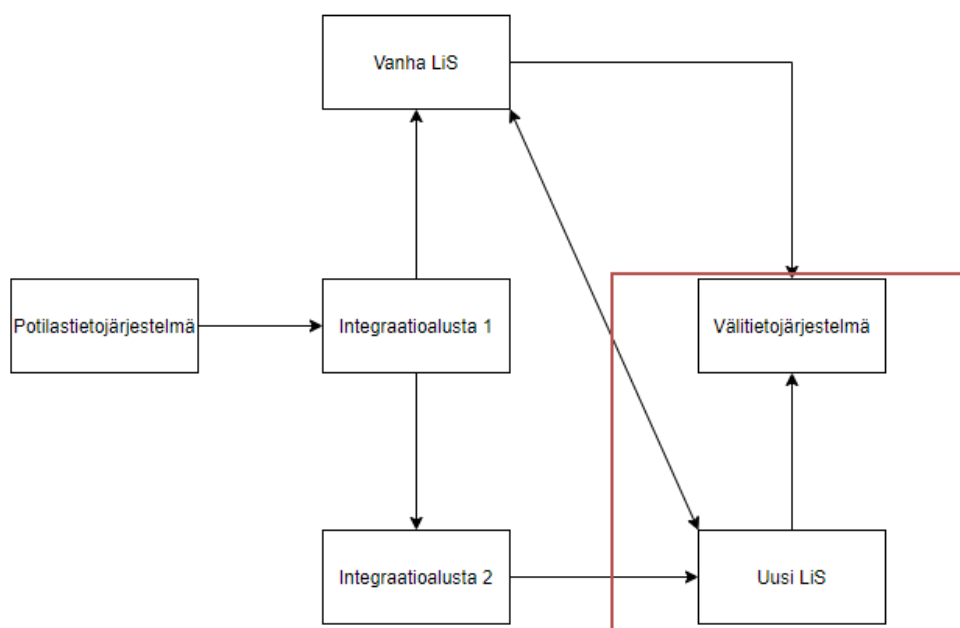
2 JÄRJESTELMÄKUVAUS

Järjestelmäkuvaus luvussa käydään läpi opinnäytöhön liittyviä keskeisiä käsitteitä ja käytettäviä järjestelmiä.

2.1 Prosessi

Laboratorionäytteen prosessi alkaa potilastietojärjestelmästä tehdystä tilauksesta, mistä Health Level Seven (HL7) sanoma siirtyy integraatioalustoille, laboratorion tietojärjestelmiin ja välitietojärjestelmiin (KUVA 1). Integraatioalustoilla HL7 sanomalle tehdään tarkastuksia, sekä tarvittavia muunnoksia riippuen tilatusta laborationäytteestä ja sen tilaus- ja kohdejärjestelmästä. Integraatioalustat tutkivat HL7 sanoman ja reitittävät sen vanhaan tai uuteen laboratoriojärjestelmään riippuen sanomasta. Prosessin lopuksi välitietojärjestelmät lähettävät potilastietojärjestelmälle HL7 vastaussanomana, minkä avulla laboratoriotutkimuksen tulos saadaan välitettyä takaisin potilastietojärjestelmään.

Opinnäytetyössä suunniteltiin ja toteutettiin pääasiassa regressiotestien automatisointia liit-
tyen uuden laboratorion tietojärjestelmän ja välitietojärjestelmän (engl. Middleware) välille.



KUVA 1. Osa laboratorionäytteen prosessista.

2.2 Health Level Seven

Health Level Seven International eli HL7 on terveydenhuollon standardeja kehittävä Yhdysvaltalainen voittoa tavoittelematon organisaatio, jonka tavoitteena on tarjota standardeja sähköisen terveystietojen siirtoa varten eri järjestelmien ja integraatioiden välille. Standardeja on kehitetty useisiin eri käyttötarkoituksiin terveydenhuoltoa varten ja yleisiä standardeja ovat:

- HL7 Version 2.x
- HL7 Version 3
- HL7 SPL (Structured Product Labeling)
- HL7 CCOW (Clinical Context Object Workgroup)
- HL7 CDA (Continuity of Care Document) (Datica, 2019)

Health Level Seven Finland ry on perustettu vuonna 1995 ja sen tavoitteena on edistää Suomen terveydenhuollossa käytettävien tietojärjestelmien kehitystä sekä niihin liittyviä standardien käyttöä. (HL7 Finland, ei pvm)

2.2.1 Health Level Seven Version 2.x

Health Level Seven Version 2 eli HL7 V2 on vuonna 1987 kehitetty sanomanvälitysstandardi, mikä mahdollistaa tietojen välityksen järjestelmien välillä. Standardi on vielä nykypäivänä hyvin laajalti käytössä ja esimerkiksi Yhdysvalloissa yli 90% terveydenhuollon järjestelmistä hyödyntää sitä, vaikka uudempi HL7 Version 3 on julkaistu vuonna 2005. (HL7 International, ei pvm). Suomessa terveydenhuolto hyödyntää HL7 versiota 2.3, esimerkiksi laboratoriosanomissa (KUVA 2)

```
MSH|^~\&|TAMLAB^TAMLABTES^|TAMLAB^PSHP|EFFICA^EFFICATES|EFFICA^JAMSA|20200322133319||ORU^R01|
PID|1|070707-0707^^^^HETU|||Testipotilas^Testi^Tesi||19070707|2||^182|
PV1|||kiros|
ORC|RE|||CM|||20200322133000|
OBR|1|625288-2793380|0030295^FL20|1999^P -K^KLNRO|||20200322133000|20200322133000||L|
OBX|1|NM|1999^P -K^KLNRO||3.8|mmo1/1|3.3-4.8|N||F||20200322133000||
```

KUVA 2. Esimerkki HL7 Version 2.3 laboratoriosanomasta.

2.2.2 Health Level Seven Version 3

Health Level Seven Version 3 eli HL7 V3 julkaistiin vuonna 2005 ja sillä tavoiteltiin HL7 V2.X standardissa oleviin yksilöityihin haasteisiin ratkaisua. Uudella HL7 V3 tavoiteltiin yhdenmu-kaista tietomallia ja käytön lisäämistä maailmanlaajuisesti, vaikka se ei ole yhteensopiva HL7 V2.X kanssa. Pääasiassa HL7 V3 on otettu käyttöön uudemmissa terveydenhuollon järjestelmissä ja se perustuu XML-pohjaiseen rakenteeseen (KUVA 3). (Lyniate, ei pvm)

```

<POLB_IN224200 ITSVersion="XML_1.0" xmlns="urn:h17-org:v3"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <id root="2.16.840.1.113883.19.1122.7" extension="CNTRL-3456"/>
  <creationTime value="200202150930-0400"/>
  <!-- The version of the datatypes/RIM/vocabulary used is that of May 2006 -->
  <versionCode code="2006-05"/>
  <!-- interaction id= Observation Event Complete, w/o Receiver Responsibilities -->
  <interactionId root="2.16.840.1.113883.1.6" extension="POLB_IN224200"/>
  <processingCode code="P"/>
  <processingModeCode nullFlavor="OTH"/>
  <acceptAckCode code="ER"/>
  <receiver typeCode="RCV">
    <device classCode="DEV" determinerCode="INSTANCE">
      <id extension="GHH LAB" root="2.16.840.1.113883.19.1122.1"/>
      <asLocatedEntity classCode="LOCE">
        <location classCode="PLC" determinerCode="INSTANCE">
          <id root="2.16.840.1.113883.19.1122.2" extension="ELAB-3"/>
        </location>
      </asLocatedEntity>
    </device>
  </receiver>
  <sender typeCode="SND">
    <device classCode="DEV" determinerCode="INSTANCE">
      <id root="2.16.840.1.113883.19.1122.1" extension="GHH OE"/>
      <asLocatedEntity classCode="LOCE">
        <location classCode="PLC" determinerCode="INSTANCE">
          <id root="2.16.840.1.113883.19.1122.2" extension="BLDG24"/>
        </location>
      </asLocatedEntity>
    </device>
  </sender>
  <!-- Trigger Event Control Act & Domain Content -->
</POLB_IN224200>

```

KUVA 3. Esimerkki HL7 V3 -sanomasta (Wikipedia, 2020).

2.3 Jira

Jira on vuonna 2002 julkaistu Atlassianin kehittämä tehtävienhallintasovellus, joka on laajalti käytössä eri yrityksissä ympäri maailmaa. Jira kehitettiin alun perin ohjelmistossa esiintyvien virheiden seurantaan, mutta nykyään sen toimintoja on kehitetty erityisesti projektityökäluna. Jiraa käytetäänkin nykyään muun muassa ohjelmistokehityksessä, projektinhallinnassa ja asiakastuessa. (Tieturi, ei pvm)

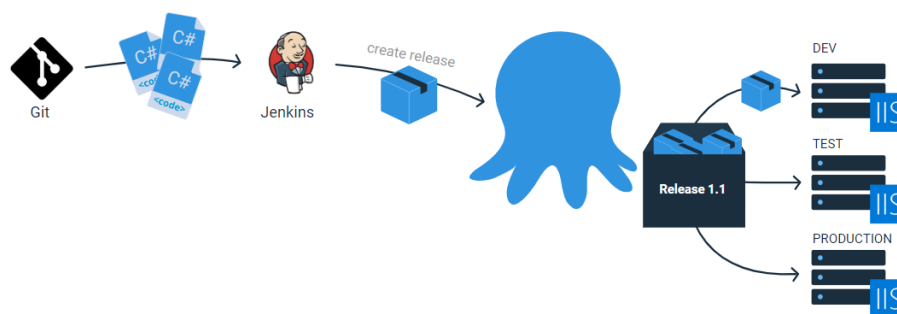
Jira voidaan jakaa neljään eri pääsääntöiseen kokonaisuuteen:

- Jira Software on sovelluskehittäjille rakennettu työkalu, mikä mahdollistaa kehittäjille käytettäväksi mm. suunnittelualustan, seuranta-alustan ja julkaisualustan.
- Jira Service Desk on yrityksen palvelupyyntöjen hallintaan rakennettu ohjelmisto.
- Jira Core mahdollistaa projektien liiketoiminnan seuraamista useasta näkökulmasta.
- Jira Opsgenie on tapahtumienhallintaa seuraava sovellus, jonka avulla seurataan eri järjestelmissä tulleita virheitä tai hälytyksiä. Sovellus tarjoaa myös laajat raportointi- ja analytiikkatiedot. (Atlassian, ei pvm)

2.3.1 Octopus

Octopus Deploy on automaattinen julkaisuihin ja käyttöönottoon liittyvä palvelin. Octopus Deploy on osa Jenkins tuoteperhettä, joka on tällä hetkellä yksi suosituimpia jatkuvan integraation palvelinohjelmistoja (engl. Continuous Integration, CI).

Octopus Deploy julkaisualusta mahdollistaa monipuoliset hallinta- ja ylläpitotyökalut kehitys-, testi- ja tuotantoympäristöille. (KUVA 4). (Octopus Deploy, ei pvm)



KUVA 4. Octopus toimintaperiaate (Octopus Deploy, ei pvm).

2.3.2 XRAY

XRAY on yksi useista Jiraan liitettävistä lisäosista. XRAY on rakennettu testitapauksien hallintaan, ohjelmistojen testaukseen ja laadunvalvontaan. Lisäosa on yksi Jiran suosituimpia manuaali- ja automatisoitujen testitapauksien laadunvalvontaan tarkoitettu laajennus. Testausprosessia varten XRAY tukee testausta koko sen elinkaaren ajan testisuunnitelmasta aina testiraportteihin asti. Testausprosessi sisältää testisuunnitelman, testikokoelman, testitapauksen, testisuorituksen ja testiraportin. (Atlassian, ei pvm)

Testikokoelma (engl. Test Set) sisältää kokoelman yksittäisiä testitapauksia, jotka liittyvät samaan testikokoelmaan tai -projektiin. Testikokoelmaa rakennettaessa sille voidaan antaa useita eri asetuksia, sekä määrittää tietty vastuuhenkilö. Yksittäistä testiä (engl. Test) luodessa testille voidaan antaa useita eri ehtoja, kuten testityypin valinta manuaali- tai automaatiotestin välillä tai testitapauksen suoritusväli. Yksittäiset testit linkitetään testikokoelmaan.

Testisuunnitelman (engl. Test Plan) avulla suunnitellaan ja dokumentoidaan suoritettava testisuorite. Testaussuunnitelmaan linkitetään testikokoelma tai harvinaisessa tapauksessa yksittäinen testitapaus. Testaussuunnitelman tarkoitus on olla selkeä dokumentaatio, joka kertoo mitä testikokoelmassa testataan, kuka sen testaa ja millä aikavälillä testaus tapahtuu.

Testisuoritteelle (engl. Test Execution) linkittyy testisuunnitelmaan linkitettyt testit tai testikokoelmat riippuen siitä mitä testisuunnitelma sisältää (KUVA 5). Testaaminen aloitetaan testisuoritteelta siinä tapauksessa, jos testitapaus on tyypiltään manuaalitestit. Manuaalitesteissä testitapauksen tulos annetaan käyttöliittymästä, sen mukaan läpäisikö testi sille asetut ehdot vai ei.

	Rank	Key	Summary	Test Type	#Req	#Def	Test Sets	Assignee	Status	
	1	LAB20-25	Kuittaukset: Koppakuittauksen perusprosessi	Manual	1	9	LAB20-8	Toni Hynynen	FAIL	 ...
	2	LAB20-453	Kopan valinta	Manual	0	0	LAB20-451	Toni Hynynen	PASS	 ...
	3	LAB20-452	Koppa tunnisteen tarkistus	Manual	0	0	LAB20-451	Toni Hynynen	PASS	 ...
	4	LAB20-287	Kuittaukset: Koppanimen käsittely Case Insensiivisenä	Manual	0	2	LAB20-8	Toni Hynynen	FAIL	 ...

Showing 1 to 4 of 4 entries

First Previous **1** Next Last

KUVA 5. XRAY manuaalitestisuorite.

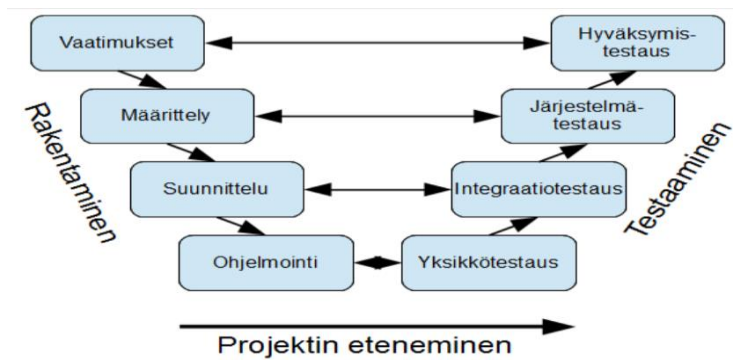
3 OHJELMISTOTESTAUS

Ohjelmistotestaus kuuluu ohjelmistotuotannon piiriin. Ohjelmistotestaus on työtä, jolla varmistetaan, että tehdään oikeaa tuotetta ja että se on toteutettu oikein. Testaus on kokonaisuutena laaja käsite, koska testaaja joutuu tekemään työssään erilaisia toimenpiteitä ja tämän vuoksi testaajan ammatti voi vaihdella huomattavasti eri ohjelmistotalojen välillä. (Kasurinen, 2013, ss. 10-15)

Ohjelmistotestaus on kehittynyt 1950-luvulta asti ja kehittyä edelleen, vaikka testauksesta on tullut monimutkaisempaa verrattuna sen ajan ongelmiin. Tänä päivänä ohjelmistotestauksessa puhutaan suurista työ- ja rahamääristä. Esimerkiksi vuonna 2002 tehdyn tutkimuksen (Tassej 2002) mukaan yhdysvaltalaiset ohjelmistoyritykset ja näiden yritysten asiakkaat menettivät 59.5 miljardia dollaria puutteellisen ja vajavaisen testauksen takia. Summaan laskettiin ohjelmistotalojen suorien tappioiden lisäksi myös asiakkaille koituneet tappiot, vahingot ja tuotannonmenetykset. (Kasurinen, 2013, ss. 10-15)

Testaustyötä voidaan tehdä muun muassa perinteisenä vesiputousmallina, jossa testaus on vain yksi työvaihe toteutuksen ja käyttöönoton välissä. V-malli on puolestaan parempi malli kehitystyölle, testauksen liittymispinnoille ja sen tapahtumiin (KUVA 6). V-mallissa testaus-toimintaa on jo toteutusvaiheen aikana ja se jatkuu projektin loppuun asti. Laadunvalvonta samalla tarkastaa, että tuote toimii halutulla tavalla. V-mallissa myös jokaiselle suunnittelun vaiheelle on oma testausmenetelmä, joilla varmistetaan tuotteen laatu. (Kasurinen, 2013, ss. 10-15)

V-mallissa rakentamispuoli toimii kuin edellä mainittu vesiputousmalli, mutta testauksen toiminnot ovat saaneet enemmän merkitystä tapahtumiin. Tässä mallissa testaus ei ole erillinen työvaihe, vaan jokaiselle rakennusvaiheelle on merkitty oma erillinen testauskategoria. Ohjelman läpäistyä kaikki testauksen tasot, se on valmis käyttöönottoon ja siirrettäväksi ylläpitoon. (Kasurinen, 2013, ss. 10-15)



KUVA 6. Testauksessa käytettävä V-malli (Kasurinen, 2013, s. 14).

3.1 TESTAUSTASOT

3.1.1 Yksikkötestaus

Yksikkötestaus (engl. Unit Testing) on ohjelmistotestauksen taso, jonka avulla testataan ohjelmistossa olevia yksittäisiä komponentteja, funktioita tai olioita. Yksikkötestausta tehdään jo yleensä sovelluksen kehittämissä vaiheissa ja kehittäjät suorittavat sen. Mahdolliset virheet ovat täten helposti korjattavissa, kun taas myöhemmässä vaiheessa korjaaminen on huomattavasti vaikeampaa. (Kasurinen, 2013, ss. 51-54)

Yksikkötestaukseen liittyvät edut nähdäänkin, kun koodiin tehdään usein korjauksia. Monipuoliset yksikkötestit ovatkin tärkeitä muun muassa Scrum ja Agile kehitysmenetelmissä. (University Of Jyväskylä, 2019)

3.1.2 Integrointitestaus

Integraatiotestaus (engl. Integration Testing) testaustapana testaa sovelluksessa käytettävien komponenttien tai rajapintojen yhteistoimintaa. Komponenttien ja rajapintojen välisten integraatioiden väliltä pyritään tavoittamaan sellaisia virheitä, mitkä eivät ole tulleet esiin muissa testauksissa. (University Of Jyväskylä, 2019)

Integrointitesteissä on kolme erilaista tapaa lähestyä:

- Ylhäältä alaspäin (engl. Top Down testing)
- Alhaalta ylöspäin (engl. Bottom Up testing)
- Hybridi-integraatiotestaus (engl. Sandwich testing)

Ylhäältä alaspäin lähestymistapa integrointitestaukseen tarkoittaa sitä, että testaus aloitetaan järjestelmän ylimmän tason moduuleista ja testausprosessi etenee alaspäin kohti alinta tasoa. Tällä testaustavalla hahmotetaan järjestelmästä yleiskuva, jonka avulla voidaan parhaimmassa tapauksessa huomata sovelluksesta puuttuvia ominaisuuksia. (Kasurinen, 2013, ss. 54-55)

Alhaalta ylöspäin lähestymistapa integrointitestaukseen tarkoittaa sitä, että testaus aloitetaan matalimman tason moduuleista. Testitapauksien edetessä sovellukseen lisätään uusia moduuleita tai rajapintoja, jotka hyödyntävät edellisistä testeistä läpäisseitä toimivia moduuleita. Lähestymistavalla pyritään löytämään järjestelmästä mahdollisia virheitä. (Kasurinen, 2013, ss. 54-55)

Hybridi-integraatiotestaus tunnetaan myös nimellä voileipätestauksena. Tämä lähestymistapa integrointitestaukseen tarkoittaa sitä, että edellä mainitut ylhäältä alaspäin-, ja alhaalta ylöspäin lähestymistavat yhdistetään. Moduuli tai rajapinta integraatiotestausta lähestytään molemmista suunnista. (GeeksforGeeks, ei pvm)

3.1.3 Järjestelmätestaus

Järjestelmätestaus (engl. System Testing) testaustavassa testataan yksikkötesteistä ja integraatiotesteistä läpäisseitä sovelluksen komponentteja yhtenä kokonaisuutena eli järjestelmänä (Kasurinen, 2013, s. 56).

Järjestelmätestaukseen ei yleensä liity minkäänlaista yksittäistä testaustapaa. Järjestelmätestauksessa voidaan hyödyntää esimerkiksi harmaa laatikko -testausta (engl. Grey Box Testing). Järjestelmätestauksen tarkoituksena on varmistaa, että järjestelmä toimii yhtenä kokonaisuutena. Testaus tapahtuu tässä vaiheessa vielä testiympäristössä eikä asiakkaan ympäristössä. (Kasurinen, 2013, ss. 56-57)

3.1.4 Hyväksymistestaus

Hyväksymistestaus (engl. Acceptance Testing) testaustapana on esitettävän V-mallin neljäs sekä viimeinen testaustapa (KUVA 6). Hyväksymistestausta käytetään tarkastamaan sitä, että täyttääkö sovellus asiakkaan asettamat vaatimukset sovellukselle. Tässä vaiheessa järjestelmän testausta tarkastellaan siitä näkökulmasta, missä sovellus tulee olemaan asiakkaan ympäristössä. Asiakas on testausprosessissa mukana, jolloin saadaan varmuus siitä, hyväksyykö asiakas sovelluksen ja onko se suunnitelman mukainen. (Kasurinen, 2013, ss. 57-59)

3.2 Testausmenetelmät

Testausmenetelmät kehitysvaiheessa noudattavat testauksen V-mallin mukaista lähestymistapaa (KUVA 6). Lähestymistavassa ensimmäisenä suoritetaan yksikkötestaus komponenteille, jonka jälkeen komponentit liitetään toisiinsa integrointitestauksessa ja viimeisenä järjestelmätestauksessa, ennen sen siirtämistä hyväksyttäväksi. Testausta voidaan suorittaa usealla eri tavalla ja seuraavaksi käydään läpi erilaisia testausmenetelmiä. (Kasurinen, 2013, s. 64)

3.2.1 Musta laatikko

Musta laatikko -testaus (engl. Black Box Testing) testausmenetelmän avulla testattava järjestelmä testataan siten, että sovelluksen toimintalogiikka on testaajalle täysin tuntematon (KUVA 7). Tällaisella testausmuodolla testataan järjestelmää siitä näkökulmasta, miten sen

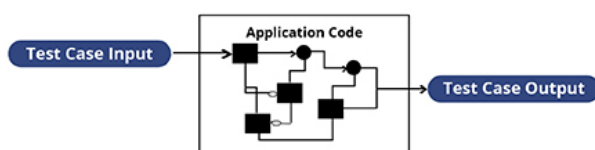
täytyisi toimia. Esimerkiksi verkkosivulla on lähetettävä lomake, jossa on kenttä mihin syötetään ikä. Testaaja ei tiedä miten kyseinen komponentti on ohjelmoitu, mutta testaaja testaa kentän toimivuuden. (Kasurinen, 2013, s. 65)



KUVA 7. Musta laatikko -testauksen periaate (Guru99, ei pvm).

3.2.2 Lasilaatikko

Lasilaatikkotestaus (engl. White Box Testing) testausmenetelmän avulla järjestelmä testataan samalla periaatteella kuin musta laatikko -testauksessa, mutta testaajalla on pääsy testattavan järjestelmän rakenteisiin ja sovelluksessa käytettävään koodiin (KUVA 8). Lasilaatikkotestauksessa annetaan järjestelmälle syötteitä ja tarkastellaan mitä järjestelmän sisällä tapahtui ohjelmakoodin perusteella. Tämän testaustavan avulla voidaan virhetilanteessa seurata kooditasolla, mikä virheen aiheutti. (Kasurinen, 2013, s. 67)



KUVA 8. Lasilaatikkotestauksen periaate (Rongala, 2015).

3.2.3 Harmaa laatikko

Harmaalaatikko -testauksen (engl. Grey Box Testing) periaate on yhdistelmä lasilaatikko- ja mustalaatikko -testauksen periaatteita (KUVA 9). Testitavassa on otettu molemmista testitavoista parhaat puolet, muodostamalla niistä harmaalaatikko -testauksen. Testitapauksessa järjestelmää tarkastellaan sisäpuolelta, kuten lasilaatikkotestauksen tapauksissa on periaatteena. Tällainen testitapa on ideaalinen sellaisiin tilanteisiin, jossa ei tarkastella ohjelman kooditasoa, mutta komponenttien toimintaa tarkastellaan toimintatasolla. (Kasurinen, 2013, s. 68)



KUVA 9. Harmaalaatikko -testauksen periaate (Guru99, ei pvm).

3.2.4 Regressio

Regressiotestaus (engl. Regression Testing) tarkoittaa järjestelmän uudelleentestaamista. Regressiotestaustapaan viitataan, kun toimivaan järjestelmään on tehty muutoksia ja sen toimivuus halutaan tarkastaa uudelleen. (Kasurinen, 2013, ss. 68-69)

Regressiotestausta voidaan hyödyntää esimerkiksi sellaisissa tilanteissa, missä kaksi eri kehityshaaraa yhdistetään yhdeksi haaraksi versionhallinnassa tai käyttöliittymään tehdään jokin muu järjestelmään liittyvä päivitys. Tällaisissa tilanteissa regressiotestauksella varmistetaan, että järjestelmässä olleet vanhat toiminnot toimivat ja että uudet lisätyt toiminnot, sekä ominaisuudet toimivat yhtenä kokonaisuutena. (Kasurinen, 2013, ss. 68 - 69)

3.2.5 Tutkiva

Tutkiva testaaminen (engl. Explorative Testing) on testausmuoto, mikä perustuu ongelmien etsimiseen ja löytämiseen järjestelmätestauksen tasolla. Tutkivassa testaamisessa hyödynnetään ohjelmistotestaajien kerryttämää kokemusta ja tietotaitoa sellaisissa tilanteissa, missä ohjelmistojen virheet tulevat yleensä järjestelmässä tai komponentin toiminnassa esille tai miten virhetilanteet syntyvät. Tutkiva testaaminen on saanut epäammattimaisen ja huonon maineen sen vuoksi, että tutkivalle testaukselle ei ole virallisia standardeja ja lähestymistavat virhetilanteiden löytämiseen ovat henkilökohtaisia. (Kasurinen, 2013, s. 74)

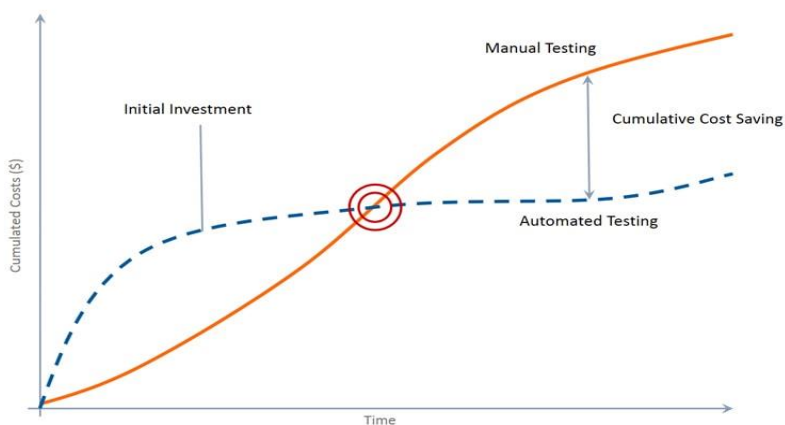
3.3 Testausautomaatio

Testausautomaatiolla tarkoitetaan testauksen muotoa, missä sovelluksen testaamista varten rakennetaan automaattisesti toimivia työkaluja, kuten robotti testien suorittamista varten. Tämän avulla pyritään luomaan kokoelma toistuvia testitapauksia ja täten ohjelmistotestaajat voivat keskittyä muihin työtehtäviin. (Kasurinen, 2013, ss. 76-79)

Yksittäiset moduulit ja rajapinnat ovat testausautomaation parhaimpia kohteita testata. Testausautomaatiolla voidaan testata esimerkiksi sovelluksen käyttöliittymästä yksittäisten komponenttien toimintaa. Useasti automaatiotestaamista pidetään manuaalitestauksen korvaajana, mutta asia ei ole näin. Automaatiotestaus täydentää manuaalitestausprosessia ja sovelluksen julkaisuprosessia, eikä se korvaa sitä täysin. (Kasurinen, 2013, ss. 76 - 79)

Testausautomaation ylläpito vaatii paljon resursseja ja ylläpitoa, joten automaatiotestauksen avulla säästö on harhaluulo (KUVA 10). Projektin kannalta on siten tärkeä tunnistaa, millaisissa tilanteissa testausautomaatiota kannattaa hyödyntää ja milloin ei. Testitapaukset ovat yleisesti herkkiä käyttöliittymässä tapahtuville muutoksille tai kooditason muutoksille, joten

sellaisia järjestelmiä ei kannata automatisoida, missä julkaisutahti on nopeaa. Testausvaiheen automatisointi vähentää manuaalitestauksen regressiotestien määrää huomattavasti. Automaatiotestien kirjoittamisen ja ylläpidon tuomasta lisätyömäärästä huolimatta automaatiotesteillä saavutetaan rahallista säästöä vasta tietyn pisteen jälkeen. (Kasurinen, 2013, ss. 76-79)



KUVA 10. Automaatio- ja manuaalitestien kulujakauma (Grinevich, 2019).

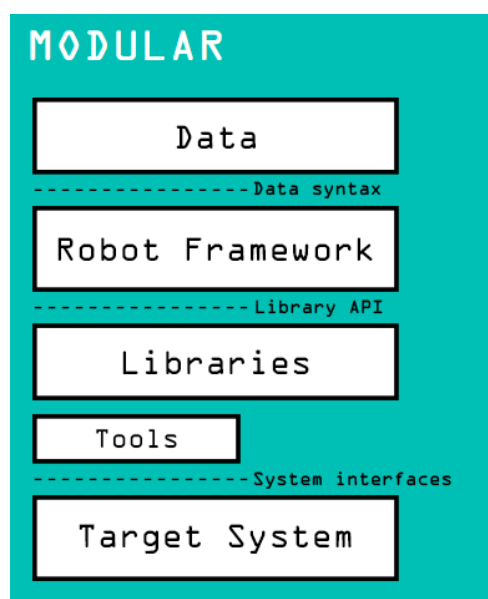
4 ROBOT FRAMEWORK

4.1 Robot Framework

Robot Frameworkin kehitys alkoi Pekka Klärckin kehittämästä diplomityöstä vuonna 2005. Robot Framework on geneerinen testiautomaatioviitekehys hyväksymistestaukseen ja hyväksymisvetoiseen ohjelmistokehitykseen. Robot Frameworkilla rakennettujen testien syntaksi perustuu avainsanapohjaiseen rakenteeseen. (Vala Group, 2017)

Robot Frameworkin etuna on sen perustuvuus avoimeen lähdekoodiin, uudelleen käytettävyys ja sen täysi laajennettavuus Pythonin, Javan (Jython) ja .Net:in (IronPython) kanssa (Robot Framework, ei pvm). Robot Frameworkia voidaan myös laajentaa lisäkirjastojen avulla muun muassa iOS- ja Android-sovelluksien testaamiseen, sekä sitä voidaan hyödyntää sulautetuissa järjestelmissä. (Eficode, 2016)

Robot Frameworkin toiminta perustuu kuvattuun Robot Frameworkin arkkitehtuuriin (KUVA 11). Arkkitehtuurin yläosassa on käyttäjän kirjoittama testitapaus (Data) ja Robot Framework suorittaa testit (Robot Framework) ja muodostaa suoritetusta testistä tulosraportin ja lokitiedoston. Ydinkehys ei tiedä testitapauksesta tai suoritettavasta kohteesta mitään, kun testitapauksessa olevat kirjastot (Libraries) keskustelevat suoraan suoritettavan testin kanssa. Testitapauksissa käytettävät kirjastot voivat käyttää sovellusrajapintoja suoraan tai käyttää alemman tason testityökaluja ohjaiminaan (Tools). Testattava järjestelmä (Target System) on Robot Frameworkin arkkitehtuurissa alimmaisena tasona. (Robot Framework User Guide, 2019)



KUVA 11. Robot Frameworkin arkkitehtuuri (Robot Framework, ei pvm).

4.2 Asentaminen

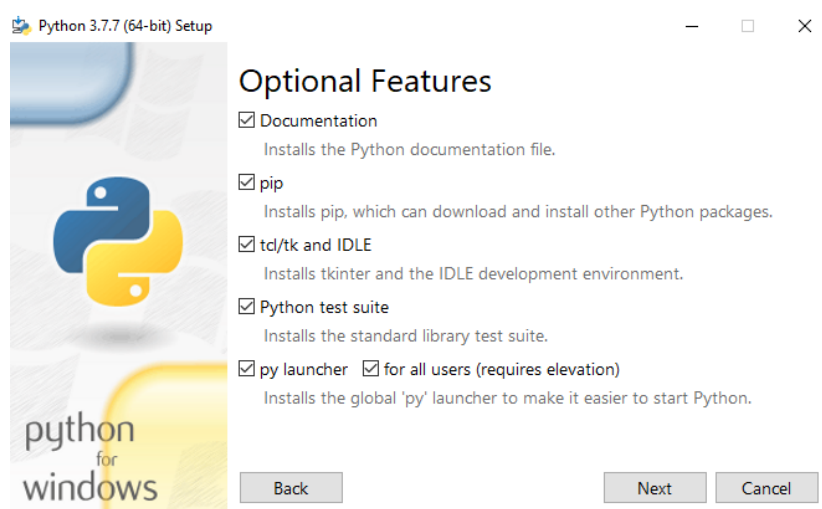
4.2.1 Windows

Tässä kappaleessa käydään läpi Pythonin, Robot Frameworkin, SeleniumLibrary -kirjaston ja RIDE editorin sovelluksen asennusprosessi Windows 10 -käyttöjärjestelmälle.

Ennen Robot Frameworkin asennusprosessin aloittamista on hyvä tiedostaa, mille alustalle Robot Frameworkia lähdetään asentamaan. Robot Framework vaatii ennen asennusta Python 2, Python 3, Jython (JVM) tai Ironpythonin (.NET) asennetuksi tietokoneelle. Opinnäytetyö rakentui Python 3.7.7 pohjalle, joten asennusprosessissa kuvataan Robot Frameworkin asennusta Python 3.7.7 versiolle.

Pythonin asennusprosessin alussa tarkastettiin, löytyikö Python valmiiksi asennettuna tietokoneelta. Windowsin komentokehoteella voidaan tarkastaa tietokoneelle asennettun Pythonin versio komennolla "Python -version". Tässä tapauksessa sitä ei ollut asennettu valmiiksi, joten päädyimme Python 3.7.7 version asennukseen.

Pythonin asennus itsessään on yksinkertainen, mutta siinä täytyy ottaa erityisesti muutama asia huomioon. Pythonin asennuksen aikana asennusohjelma kysyy, halutaanko PIP asentaa samalla Pythonin yhteydessä (KUVA 12). PIP on Pythonille tarkoitettu paketasennus järjestelmä, joten se on välttämätön Robot Frameworkin asennusprosessissa.



KUVA 12. Pythonin asennus.

Pythonin asentamisen jälkeen komentokehoteelta voidaan varmistaa, että Python asentui onnistuneesti tietokoneelle komennolla "Python -version" ja "PIP -version". Opinnäytetyössä Python asennettiin C-levyn juureen, joten komentokehoite ei tunnistanut, että tietokoneelle oli asennettu Python ja PIP.

Windows -käyttöjärjestelmälle täytyy manuaalisesti määrittää Windowsin ympäristömuuttujiin tieto, jos Pythonia ei asenneta asennusohjelman ehdottamaan kansioon. Kuvassa (KUVA 13) nähdään, että Windowsin ympäristömuuttujiin on lisätty tieto Python 3.7.7 asennushakemistosta. Ympäristömuuttujien lisäämisen jälkeen komentokehoite tunnisti, että Python 3.7.7 versio ja PIP 19.2.3 versio oli asennettu onnistuneesti käyttöjärjestelmälle (KUVA 14).



KUVA 13. Windowsin ympäristömuuttujat.

```
C:\Windows\system32>python --version
Python 3.7.7

C:\Windows\system32>pip --version
pip 19.2.3 from c:\python3.7.7\lib\site-packages\pip (python 3.7)

C:\Windows\system32>
```

KUVA 14. Pythonin ja PIPin versiot.

Robot Framework asennetaan Windows käyttöjärjestelmälle Pythonin PIP paketasennus järjestelmän avulla (KUVA 15). Asennus tapahtui Windowsin komentokehoitteen kautta syöttämällä komento "pip install robotframework".

```
C:\Windows\system32>pip install robotframework
Collecting robotframework
  Using cached https://files.pythonhosted.org/packages/22/0f/1b9ffa0c4e59789bd/robotframework-3.1.2-py2.py3-none-any.whl
Installing collected packages: robotframework
Successfully installed robotframework-3.1.2
```

KUVA 15. Robot Framework asennuskomento.

Seuraavana asennusprosessissa on tarvittavan ulkopuolisen Robot Frameworkin kirjaston asentaminen. SeleniumLibrary:n asennusprosessi on samanlainen kuin Robot Frameworkin asennus. Windowsin komentokehoitteeseen syötetään komento "pip install robotframework-seleniumlibrary".

RIDE editorin asennus tapahtui samalla periaatteella, kuin Robot Frameworkin ja SeleniumLibrary -kirjaston asentaminen, Windowsin komentokehötteen kautta "pip install robotframework-ride". Robot Frameworkin testejä voi kirjoittaa millä tahansa tekstieditorilla, mutta RIDE on erityisesti suunniteltu Robot Frameworkin testien kirjoittamiseen.

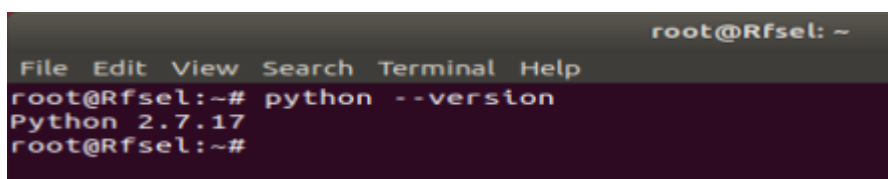
Verkkosivujen automaatiotestit tarvitsevat WebDriver ajurin, jotta SeleniumLibrary -kirjaston testit voidaan suorittaa selaimella. Ajuripaketit ovat selainkohtaisia ja se on saatavilla Mozilla Firefox, Google Chrome, Safari, Edge, Opera ja Internet Explorer selaimille. Opinnäytetyössä on kehitysvaiheessa käytetty Google Chrome ja Mozilla Firefox -selaimia testitapauksissa. Esimerkiksi Google Chromen ajuripaketin löytää ChromeDriver -nimellä ja tämä tukee kirjoitus hetkellä virallisesti selaimen versioita 79.0.39, 80.0.39 ja 81.0.4. Viralliselta lataussivustolta löytyvät myös ohjeet vanhempien selainversioiden käytölle. Mozilla Firefox -selaimen ajuripaketti löytää Googelta nimellä Geckodriver.

Ajurista lisätään tieto Windowsin ympäristömuuttujiin, että Robot Frameworkilla kirjoitetut testit osaavat hyödyntää sitä. Opinnäytetyössä ajuri siirrettiin C:\Python\Scripts kansioon, joten Windowsin ympäristömuuttujiin ei tarvinnut tehdä erikseen muutosta. Windowsin ympäristömuuttujiin oli aikaisemmin määritetty Pythonin asennuskansio, sekä myös sen alikansiot. (KUVA 13)

4.2.2 Ubuntu

Ohessa käydään läpi Pythonin, Robot Frameworkin, SeleniumLibrary -kirjaston ja RIDE editorin -sovelluksen asennusprosessi Ubuntu 18.04.4 LTS käyttöjärjestelmälle.

Ubuntulle asennusprosessi eroaa hieman verrattuna Windows käyttöjärjestelmään. Ubuntussa Python on asennettuna automaattisesti, eikä sitä tarvitse erikseen asentaa, mutta tarvittaessa Ubuntulle voidaan asentaa tietty versio Pythonista. Oletuksena Ubuntu 18.04.4 LTS versiossa oli Pythonin versio 2.7.17 (KUVA 16).



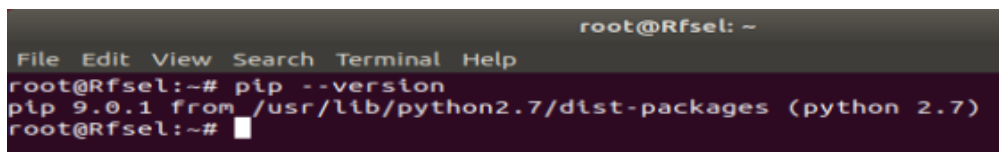
```

root@RfSel: ~
File Edit View Search Terminal Help
root@RfSel:~# python --version
Python 2.7.17
root@RfSel:~#
  
```

KUVA 16. Python versio.

Pythonille tarkoitettu pakettihallintajärjestelmä PIP ei ole oletuksena asennettu. Paketinhallintajärjestelmän kautta asennetaan Robot Framework, sekä siihen liittyvät lisäkirjastot. PIP

saadaan asennettua syöttämällä terminaaliin komento: "sudo apt install python-pip". Asennuksen jälkeen tulee varmistaa, että PIP:n asentaminen onnistui syöttämällä komennon: "pip --version" (KUVA 17).



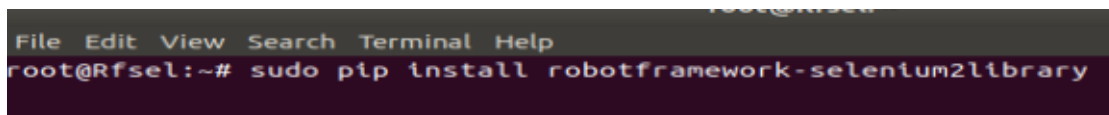
```

root@Rfsel: ~
File Edit View Search Terminal Help
root@Rfsel:~# pip --version
pip 9.0.1 from /usr/lib/python2.7/dist-packages (python 2.7)
root@Rfsel:~#

```

KUVA 17. PIPin versio.

Asentaessa Robot Frameworkia osassa Pythonin versioissa täytyy asentaa WxPython, joka on pakollinen RIDE editorin sovellusta käytettäessä. WxPython asennetaan tietyissä versioissa RIDE:n asennuskomennon kanssa, mutta tässä versiossa se täytyy asentaa erikseen Ubuntun terminaalista komennolla: "sudo apt-get install python-wxgtk2.8". WxPythonin asentamisen jälkeen asennetaan Robot Framework ja Selenium2library -kirjasto. Asennus tapahtuu terminaalin kautta komennolla: "sudo pip install robotframework-selenium2library" (KUVA 18).



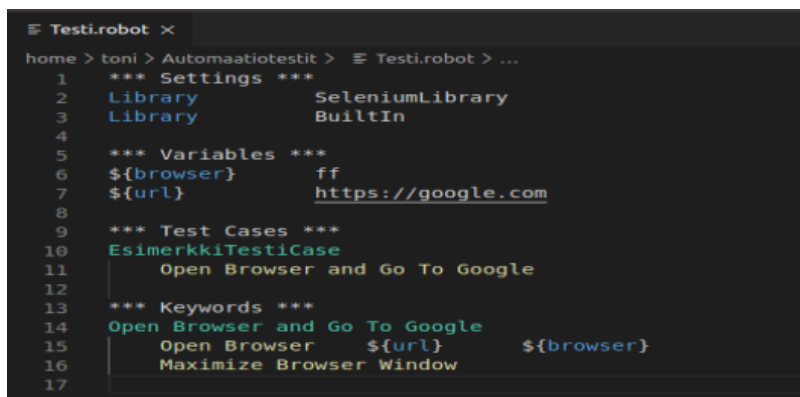
```

root@Rfsel: ~
File Edit View Search Terminal Help
root@Rfsel:~# sudo pip install robotframework-selenium2library

```

KUVA 18. Robot Frameworkin ja Selenium2libraryn asennuskomento.

Robot Frameworkin automaatiotestien kirjoittamiseen voidaan käyttää mitä tahansa tekstieditoria. Ubuntulle asennettiin Visual Studio Code ja siihen Robot Framework Intellisense -laajennus (KUVA 19). RIDE editorin asennus tapahtuu terminaaliin kautta kirjoittamalla komennon: "sudo pip install robotframework-ride".



```

Testi.robot x
home > toni > Automaatiotestit > Testi.robot > ...
1  *** Settings ***
2  Library           SeleniumLibrary
3  Library           BuiltIn
4
5  *** Variables ***
6  ${browser}        ff
7  ${url}            https://google.com
8
9  *** Test Cases ***
10 EsimerkkiTestiCase
11   Open Browser and Go To Google
12
13 *** Keywords ***
14 Open Browser and Go To Google
15   Open Browser    ${url}    ${browser}
16   Maximize Browser Window
17

```

KUVA 19. Esimerkki testistä Visual Studio Codella avattuna.

Verkkosivujen automaatiotestit tarvitsevat WebDriverin, jotta SeleniumLibrary -kirjaston testit voidaan suorittaa selaimella. WebDriverit ovat selainkohtaisia ja Ubuntulle on saatavilla

Google Chrome ja Mozilla Firefox selaimille tarvittavat ajurit. Ubuntulla suoritettavat automaatiotestit käyttivät Mozilla Firefox selainta. Ajurin lataamisen jälkeen se siirrettiin `/usr/local/bin` -kansioon, jonka jälkeen tämä oli käytettävissä testitapauksissa.

4.3 Rakenne

Asetuksilla (engl. Settings) määritetään, mitä lisäkirjastoja testi käyttää. Lisäkirjasto saadaan käyttöön kirjoittamalla "Library" sekä halutun kirjaston nimi (KUVA 20). Lisäkirjastoja voidaan jakaa kahteen eri kategoriaan. Standardi (engl. Standard) kirjastoja käytettäessä näitä ei tarvitse asentaa erikseen, vaan ne voidaan ottaa suoraan käyttöön kirjoittamalla kirjaston nimi. Ulkopuolisten (engl. External) kirjastot ovat nimensä mukaisesti ulkopuolisia kirjastoja, jotka täytyvät ladata ennen niiden käyttöä. Ulkopuoliset kirjastot ladataan PIP:n eli Pythonin paketinhallintajärjestelmän kautta. Resurssitiedostolle (engl. Resource) määritetään yhden tai useamman resurssitiedoston polku, mistä resurssitiedosto löytyy. Resurssitiedoston etuna on se, että sieltä voidaan ladata testissä käytettäviä muuttujia (engl. Variables) ja avainsanoja (engl. Keywords).

```
*** Settings ***
Library          SeleniumLibrary
Library          BuiltIn
Library          Collections
Library          String
Resource         ../Resources/robot1resource.robot
```

KUVA 20. Testin asetukset ja lisätyt kirjastot.

Muuttujilla määritetään testin aikana käytettäviä muuttujia (KUVA 21). Robot Frameworkin muuttuja tyypit esitellään seuraavasti: yksittäiset muuttujat esitellään koodilla "`${muuttuja}`", listat koodilla "`@{listanimi}`" ja hakemistot koodilla "`&{nimi}`".

```
*** Variables ***
${browser}      chrome
${url}          https://robotframework.org
@{list}         robot    framework
&{dict}         python=3.7.7    rf=3.1.2
```

KUVA 21. Testin muuttujat.

Avainsanoilla (engl. Keywords) määritetään testin aikana suoritettavat avainsanat, sekä määritetään, mitä kukin avainsana suorittaa. Esimerkkinä Startup -avainsana hyödyntää SeleniumLibrary -kirjaston avainsanaa "Open Browser", mikä avaa koodin "`${browser}`" -muuttujassa olevan verkkoselaimen ja koodin "`${url}`" -muuttujassa asetetun verkkosivun (KUVA

21 ja KUVA 22). Lopputuloksena Startup -avainsana avaa määritetyn verkkosivun ja selaimen eli Google Chrome -selaimella Robot Frameworkin etusivun.

Avainsanoja voidaan hyödyntää resurssitiedoston kautta tai kirjoittamalla ne suoraan testiin (KUVA 22). Avainsanat suositellaan kirjoittamaan erilliseen resurssitiedostoon siinä tapauksessa, jos useammat testit voivat hyödyntää samoja avainsanoja.

```

*** Keywords ***
Startup
  Open browser    ${url}    ${browser}
  Maximize Browser Window

Avaa Verkkokauppa
  Wait Until Element Is Visible    xpath:/html/body/div/div[1]/section/nav[1]/li[10]/a    30s
  Click Element    xpath:/html/body/div/div[1]/section/nav[1]/li[10]/a

Valitse Muki
  Select Window    NEW    30s
  Wait Until Element Is Visible    xpath:/html/body/div/div/div/div[2]/div[4]/div/div[7]/a/div[1]    30s
  Click Element    xpath:/html/body/div/div/div/div[2]/div[4]/div/div[7]/a/div[1]

Lisää Muki Ostoskoriin
  Wait Until Element Is Visible    xpath:/html/body/div/div/div/div[2]/div[1]/div[3]/div/div[2]/div/button    30s
  Click Button    xpath:/html/body/div/div/div/div[2]/div[1]/div[3]/div/div[2]/div/button

```

KUVA 22. Testin avainsanat.

Testitapaukset (engl. Test Cases) kohdassa määritetään suoritettavat testit. Testitapaukset rakentuvat avainsanoista, jotka esiteltiin avainsanoissa (KUVA 23). Testitapauksille voidaan määrittää tunniste (engl. Tags), minkä avulla kyseinen testi voidaan yksilöidä ja suorittaa nimenomainen testi antamalla tunnisteparametri. Alustuksella (engl. Setup) tarkoitetaan annetun avainsanan suorittamista ennen kuin testiä aloitetaan suorittamaan. Purulla (engl. Teardown) suoritetaan annettu avainsana, kun testitapaus on suoritettu riippumatta siitä, onnistuiko testi vai ei.

```

*** Test Cases ***
TestiEsimerkki
  [Tags]    ESIM-1234
  [Setup]    Startup
  Avaa Verkkokauppa
  Valitse Muki
  Lisää Muki Ostoskoriin
  [Teardown]    Close All Browsers

ToinenEsimerkki
  [Tags]    ESIM-4321
  [Setup]    Startup
  Avaa Verkkokauppa
  Valitse Muki
  Lisää Muki Ostoskoriin
  [Teardown]    Close All Browsers

```

KUVA 23. Testin testitapaukset.

4.4 Suorittaminen ja raportit

Robot Frameworkin testit voidaan suorittaa esimerkiksi käyttöjärjestelmän komentokehoteen kautta, RIDE editorin käyttöliittymän kautta tai Visual Studio Coden terminaalien kautta. Testi voidaan suorittaa navigoimalla komentokehoitteella kansioon missä testi sijaitsee ja kirjoittamalla komentokehoitteeseen "robot testinimi.robot" eli tässä tapauksessa komento "robot TestiEsimerkki.robot". Testin suorittamisen jälkeen Robot Framework luo automaattisesti testiraportin, testilokin, output XML-tiedoston ja virhetilanteessa SeleniumLibrary -kirjasto tallentaa kuvankaappauksen tapahtumasta missä testi epäonnistui.

Komentokehoitteeseen voidaan määrittää lukuisia parametrejä testiä käynnistäessä. Testiraportit muodostuvat automaattisesti samaan kansioon mistä testi ajetaan. Lisäparametreillä testin käynnistäessä saadaan tulokset generoitua uuteen kansioon, jotta testien rakenne pysyy siistinä. Testin käynnistyessä "-d kansionnimi" -parametrillä raportit muodostuvat raportit kansioon komennolla "robot -d raportit TestiEsimerkki.robot" (KUVA 24). RIDE editoria käytettäessä testien raportit generoituvat automaattisesti uuteen kansioon eikä sille tarvitse määrittää parametrejä.

```
ToinenEsimerkki | PASS |
-----
Test | PASS |
2 critical tests, 2 passed, 0 failed
2 tests total, 2 passed, 0 failed
-----
Output: C:\Users\Toni\Automaatiotestit\raportit\output.xml
Log: C:\Users\Toni\Automaatiotestit\raportit\log.html
Report: C:\Users\Toni\Automaatiotestit\raportit\report.html
```

KUVA 24. Testin tulokset komentokehoteessa.

Testilokista nähdään suoritettut testit, testien tulokset, testien tiedot ja aikaleimat (KUVA 25). Testin tulokset ja avainsanat näkyvät vihreänä testin onnistuttua, mutta virhetapauksissa avainsana sekä testi muuttuvat punaiseksi.

Test Log

Generated
20200322 17:25:40 UTC+02:00
21 minutes 8 seconds ago

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		2	2	0	00:00:53	
All Tests		2	2	0	00:00:53	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
ESIM-1321		2	2	0	00:00:53	

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
Test		2	2	0	00:00:53	

Test Execution Log

SUITE Test

Full Name: Test
 Source: C:\Users\Ton\Automaatiotestit\test.robot
 Start / End / Elapsed: 20200322 17:24:47.631 / 20200322 17:25:40.340 / 00:00:52.709
 Status: 2 critical test, 2 passed, 0 failed
 2 test total, 2 passed, 0 failed

TEST TestiEsimerkki

Full Name: Test.TestiEsimerkki
 Tags: ESIM-1321
 Start / End / Elapsed: 20200322 17:24:47.780 / 20200322 17:25:15.101 / 00:00:27.321
 Status: **PASS** (critical)

- SETUP** Startup
- KEYWORD** Avaa Verkkokauppa
- KEYWORD** Valitse Muki
- KEYWORD** Lisää Muki Ostoskoriin
- TEARDOWN** SeleniumLibrary.Close All Browsers

KUVA 25. Lokitiedosto onnistuneesta testistä.

Onnistuneen testin testiraportista nähdään suoritettujen testien tilat, aikaleimat, suoritusajat testeille ja lokitiedosto (KUVA 26).

Test Report

Generated
20200322 17:25:40 UTC+02:00
34 minutes 29 seconds ago

Summary Information

Status: All tests passed
 Start Time: 20200322 17:24:47.631
 End Time: 20200322 17:25:40.340
 Elapsed Time: 00:00:52.709
 Log File: log.html

Test Statistics

Total Statistics		Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests		2	2	0	00:00:53	
All Tests		2	2	0	00:00:53	

Statistics by Tag		Total	Pass	Fail	Elapsed	Pass / Fail
ESIM-1321		2	2	0	00:00:53	

Statistics by Suite		Total	Pass	Fail	Elapsed	Pass / Fail
Test		2	2	0	00:00:53	

Test Details

Totals Tags Suites Search

Type: Critical Tests All Tests

KUVA 26. Raportti onnistuneesta testistä.

Epäonnistuneessa testissä testin tilana on "FAIL" eli testi on epäonnistunut (KUVA 27). Startup -avainsana on epäonnistunut, koska "{\$url}" -muuttujaan oli asetettu osoitteeksi virheelinen verkko-osoite. Virhetilanteessa SeleniumLibrary -kirjasto tallentaa kuvankaappauksen käytettävästä selaimesta.

```

- TEST TestiEsimerkki
  Full Name:      Test.TestiEsimerkki
  Tags:           ESIM-1321
  Start / End / Elapsed: 20200322 18:04:26.197 / 20200322 18:04:33.969 / 00:00:07.772
  Status:         FAIL (critical)
  Message:        Setup failed:
                  InvalidArgumentException: Message: invalid argument
                  (Session info: chrome=80.0.3987.149)

- SETUP Startup
  Start / End / Elapsed: 20200322 18:04:26.198 / 20200322 18:04:31.855 / 00:00:05.657
  - KEYWORD SeleniumLibrary.Open Browser ${url}, ${browser}
    Documentation: Opens a new browser instance to the optional url.
    Start / End / Elapsed: 20200322 18:04:26.198 / 20200322 18:04:31.855 / 00:00:05.657
    + KEYWORD SeleniumLibrary.Capture Page Screenshot
      18:04:26.198 INFO Opening browser 'chrome' to base url 'robotframework.org'.
      18:04:31.854 FAIL InvalidArgumentException: Message: invalid argument
                        (Session info: chrome=80.0.3987.149)
    + TEARDOWN SeleniumLibrary.Close All Browsers
  
```

KUVA 27. Lokitiedosto epäonnistuneesta testistä.

Virhetilanteessa testiraportti muuttuu punaiseksi, jos testi epäonnistuu (KUVA 28). Testiraportista nähdään mitkä testit ovat epäonnistuneet, mutta epäonnistumisen syy selviää vasta tarkasteltaessa lokitiedostoa.

Test Report Generated
20200322 18:04:41 UTC+02:00
11 minutes 55 seconds ago

Summary Information

Status: **2 critical tests failed**
 Start Time: 20200322 18:04:26.068
 End Time: 20200322 18:04:41.746
 Elapsed Time: 00:00:15.678
 Log File: [log.html](#)

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	2	0	2	00:00:16	<div style="width: 100%; height: 10px; background-color: red;"></div>
All Tests	2	0	2	00:00:16	<div style="width: 100%; height: 10px; background-color: red;"></div>

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
ESIM-1321	2	0	2	00:00:16	<div style="width: 100%; height: 10px; background-color: red;"></div>

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Test	2	0	2	00:00:16	<div style="width: 100%; height: 10px; background-color: red;"></div>

Test Details

Totals Tags Suites Search

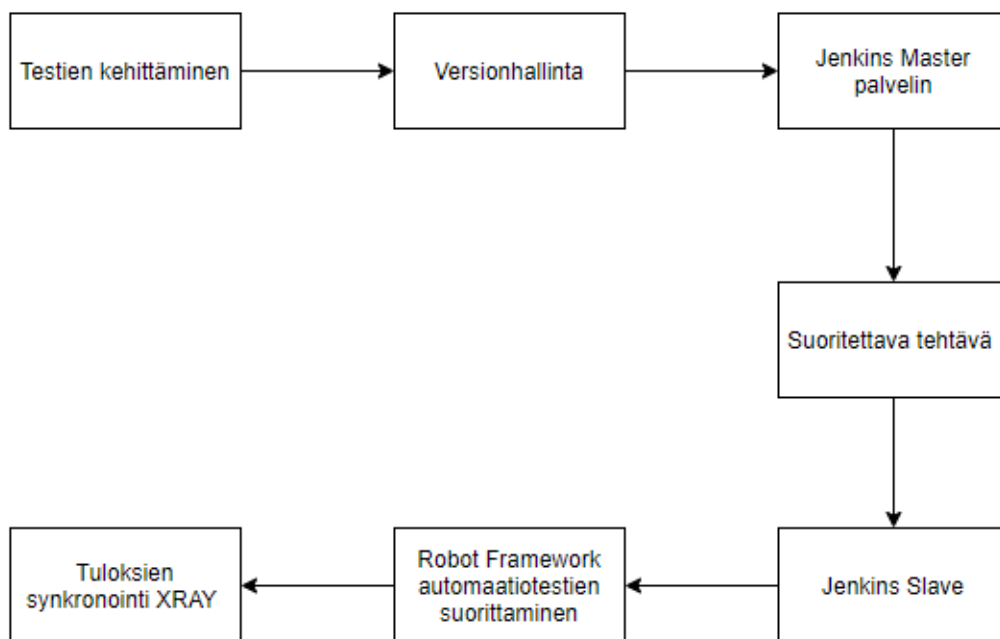
Type: Critical Tests All Tests

KUVA 28. Raportti epäonnistuneesta testistä.

4.5 Jenkins

Yrityksellä oli Jenkins Master palvelin käytössä jo entuudestaan, mutta Robot Framework:ia varten luotiin Windows Server 2016 palvelin, joka yhdistettiin Jenkins Master palvelimen slaveksi (KUVA 29). Tämän avulla saatiin rakennettua kokonaisuus siten, että Master palvelimelle rakennettiin uusi tehtävä (engl. Job), joka suoritettiin Jenkins slavella.

Tehtävälle voidaan antaa useita eri asetuksia/määrittämiä liittyen lähdekoodin hallintaan versionhallinnasta, koontiversion käynnistykseen, koontiversion ympäristöön sekä suoritukseen. Tehtävälle määrättiin, että Robot Framework testit suoritetaan "xxxx" nimisellä Jenkins slavella ja sille ennalta määritetyillä parametreilla. Robot Framework luo testien suorittamisen jälkeen XML-tiedoston, mikä synkronoituu Jenkinsin kautta XRAY:n testisuunnitel- malle. XRAY:n testisuunnitelman tila päivittyy automaattisesti sen mukaan, epäonnistuiko testit vai ei.



KUVA 29. Testin prosessikaavio.

5 TESTITAPAUKSET

Opinnäytetyön tavoitteena oli ottaa automaatiotestit Robot Frameworkilla käyttöön ja suunnitella automaatiotestejä. Testitapauksien rakentamisen tukemiseksi testitapauksille tehtiin testaussuunnitelma, missä jaettiin testit eri järjestelmien kesken omille testisuunnitelmille ja testiseteille. Oheisessa luvussa käydään läpi opinnäytetyössä yksi toteutetuista testaussuunnitelmista, missä regressiotestit automatisoidaan ja testit kohdennetaan yrityksen Web-pohjaiseen sovellukseen.

Opinnäytetyön aikana automaatiotestejä rakennettiin useita kymmeniä sekä useita testaussuunnitelmia, mutta tässä luvussa käydään vain yhtä näistä läpi.

5.1 Testaussuorite

Testaussuunnitelmaa rakennettaessa pohdittiin, mihin järjestelmiin ja millaisissa tilanteissa testejä kannattaa automatisoida. Testaussuunnitelmat suunniteltiin toteutettavaksi siten, että eri järjestelmille jaettiin omat regressiotestit ja muille testeille omat, jotta niiden suunnittelu ja toteutus on mahdollisimman helposti ylläpidettävää sekä tehokasta.

Testisuoritteelta nähdään, että regressiotestit järjestelmään "X" on lisätty listalle suoritettavaksi (KUVA 30). Suunnitelma perustuu uuden laboratoriotietojärjestelmän käyttöliittymään ja siihen liittyvien käyttöliittymien ja sen komponenttien testaamiseen. Regressiotestit tällä testaussuunnitelmalla perustuivat laboratorionäytteiden tilaamiseen, tulosten antamiseen, tilattujen tutkimusten seurantaan, näytteiden toimipisteen väliseen vaihtoon sekä tulosten tarkastamiseen.

Kuvasta 30 nähdään, että jokaiselle testille on generoitunut avain (engl. Key), joka lisätään automaatiotestien tunnisteeksi eli Tagiksi (KUVA 31). Testien suorittamisen jälkeen Robot Framework luo suoritetuista testeistä XML-raportti tiedoston, jonka Jenkins-palvelin lähettää XRAY:lle. Tulosraportti yhdistyy testisuoritteelle ja antaa testeille tuloksen riippuen siitä, mikä oli testien tulos. Testisuorite tunnistaa yksittäiset testit käytettyjen Tagien avulla.

Rank	Key	Summary	Test Type	#Req	#Def	Test Sets	Assignee	Status
1	LAB20-494	Regressio: [REDACTED] näytteenotto	Generic	0	0	LAB20-493	Toni Hynynen	TODO
2	LAB20-497	Regressio: [REDACTED] tutkimuksen tilaus	Generic	0	0	LAB20-493	Toni Hynynen	TODO
3	LAB20-498	Regressio: [REDACTED] tilatut tutkimukset	Generic	0	0	LAB20-493	Toni Hynynen	TODO
4	LAB20-499	Regressio: [REDACTED] vastaukset uusi näkymä	Generic	0	0	LAB20-493	Toni Hynynen	TODO
5	LAB20-500	Regressio: [REDACTED] toimipisteen vaihto	Generic	0	0	LAB20-493	Toni Hynynen	TODO

Showing 1 to 5 of 5 entries

First Previous 1 Next Last

KUVA 30. XRAY testisuorite järjestelmään "X" kohdistuvat regressiotestit.

5.2 Automaatiotesti

Oheisessa kappaleessa käydään opinnäytetyön aikana yhtä rakennettua testitapausta läpi, missä testitapaus merkitsee tilatun laboratorionäytteen otetuksi ja käy siihen liittyvän prosessin läpi, sekä antaa näytteelle asetetun tulokset. Testitapaus mikä käydään läpi ei liity regressio testisuoritteeseen, mutta se hyödyntää samaa järjestelmää (KUVA 30). Testitapah- tumassa osa avainsanoista on muutettu "Järjestelmään 1" tai "Järjestelmään 2", koska virall- lissa versiossa tiedot ovat virallisten järjestelmien nimillä (KUVA 31 ja KUVA 32).

```

*** Test Cases ***
Näyteprosessi: Näytteenotto ja tuloksien kirjaaminen
  [Tags]    LAB20-477
  [Setup]   Avaa selain
  Syötä tunnukset
  Vaihda toimipiste
  Valitse potilas
  Valitse näyte
  Tulosta tarra
  Kirjaudu ulos järjestelmästä

  Järjestelmään 1 siirtyminen
  Kirjautuminen sisään
  Järjestelmä 1 tuloksen haku
  Näytteen validointi
  Tarranumeron tallennus
  Kirjaudu ulos

  Järjestelmään 2 siirtyminen
  Kirjaudu sisään
  Hae tulos
  Aseta näytteelle tulokset

```

```

143 | Aseta näytteelle tulokset
144 | Wait Until Element Is Visible    xpath://html/body/form    30s
145 | Sleep    2s
146 | Unselect Frame
147 | Select Frame    xpath://html/body/div/div/div/div[3]/div/iframe
148 | Select Frame    xpath://html/frameset/frame[1]
149 | Sleep    2s
150 | Click Element    xpath://html/body/form/table/tbody/tr[2]/td[2]/a
151 | Wait Until Element Is Visible    xpath://html/body/form/table[2]/tbody/tr[2]/td[1]/input    30s
152 | Input Text    xpath://html/body/form/table[2]/tbody/tr[2]/td[3]/input    ${result}
153 | Click Element    xpath://html/body/form/div[1]/input[10]
154 | Wait Until Element Is Visible    xpath://html/body/form/table/tbody/tr[2]    30s
155 | Sleep    2s
156 | ${checkbox}    Run Keyword And Return Status    Checkbox Should Be Selected    xpath://html/body/form/table/tbody/tr[2]/td[10]/span[1]/input
157 | Sleep    2s
158 | Run Keyword If    "${checkbox}" == "${TRUE}"    Click Element    xpath://html/body/form/div[1]/div[1]/input[1]
159 | ... ELSE    Select Checkbox    xpath://html/body/form/table/tbody/tr[2]/td[10]/span[1]/input
160 | ... AND    Click Element    xpath://html/body/form/div[1]/div[1]/input[1]

```

KUVA 31-32. Näyteprosessi: Näytteenotto ja tuloksen kirjaaminen testitapaus, Näytteelle asetettava tuloksen avainsana.

6 TULOKSET JA POHDINTA

Opinnäytetyön tavoitteena oli ottaa käyttöön ja kehittää automaatiotestausta manuaalitestauksen rinnalle, yrityksen Web-sovelluksen käyttöliittymän regressiotestien testaamiseen, hyödyntäen Robot Frameworkia ja SeleniumLibrary -kirjastoa. Opinnäytetyölle asetetut tavoitteet toteutuivat ja testitapauksia rakennettiin noin 20 kappaletta, joista osa perustui regressiotestaukseen ja osa testasi useamman eri järjestelmän välistä yhteistoimintaa.

Opinnäytetyön aikana tuli muutamia ongelmia vastaan, mutta ne saatiin ratkaistua suhteellisen hyvin. Isoin ongelma opinnäytetyön aikana oli, että testejä oli suoritettu ja kehitetty toimimaan Google Chrome sekä Mozilla Firefox verkkoselaimille. Testit siirrettiin versionhallinnan kautta Windows Server 2016 -palvelimelle, jonka kautta Robot Framework -testit käynnistyivät, kun Jenkins Master käynnisti annetun tehtävän. Ongelmaksi tuli, että testit toimivat muilla selaimilla moitteettomasti, mutta eivät Internet Explorerilla. Testejä täytyi rakentaa uudelleen, jotta ne toimisivat myös Internet Explorerilla, mutta silti kaikkia haluttuja testejä ei saatu toimimaan.

Ennen opinnäytetyötä minulla ei ollut lainkaan kokemusta ohjelmistotestaamisesta, Robot Frameworkista, Pythonista, Jirasta tai Jenkinsistä. Opinnäytetyöni aikana kehityin kaikilla osa-alueilla, joista minulla ei ollut lainkaan aiemmin kokemusta ja työn aikana saadusta kokemuksesta on varmasti tulevaisuudessa hyötyä.

Yritys ja opinnäytteen kirjoittaja olivat tyytyväisiä saatuihin lopputuloksiin kyseisen opinnäytetyön osalta. Jatkokehityssuunnitelmia opinnäytetyössä rakennetulle testausautomaatiokokonaisuudelle on suunniteltu muun muassa uusien automaatiotestien kehittämistä ja Robot Framework:in hyödyntämistä ohjelmistorobotiikan (RPA) puolella.

7 LÄHDELUETTELO

- Atlassian. (ei pvm). *Atlassian Marketplace*. Haettu 15. Maaliskuu 2020 osoitteesta Atlassian Marketplace: <https://marketplace.atlassian.com/apps/1211769/xray-test-management-for-jira?hosting=cloud&tab=overview>
- Atlassian. (ei pvm). *Jira Software*. (Atlassian, Toimittaja) Haettu 22. Maaliskuu 2020 osoitteesta Jira Software: <https://www.atlassian.com/software/jira/guides/getting-started/overview#jira-software-hosting-options>
- Datica. (18. Huhtikuu 2019). *Datica*. (D. Levin, Toimittaja) Haettu 17. Huhtikuu 2020 osoitteesta Datica: <https://datica.com/blog/what-is-hl7/>
- Eficode. (20. Kesäkuu 2016). *Eficode*. Haettu 15. Maaliskuu 2020 osoitteesta Eficode: <https://www.eficode.com/blog/en/blog/robot-framework>
- Fimlab Laboratoriot Oy. (15. Huhtikuu 2020). *Fimlab Laboratoriot Oy*. Haettu 15. Huhtikuu 2020 osoitteesta Fimlab Laboratoriot Oy: www.fimlab.fi
- GeeksforGeeks. (ei pvm). *GeeksforGeeks*. Haettu 15. Maaliskuu 2020 osoitteesta GeeksforGeeks: <https://www.geeksforgeeks.org/sandwich-testing-software-testing/>
- Grinevich, A. (24. Kesäkuu 2019). *Medium*. Haettu 20. Maaliskuu 2020 osoitteesta Medium: <https://medium.com/@alexey.grinevich/scenarios-or-why-some-automation-projects-fail-c35ede8b40ff>
- Guru99. (ei pvm). *Guru99*. Haettu 16. Maaliskuu 2020 osoitteesta Guru99: <https://www.guru99.com/black-box-testing.html>
- Guru99. (ei pvm). *Guru99*. Haettu 16. Maaliskuu 2020 osoitteesta Guru99: <https://www.guru99.com/grey-box-testing.html>
- HL7 Finland. (ei pvm). *HL7 Finland*. Haettu 2020. Maaliskuu 2020 osoitteesta HL7 Finland: <http://www.hl7.fi/esitely/>
- HL7 International. (ei pvm). *HL7 International*. Haettu 22. Maaliskuu 2020 osoitteesta HL7 International: https://www.hl7.org/implement/standards/product_brief.cfm?product_id=185
- Kasurinen, J. P. (2013). *Ohjelmistotestauksen käsikirja*. Jyväskylä: Docendo. Haettu 14. Maaliskuu 2020
- Lyniate. (ei pvm). *Lyniate*. Haettu 21. Maaliskuu 2020 osoitteesta Lyniate: <https://www.lyniate.com/knowledge-hub/hl7-standard-versions/>
- Octopus Deploy. (ei pvm). *Octopus Deploy*. Haettu 17. Maaliskuu 2020 osoitteesta Octopus Deploy: <https://octopus.com>
- Robot Framework. (ei pvm). *Robot Framework*. Haettu 27. Maaliskuu 2020 osoitteesta Robot Framework: www.robotframework.org
- Robot Framework User Guide. (24. Toukokuu 2019). *Robot Framework User Guide*. Haettu 15. Maaliskuu 2020 osoitteesta Robot Framework User Guide: <https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#creating-test-data>
- Rongala, A. (24. Helmikuu 2015). *Invensis*. (A. Rongala, Toimittaja) Haettu 16. Helmikuu 2020 osoitteesta Invensis: <https://www.invensis.net/blog/it/white-box-software-testing-advantages-disadvantages/>
- Tieturi. (ei pvm). *Tieturi*. (T. Saranen, Tuottaja) Haettu 9. Huhtikuu 2020 osoitteesta Tieturi: <https://www.tieturi.fi/blogi/jira-ja-confluence-tyokalut-liiketoiminnan-onnistuneeseen-digitalisointiin/>
- University Of Jyväskylä. (1. Tammikuu 2019). *Smart Education*. Haettu 14. Maaliskuu 2020 osoitteesta Smart Education: <http://smarteducation.jyu.fi/projektit/systech/Periaatteet/suunnittelun-periaatteet/testaus/testauksen-tasot>

Vala Group. (15. Helmikuu 2017). *Vala Group*. Haettu 26. Huhtikuu 2020 osoitteesta Vala Group: <https://www.valagroup.com/fi/2017/02/vala-group-liittyi-robot-framework-foundationiin/>

Wikipedia. (27. Tammikuu 2020). *Wikipedia*. Haettu 23. Maaliskuu 2020 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Health_Level_7