jamk.fi

Web-pohjaisen sovelluksen hyväksymistestauksen automatisointi

Jukka Väyrynen

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), hyvinvointiteknologian tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Väyrynen, Jukka	Opinnäytetyö, AMK	huhtikuu, 2017
	Sivumäärä 39	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Web-pohjaisen sovelluksen hyväksymistestauksen automatisointi

Tutkinto-ohjelma
Hyvinvointiteknologia

Työn ohjaaja(t)
Ström Markku, Siistonen Matti

Toimeksiantaja(t)
Sensire Oy

Tiivistelmä

Hyväksymistestauksen tarkoituksena on varmistaa, että sovellus toimii niin kuin on määritetty. Onnistuneen hyväksymistestauksen jälkeen sovellus, tai sen uusi versio, voidaan julkaista asiakkaan käytettäväksi. Joensuulainen Sensire Oy toteuttaa asiakkaillensa ratkaisuja digitaaliseen laadunvalvontaan ja automatisoituun olosuhdeseurantaan langattomien sensoreiden ja mobiililaitteiden avulla. Asiakas voi tarkastella sensoreiden mittauksia ja mobiilikirjauksia yrityksen itse luoman web-sovelluksen kautta.

Kehittämistutkimuksen tavoitteena oli kehittää tapa web-sovelluksen hyväksymistestauksen automatisoimiselle. Tavoitteena oli perehtyä varsinkin Robot Frameworkiin, Seleniumiin ja Jenkinsiin. Työkaluilla tuli luoda ympäristö, jossa voi luoda automatisoituja testejä ja suorittaa niitä etäyhteyden avulla toisella tietokoneella. Tutkimuksen aineistona käytettiin valmista aineistoa, jota löytyi kirjallisuudesta ja internetistä. Aineisto jaettiin teemoihin, jotka olivat: ohjelmistotestaus, automaatiotestaus ja automaatiotestauksen työkalut.

Työkaluihin tutustumisen jälkeen luotiin ympäristö, jossa pystytään Jenkinsistä käsin suorittamaan Robot Framework –testi Selenium Gridin avulla toisella tietokoneella. Kun ympäristö oli valmis, aloitettiin hyväksymistestien tekeminen web-sovellukselle.

Tutkimuksen tuloksena saatiin luotua toimiva ympäristö hyväksymistestauksen automatisoimiselle. Automatisoidut testit ovat nopeita, tarkkoja ja erittäin helposti toistettavissa, eivätkä ne sorru huolimattomuusvirheisiin ihmisten lailla. Toisaalta tarkkuuden voidaan katsoa olevan myös ongelma, koska automatisoitu testi tarkistaa vain sen mitä sen on käskettykin tarkistaa. Eli sovelluksessa voi olla virheitä, vaikka testi onnistuisikin. Tästä syystä automatisointi ei poista kokonaan manuaalisen testaamisen tarvetta.

Avainsanat (asiasanat)
Robot Framework, Jenkins, Selenium, Selenium Grid, Selenium IDE, jatkuva integraatio, automaatiotestaus, hyväksymistestaus

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Väyrynen, Jukka	Bachelor's thesis	Huhtikuu, 2017
		Language of publication: Finnish
	39	Permission for web publication: x

Automating acceptance testing of web-based application

Degree programme
Wellness Technology

Supervisor(s)
Ström Markku, Siistonen Matti

Assigned by
Sensire Oy

Abstract

The purpose of acceptance testing is to make sure that an application works like it should work. After successful acceptance testing an application can be released to the use of the customer. Sensire Oy is a company that is located in Joensuu. Sensire implements solutions for digitalized quality assurance and for automated inhousecontrol. The solutions are based on wireless sensors or mobile devices. Sensire provides a web-application that the customer can use to view the measurements or the mobile records.

The aim of this development research was to create a method for how to automate acceptance testing. The goal was to especially explore Robot Framework, Selenium and Jenkins. The outcome should consist of an environment where automated tests can be executed remotely from another computer. The study used existing material from literature and the internet. The found material was divided into different themes, which were software testing, test automation and tools of test automation.

After the utilizations of the tools were recognized, tools were used to build an environment where an automated Robot Framework test can be executed from Jenkins in a remote computer with the connection provided by Selenium Grid. Creation of the acceptance testes was started after the functionality of the environment was proven.

As a result, a workable environment for acceptance testing was created. Automated tests are fast, accurate and repeatable. They are not liable to careless errors like a human is. The accuracy of the automated tests can be seen as a problem as well as a benefit because they only test what they have been told to. That means that there can be bugs in the application although the tests succeed. That is the main reason why test automation doesn't remove the need for manual testing entirely.

Keywords/tags (subjects)
Robot Framework, Jenkins, Selenium, Selenium Grid, Selenium IDE, continuous integration, testautomation, acceptance testing

Miscellaneous

Sisältö

Kuviot

1 Johdanto

1.1 Opinnäytetyön tausta ja tavoitteet

Ohjelmistotestaus on yksi osa ohjelmistokehitystä. Testaus on tärkeä tekijä ohjelmiston laadun varmistuksessa. Ohjelmistotuotannossa käy usein niin, että asetetut aikarajat ovat liian tiukkoja, minkä johdosta jostain työvaiheesta joudutaan tinkimään. Usein testaus on juuri se vaihe, jolle varatusta ajasta käytetään osa, ellei kaikki, muihin työvaiheisiin. Jotta testaus siis pystyttäisiin helpommin suorittamaan, sen tulisi olla mahdollisimman nopeaa ja vaivatonta. Apua tähän ongelmaan tuo koko ajan yleistyvä automatisoitu testaus, kuten käy ilmi Testing Trends in 2017 -nimisestä kyselytutkimuksesta (2017). Tämän työn toimeksiantaja, Sensire Oy, haluaakin apua oman web-sovelluksensa testaamiseen testausautomaatiosta.

Tällä hetkellä yrityksessä suoritetaan web-sovellukselle jonkin verran manuaalista hyväksymistestausta, käyttäen alustana Atlassian Zephyr -nimistä testausjärjestelmää. Järjestelmään on luotu testitapauksia jokaiselle sovelluksen sisältämälle moduulille. Testitapaukset sisältävät askeleita, joissa kerrotaan testaajalle vaihe kerrallaan, mitä hänen tulee tehdä ja mitä siitä tulisi seurata. Ennen kuin sovelluksesta julkaistaan asiakkaalle uusi versio, pyritään nämä manuaaliset testit suorittamaan julkaisuehdokkaalle. Testiä suorittaessaan testaaja vertaa jokaisen testiaskeleen kohdalla tapahtunutta asiaa oletettuun lopputulokseen ja raportoi, mikäli huomaa virheitä. Löydetyt virheet pyritään korjaamaan mahdollisimman pian, minkä jälkeen testit, joiden avulla virheet löytyivät, suoritetaan uudelleen. Tätä kiertoa jatketaan niin kauan, kunnes merkittäviä virheitä ei enää esiinny. Kun tämä tilanne on saavutettu, uusi versio voidaan julkaista.

Automatisoimalla osa versiopäivitysten ohessa suoritettavasta testauksesta saataisiin nopeutettua testausta ja vähennettyä manuaalisen testauksen tarvetta. Yrityksessä on tällä hetkellä jo teoriassa tiedossa, millä työkaluilla ja menetelmillä automatisointi on mahdollista, mutta käytännön tieto ja taito puuttuvat. Työn tavoitteena on opetella menetelmät ja tavat testauksen automatisointiin. Tämä opinnäytetyö on kehittämistutkimus, jonka avulla yritykselle luodaan tapa ja osaaminen testausautomaation toteuttamiselle. Työkalut, joihin toimeksiantaja haluaa työn keskittyvän, ovat Ro-

bot Framework, Selenium ja Jenkins. Työkaluilla tulee rakentaa ympäristö, jossa voidaan suorittaa erikseen testaamiseen varatulla tietokoneella automatisoituja testejä toiselta tietokoneelta käsin.

1.2 Sensire Oy

Sensire Oy on Joensuulainen vuonna 2007 perustettu yritys, joka on erikoistunut digitaaliseen laadunvalvontaan ja automatisoituun olosuhdeseurantaan. Yritys työllistää yhteensä 23 henkilöä toimipaikoillaan Joensuussa ja Helsingissä. Päätoimipaikka sijaitsee Joensuussa. (Sensire yrityksenä 2017.) Sensire Oy tarjoaa ratkaisuja muun muassa kuljetusten ja lämpötilojen seurantaan sekä omavalvontatehtävien mobiilikirjauksiin. Kaikissa ratkaisuissa on ideana langattomuus ja datan automaattinen siirtyminen pilveen. Jokaisen ratkaisun mukana asiakas saa käyttöönsä yrityksen itse suunnitteleman ja toteuttaman web-pohjaisen sovelluksen, josta mittauksia ja kirjauksia voidaan seurata ja hallita. Yrityksessä halutaan panostaa web-sovelluksen testaamiseen laadun ja hyvän käyttäjäkokemuksen takaamiseksi. Testausprosessin kehittäminen on alkutekijöissään, mutta testauksen tärkeys on tiedostettu ja siihen on varattu resursseja.

1.3 Työn aineisto ja rakenne

Työssä käytettiin saatavilla olevaa valmista aineistoa. Kirjallisuudesta löytyi aiheeseen liittyvää teoriaa ja taustatietoa. Internet-lähteitä hyödynnettiin mahdollisimman tuoreen tiedon saamiseksi testausautomaation työkaluista ja menetelmistä. Laadullinen aineisto jaoteltiin teemoihin, minkä jälkeen jokaista teemaa tarkasteltiin yksityiskohtaisemmin. Teemat ovat ohjelmistotestaus, automaatiotestaus ja automaatiotestauksen työkalut.

Teemoittelu loi pohjan myös työn rakenteelle. Työn taustan, tavoitteiden ja toimeksiantajan esittelevän johdanto-osion jälkeen alkava teoriaosuus esittelee yleisesti ohjelmistotestausta, automaatiotestausta ja automaatiotestauksen työkaluja. Sen jälkeen tutustutaan tarkemmin toimeksiantajan haluamiin työkaluihin. Tämän jälkeen työkaluilla rakennetaan testausympäristö ja opetellaan tekemään sekä suorit-

tamaan automatisoituja hyväksymistestejä. Työn lopussa tarkastellaan saavutettuja tuloksia ja suoritetaan pohdinta.

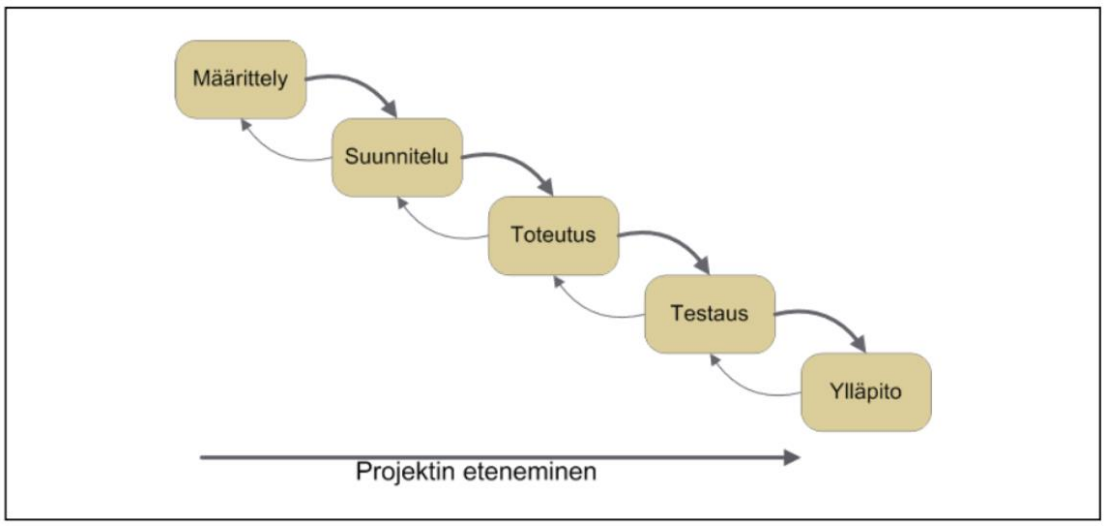
2 Ohjelmistotestaus

Ohjelmistotestaus on yksi kokonaisuus ohjelmistotuotannossa. Ohjelmistotestaus on työtä, jonka tarkoituksena on varmistaa, että toteutettu ohjelmisto toimii halutulla tavalla. Testaamalla ohjelmisto halutaan tarkistaa, että kaikki siihen luodut ominaisuudet toimivat juuri niin kuin on tarkoituskin. Ohjelmistotestauksen voidaan käytännössä katsoa olevan jatkuvaa vertailutyötä, jossa verrataan, että vastaako toteutus sitä, mitä oli tarkoitus tehdä. (Kasurinen 2013, 10.)

Haikala ja Mikkonen puolestaan määrittelevät ohjelmistotestauksen suunnitelmalliseksi virheiden etsimiseksi koko ohjelmaa tai sen osaa suorittamalla. He toteavat myös, että hyvin usein testauksesta kuitenkin puuttuu suunnitelmallisuus kokonaan ajan tai motivaation puutteesta johtuen, ja testausta suoritetaan vain kokeilemalla ohjelmaa umpimähkäisesti. Joskus, varsinkin testaajan ollessa itse ohjelmiston koodaaja, testauksen tavoitteena on vain osoittaa ohjelmiston toimivuus. (Haikala ja Mikkonen 2011, 205–206.)

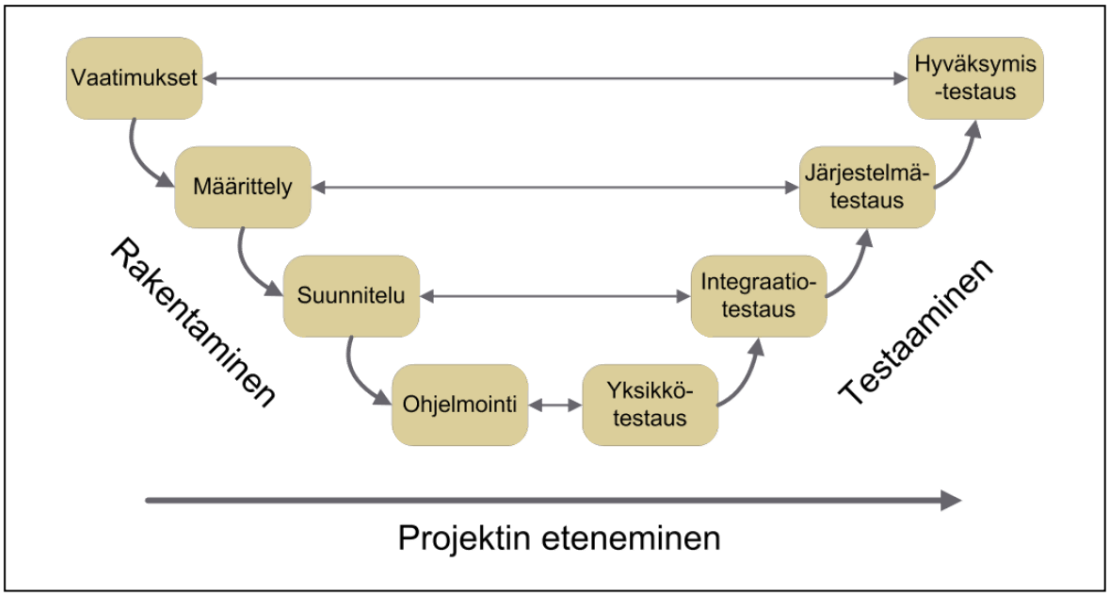
Haikala ja Mikkonen selventävät, että ohjelmistotestauksessa virheellä tarkoitetaan poikkeamaa siitä, mitä on määritetty. Tästä syystä johdonmukainen testaaminen ilman kunnon määrittelyä onkin mahdotonta, koska lopputuloksen oikeellisuutta ei voida todentaa. Yleinen käytäntö on, että määrittely jaetaan toiminnalliseen ja tekniseen määrittelyyn. Ohjelmistosta löydetyt virheet voivat olla vakavuudeltaan erilaisia. Vakavuus voi vaihdella ohjelmiston käytön estävästä virheestä pieneen kosmeettiseen puutteeseen. (Mts. 205–206.)

Vesiputousmalli (ks kuvio 1.) kuvastaa yleistä mielikuvaa siitä, miten testaus ajatellaan vain yhdeksi vaiheeksi portaittain laskevassa ohjelmistotuotannossa. Ajatus mallissa on se, että projekti etenee aina seuraavaan vaiheeseen edellisen loputtua, alkaen määrittelystä ja edeten ylläpitoon asti. Mallissa on mahdollista palata isojen ongelmien tapahtuessa edelliseen vaiheeseen, mutta tavoitteena on, ettei niin tarvitsisi tehdä. Malli on nykyään monellakin tapaa melko vanhanaikainen muun muassa sen takia, että monessa projektissa on mahdotonta kerätä vaatimukset ja luoda määrittelyt kokonaan etukäteen. Mallissa testausta tehdään myös vain yhdessä vaiheessa, jolloin kaikki suunnittelu- ja kehitystyö on toteutettu. (Kasurinen 2013, 13–14.)



Kuvio 1. Ohjelmistotuotannon vesiputousmalli (Kasurinen 2013, 13).

Vesiputousmallia parempi esimerkki ohjelmistotuotannon etenemisestä testauksen kannalta on V-malli (ks. kuvio 2), jossa testaus alkaa samaan aikaan kuin toteutuskin ja jatkuu projektin loppuun asti. Mallin ajattelutavassa testaus ei ole ainoastaan yksi vaihe projektissa, vaan jokaiselle kehitysvaiheelle on olemassa oma testauskategoria. Perusideana on, että ohjelmointityö testataan yksikkötestauksella, suunnitelmien toteutuminen integrointitestauksella, määrittelyjen toteutuminen järjestelmätestauksella ja asiakkaan yleiset vaatimukset hyväksymistestauksella. (Mts. 14.)



Kuvio 2. Ohjelmistotuotannon V-malli (Kasurinen 2013, 14).

2.1 V-mallin sisältämät testaustavat

Yksikkötestaus on ohjelmiston testaustavoista tavallisin ja yleisesti käytetty kaikissa ohjelmisto-organisaatioissa. Yksikkötestaus on testausta, jossa ohjelmoija tai kehittäjä itse tarkastelee välittömästi toteutuksen yhteydessä yksittäisen olion, funktion tai moduulin toimintaa. Tavoitteena yksikkötestauksessa on tarkistaa, että juuri tehty muutos tai toiminto olemassa olevaan järjestelmään toimii edes jollain tasolla. Yksikkötestauksessa tarkastetaan, että funktio toteuttaa toiminnalliset ominaisuutensa. Yksikkötestauksen suorittamisen ansiosta funktiossa havaittu vika voidaan korjata ennen kuin kyseinen funktio on edes käynyt osana laajempaa ohjelmaa. (Kasurinen 2013, 51–52.)

Integrointitestauksessa järjestelmän eri osia liitetään toisiinsa, jotta voidaan testata niiden toimivuus yhdessä. Integraatiotestauksessa toteutetut testitapaukset ovat kattavampia verrattuna yksikkötestaukseen, mutta eivät vielä koko järjestelmän kattavia testejä. Järjestelmän eri moduulien välillä tapahtuvan viestinvaihdon varmistaminen on esimerkki integrointitestauksen kohteesta. (Mts. 54.)

V-mallin kolmas testausvaihe on järjestelmätestaus. Järjestelmätestaukseen päästään siirtymään sitten, kun rakennettu komponentti on ensin yksikkötestattu ja sen jälkeen yhdistetty toimivaksi kokonaisuudeksi integrointitestauksessa. Nimensä mukaan tässä vaiheessa testataan koko järjestelmää. Järjestelmätestaus ei sinänsä tarkoita yhtä tiettyä testaustapaa, vaan se kattaa alleen kaikki tavat testaukselle, jonka kohteena on kokonainen järjestelmä. (Mts. 56–57.)

V-mallin neljäs ja viimeinen vaihe testaukselle on hyväksymistestaus. Hyväksymistestauksen ykköstavoite on osoittaa, että sovellus kykenee täyttämään vaatimusmäärittelyssä sille asetetut vaatimukset. Hyväksymistestaus voi tarkoittaa virallista työvaihetta, jossa itse asiakas tarkastaa sovelluksen oikeassa toimintaympäristössä. (Mts. 57.)

Hyväksymistestaus voidaan myös suorittaa kehittäjän toimesta, esimerkiksi tapauksissa, joissa sovellus on oma tuote asiakkaan tilaaman tuotteen sijasta. Vingrysin (2010) mukaan hyväksymistestauksen automatisointi on yleistä, sillä se helpottaa iterointia ja regressiotestausta. Kasurinen (2013, 68) kertoo, ettei regressiotestauk-

sella tarkoiteta mitään tiettyä testausmuotoa, vaan se on yleistermi uudelleen testaamiselle.

Lewis mainitsee, että hyväksymistestaukseen käytetään musta laatikko -tekniikkaa. Lewis täsmentää, että musta laatikko -tekniikka tarkoittaa testausta, jossa sovellukselle annetaan erilaisia syötteitä ja katsotaan, mitä sovellus tekee, välittämättä siitä mitä ohjelman sisällä tapahtuu. Autoakin voidaan ajaa ilman, että tiedetään, miten se oikeastaan toimii – tämä Lewiksen mainitsema ajatus kuvastaa menetelmää hyvin. Lewis tähdentää, että menetelmässä testaaja keskittyy testaamaan sovelluksen toiminnallisuutta määrityksiä vasten. (Lewis 2004, 94.)

3 Automaatiotestaus

Automaatiotestaus tarkoittaa sitä, että sovelluksen manuaalinen testi automatisoidaan niin, ettei sen suorittamiseen tarvita ihmisen läsnäoloa. Automaatiotestaus on testauksen muoto, jossa ihminen kirjoittaa testiskriptin itse, ja sen jälkeen käyttää sopivia ohjelmia skriptin suorittamiseen testattavalle sovellukselle. Automaatiotestaus siis käyttää automaatiotestauksen työkaluja testiskriptien tekemiseen ja suorittamiseen. (What is automation testing? n.d.)

Kasurisen mukaan ajatuksena testausautomaatiolla on vähentää manuaalisen työn tarvetta, nopeuttaa testausta ja parantaa tuotteen laatua. Automatisoimalla testit, joita suoritetaan useasti, on mahdollista vapauttaa testaajilta aikaa muihin tehtäviin. Automaatiotestaus soveltuu hyvin varmistamaan, että aiemmin toimineet osat eivät ole lopettaneet toimimasta uusien päivitysten yhteydessä. (Kasurinen 2013, 77–78.)

3.1 Automaatiotestauksen hyödyt

Automaatiotestauksen hyötynä on testauksen tarkkuus, toistettavuus, nopeus ja kustannustehokkuus. Koska automatisoitu testi on tietokoneohjelma, se ei ihmisen lailla sorru huolimattomuusvirheisiin, se on toistettavissa jatkuvasti ja se suoriutuu testistä nopeammin kuin ihminen. Tarkinkin testaaja sortuu jossain vaiheessa virheisiin, mutta automatisoitu testi suoriutuu sen sijaan testistä jokaisella kerralla täysin samalla tavalla. Automaatiotestaukseen ei tarvitse varata henkilöresursseja läheskään niin paljon kuin manuaaliseen testaamiseen. Kun testaus nopeutuu, myös virheiden löytyminen ja sitä myöten niihin puuttuminen nopeutuu. (Berglund 2015.)

Automaattisten testien kustannustehokkuus saadaan aikaan toistojen avulla. Automatisoidun testin tekeminen yhdelle testitapaukselle on hitaampaa ja kalliimpaa kuin yhden manuaalisen testin ajaminen, mutta testien uudelleen ajaminen on käytännössä täysin ilmaista. Tästä syystä testit, jotka voidaan toistaa useasti muuttumattomina, tai hyvin vähäisillä muutoksilla ovat niitä, jotka kannattaa automatisoida. (Kasurinen 2013, 79.)

Testien automatisoinnilla on myös mahdollista saada aikaan tilanteita, joita manuaalisella testauksella on mahdotonta saada aikaan. Automatisoimalla pystytään luo-

maan esimerkiksi tilanteita, joissa kymmeniä, satoja tai jopa tuhansia käyttäjiä käyttää sovellusta yhtä aikaa. (Why automated testing n.d.)

Kun automatisoitujen testien ajaminen ei vaadi ihmisen läsnäoloa, on mahdollista esimerkiksi jättää tietokone suorittamaan tarkistuksia yöksi, jolloin kehittäjät voivat työpäivän aluksi käydä läpi tarkastuksen tulokset ja aloittaa korjaamaan mahdollisesti löydettyjä virheitä. Vaikka automaatiotestauksen rakentaminen vie aikaa ja resursseja, ovat sen ylläpito ja käyttäminen edullisempaa ja helpompaa kuin toistuvasti suoritettavan käsin testauksen. Resursseja ei välttämättä tarvitse varata työkaluihin, koska monet niistä ovat täysin ilmaisia. (Kasurinen 2013, 76–77.)

3.2 Automaatiotestauksen ongelmat

Automatisoitujen testien luominen ei ole helppoa ja alkuun pääseminen vie aikaa – varsinkin jos menetelmät eivät ole testaajalle ennestään tuttuja. Ghahrai huomauttaa, että automatisoitujen testien tarkkuus voidaan katsoa myös huonoksi puoleksi, koska testit tarkistavat testattavasta sovelluksesta vain mitä niiden on käskettykin tarkistaa. Tällöin sovelluksesta jää väkisinkin jotain tarkistamatta, mistä johtuen sovellus voi sisältää virheitä, vaikka sille suoritettu testi onnistuisikin. (Ghahrai 2015.)

Automatisoidusta testistä on haastavaa saada riittävän vaihteleva. Usein testattavasta sovelluksesta löytyy eniten virheitä ”vahingossa”, kun suoritetaan kokeilevaa testausta (exploratory testing). Tämä johtuu siitä, että kun sovellusta testataan vapaasti tutkimalla, sitä tarkastellaan monella eri tavalla. Automatisoitu testi suorittaa testin aina samalla tavalla, niin kuin se on kirjoitettu. Automatisoituunkin testiin on kuitenkin mahdollista saada vaihtelevuutta parametrien avulla, mutta vain vähän verraten ihmisen suorittamaan kokeilevaan testaukseen. (Mt.)

Ghahrai kertoo myös, että automatisoitu testi voi epäonnistua monesta syystä aiheuttaen turhia hälytyksiä. Testi voi kaatua esimerkiksi silloin, jos testattava palvelu on alhaalla tai verkkoyhteydessä on hetkellisiä katkoksia. Myös hyvinkin pienet muutokset käyttöliittymässä voivat aiheuttaa testin epäonnistumisen. Testien ylläpitoon kuluu aikaa ja työpanosta varsinkin silloin, jos testejä on paljon. (Mt.)

Pettichord pitääkin juuri testien tarvetta ylläpidolle testausautomaation suurimpana haasteena. Testit täytyy päivittää vastaamaan uusia päivityksiä, jotta ne eivät kaadu

uusien ominaisuuksien huomiotta jättämisen vuoksi virheiden löytämisen sijasta. Pettichord huomauttaa myös, että testausautomaation ylläpidossa on huomioitava ihmisten vaihtuminen. Osaavan ihmisen lähtiessä täytyy varmistaa, että testien ylläpitämiseen löytyy jatkaja. (Pettichord 2001.)

4 Automaatiotestauksen työkalut

4.1 Automaatiotestauksen viitekehykset

Viitekehys on yhdistelmä erilaisia protokollia, sääntöjä, ohjenuoria ja koodausstandardeja, jotka sisältyvät yhdeksi kokonaisuudeksi. Viitekehys toimii ikään kuin rakennustelineenä ympäristölle, jossa voidaan luoda automatisoituja testiskriptejä. Viitekehykset tarjoavat yleensä hyötyjä testiskriptien kehittämiseen, suorittamiseen ja raportointiin. (Most popular test automation frameworks 2016.)

Automaatiotestauksen viitekehyksiä on olemassa tyypiltään erilaisia. Viitekehykset voidaan jakaa karkeasti kuuteen eri kategoriaan: moduulipohjaisiin (module based), kirjastorakenteisiin (library architecture), dataohjattuihin (data driven), avainsanaohjattuihin (keyword driven), hybridisiin (hybrid) ja käyttäytymisohjattuihin (behavior driven). (Mt.)

Artikkelissa kerrotaan, että moduulipohjaisen viitekehyksen idea on jakaa testattava sovellus loogisiin ja erillisiin moduuleihin, jonka jälkeen jokaiselle moduulille luodaan oma, itsenäinen testiskripti. Kirjastorakenteinen viitekehys perustuu pohjimmiltaan moduulipohjaiseen, mutta eroaa siten, että sovellusta ei jaeta erillisiin moduuleihin vaan siitä erotellaan yleisiä toimintoja, jotka toimivat joka puolella sovellusta. Näistä toiminnoista kasataan sitten kirjastoja, joita voidaan kutsua testiskripteissä. (Mt.)

Dataohjattu viitekehys on artikkelin mukaan kehitetty siksi, että joskus samaa sovellusta on tarve testata useaan kertaan, käyttäen jokaisella kerralla hieman erilaista testidataa. Dataohjattu viitekehys mahdollistaa testidatan ja testiskriptin erottamisen toisistaan. Testidata on mahdollista säilöä erilliseen tietokantaan, kuten Excel-tiedostoon. (Mt.) Dataohjatussa viitekehyksessä testiskripti toimii ikään kuin testidatan toimittajana sovellukselle (Kelly 2003). Avainsanaohjattu viitekehys on dataohjatun viitekehyksen laajennos, joka ei ainoastaan erottele testidataa erilliseen tiedostoon, vaan pitää myös osan testiskriptin koodista ulkoisessa tiedostossa. Tämä ulkoinen koodi sisältää avainsanoja, jotka ovat sovellukselle suoritettavia itsestään ohjautuvia toimintoja. Avainsanat ja testidata pidetään tiedostoissa taulukkomaisessa rakenteessa. (Most popular test automation frameworks 2016.)

Käyttäytymisohjattu viitekehys mahdollistaa toimintojen hyväksymisen automatisoinnin helposti luettavassa ja ymmärrettävässä muodossa. Tällainen viitekehys ei välttämättä vaadi käyttäjältään koodauksen tuntemusta. (Mt.)

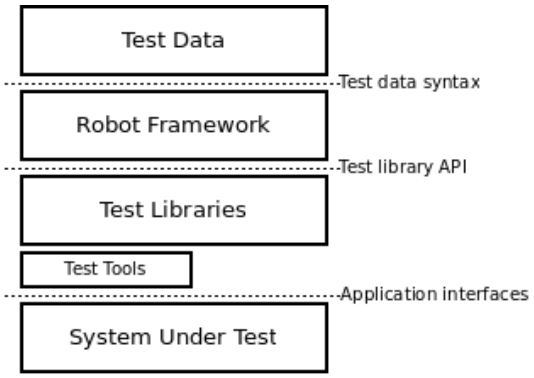
Hybrid-viitekehys taas on nimensä mukaisesti yhdistelmä muita viitekehyksiä. Tällaisen viitekehyksen ideana on yhdistää yllä mainittujen kehysten vahvuudet ja vähentää sitä kautta heikkouksia. (Kelly 2003.)

4.1.1 Robot Framework

Robot Framework kuuluu avainsanaohjattuihin viitekehyksiin, joka on suunniteltu hyväksymistestausta ja hyväksymisvetoista kehitystä varten. Robot Frameworkin testiskriptien syntaksi on taulukkomainen ja avainsanoihin perustuva, minkä ansiosta testit ovat selkeälukuisia ja helppoja tehdä. Robot Framework sisältää valmiiksi useita avainsanoja sisältäviä kirjastoja, joita on myös mahdollista lisätä jälkikäteen. Yhdistelemällä valmiita avainsanoja, käyttäjä voi luoda myös omia avainsanoja. Viitekehys luo suoritetuista testeistä selkeät html-muotoiset raportit, joista näkee hyvin tarkkaan, mihin kohtaan testi on mahdollisesti kaatunut ja miksi. Robot Framework on toteutettu Python-ohjelmointikielellä, mutta se tukee myös Jython- ja Ironpython-ohjelmointikieliä. (Robot Framework n.d.)

RIDE (Robot Intgrated Drive Electronics) on varta vasten Robot Frameworkille tehty testiskriptien ohjelmointiympäristö. RIDE-editorissa on ominaisuuksia, jotka helpottavat testien kirjoittamista ja hallintaa. Ohjelman avulla voi muun muassa nähdä jokaisen avainsanan ohjeistuksen, hakea avainsanoja ja luoda omia avainsanoja. (RIDE – How To 2012.) RIDE:n lisäksi testiskriptejä voi luoda lukuisilla eri sovelluksilla. Yksinkertaisimmillaan testien luominen onnistuu aivan tavallisella tekstieditorilla.

Valmiit testit voidaan suorittaa suoraan komentoriviltä tai RIDE-editorista käsin. Kun testi käynnistetään, ensimmäisenä viitekehys jäsentää testidatan. Seuraavaksi se käyttää avainsanoja hyväkseen kommunikoidakseen testattavan sovelluksen kanssa. Avainsanakirjastot voivat kommunikoida testattavan sovelluksen kanssa suoraan tai jonkin toisen työkalun avulla. Kuvio 3 havainnollistaa Robot Frameworkin toimintaperiaatetta. (Robot Framework n.d.)



Kuvio 3. Robot Frameworkin toimintaperiaate (Robot Framework n.d.).

Robot Framework sai alkunsa vuonna 2005 Pekka Klärck nimisen henkilön Pro Gradu -työn myötä. Samana vuonna Nokia Networks kehitti viitekehyksestä ensimmäisen version, ja versio 2.0 julkaistiin kaikille avoimena vuonna 2008. (Wikipedia, Robot Framework 2017.) Robot Frameworkillä on kattava käyttäjäkunta - usealla eri foorumilla voi esittää kysymyksiä ja keskustella yleisesti viitekehyksestä. Eri puolilla maailmaa on myös olemassa yrityksiä, jotka tarjoavat tukea, konsultointia ja koulutuksia viitekehykseen liittyen. Nokia, Metso ja Kone ovat esimerkkejä yrityksistä, jotka käyttävät Robot Framework -viitekehystä apuna ohjelmistotestauksessaan. (Robot Framework n.d.)

4.1.2 Muut viitekehykset

Testiautomaatioon kehitettyjä viitekehyksiä on olemassa todella paljon. Jokaisessa kategoriassa löytyy useita vaihtoehtoja, joista valita. Viitekehyksiä on olemassa sekä ilmaisia että maksullisia. Maksullisistakin viitekehyksistä on yleensä mahdollista saada ilmainen versio, mutta ilmaisversiot ovat toiminnaltaan rajoitettuja tai niiden käytölle on asetettu aikaraja. Viitekehyksiä on olemassa erilaisille testausmenetelmille kuten hyväksymistestaus, käyttäytymisohjattu testaus tai käyttöliittymätestaus. Myös testiskriptien ohjelmointikieli ja ohjelmoinnin määrä vaihtelevat riippuen siitä, mikä viitekehys on kyseessä. Muita viitekehysvaihtoehtoja tutkimalla ei löytänyt syytä olla käyttämättä Robot Frameworkiä. Robot Framework sopii työn tarpeisiin, koska avainsanojen ansiosta sillä on mahdollista luoda automatisoituja testejä ilman ohjelmointikielien tuntemusta. Viitekehys on todella moniulotteinen, sillä lisättävien

avainsanakirjastojen ansiosta sillä on mahdollista automatisoida web-sovelluksien lisäksi muun muassa Android- ja iOS-sovelluksia (Robot Framework n.d.).

4.2 Selenium

Selenium on sarja ohjelmistotyökaluja web-selainten automatisointiin. Jokaisella työkalulla on oma roolinsa testiautomaation tukemisessa. Seleniumia on mahdollista käyttää monella eri selaimella ja ympäristöllä sekä sen hallitseminen on mahdollista monella eri ohjelmointikielellä ja viitekehyksellä. (Selenium Introduction 2017.)

4.2.1 Selenium Grid

Selenium Grid mahdollistaa automatisoitujen testien suorittamisen useammalla tietokoneella ja selaimella yhtäaikaisesti. Sovelluksella on mahdollista määrittää millä tietokoneella, millä selaimella ja millä selaimen versiolla testi suoritetaan. Sovelluksesta on hyötyä silloin, kun ajettavia testejä on paljon, koska testejä voidaan ajaa yhtäaikaisesti useammalla eri koneella. (Selenium Grid 2017.)

Grid muodostuu yhdestä hubista ja yhdestä tai useammasta nodesta. Kummankin asentamiseen tarvitaan Selenium server jar -tiedosto. Hub käynnistetään tietokoneelle, jossa testit sijaitsevat. Hub käynnistetään komentoriviltä komennolla *java -jar selenium-server-standalone-<versio>.jar -role hub.* Hub käynnistyy käyttäen oletuksena porttia 4444. Portti on myös mahdollista määrittää itse lisäämällä komentoon parametri *-port <numero>.*

Testit suorittavat tietokoneet rekisteröidään nodeiksi hubiin. Node yhdistetään hubiin komennolla *java -jar selenium-server-standalone-<versio>.jar -role node -hub http://<hub-koneen IP-osoite>:4444/grid/register.* Noden oletusportti on 5555, jonka voi myös määrittää itse komennolla *-port <numero>.* Oletuksena hubilla on käytettävissä node-koneella viisi Chrome- ja Firefox-selainta sekä yksi Internet Explorer -selain. Myös yhtäaikaisten testien määrä on oletuksena viisi. Käyttäjä voi kuitenkin itse määrittää nämä asetukset sekä käytettävän selainversion ja käyttöjärjestelmän noden käynnistämisen yhteydessä parametrien avulla. Esimerkiksi, jos halutaan rekisteröidä Linux-node, jossa on käytettävissä kolme 3.6 versioista Firefox-selainta ja samanaikaisten testien määrän halutaan olevan kaksi, annetaan komento *java -jar*

selenium-server-standalone<versio>.jar -role node -hub http://<hub-koneen IP-osoite>:4444/grid/register -browser browserName=firefox,version=3.6,maxInstances=3,platform=LINUX -maxSession 2. (Grid2 2016.)

Kun Grid on pystyssä, yhteys ja käytettävissä olevat selaimet voidaan tarkistaa menemällä selaimella osoitteseen <http://hub-koneen IP-osoite:4444/grid/console>. (How To Use Selenium Grid n.d.) Noden yhteyden hubiin saa purettua painamalla Ctrl- ja c-painiketta yhtäaikaisesti komentorivillä, jossa node on rekisteröity. Hubin saa myös sammutettua samaisella komennolla.

4.2.2 Selenium IDE

Selenium IDE on Firefox-selaimen liitännäinen selaimen kanssakäymisen nauhoittamiseen ja toistamiseen. IDE:llä voidaan nauhoittaa käyttäjän toiminta sen tapahtuessa valmiiksi testiskriptiksi. Nauhoituksen jälkeen skripti on toistettavissa testikäyttöön. (Selenium IDE 2017.) Selenium IDE:n kaltaiset nauhoitus- ja toistotyökalut ovat helpoin lähestymistapa testien automatisointiin. Näiden sovellusten heikkoutena on kuitenkin se, ettei niillä monestikaan saada aikaan kestäviä ja helposti huollettavia testejä. (Test Automation Frameworks n.d.)

4.2.3 Selenium2Library

Selenium2Library on Robot Frameworkille tehty avainsanakirjasto web-sovellusten testaamiseen. Kirjasto on SeleniumLibrary-kirjaston haarauma. Selenium2Library käyttää toimiakseen Selenium 2 -kirjastoja. Kirjasto suorittaa testit oikeiden selainten ilmentymissä, tukien kaikkia yleisimpiä selaimia. Selenium2Libary-kirjaston avainsanojen avulla testit pystyvät vuorovaikuttamaan muun muassa evästeiden, web-elementtien, selaimen ikkunoiden ja sivujen kanssa. (Selenium2Library 2016.)

4.2.4 Testien hajauttaminen

Selenium Gridin lisäksi on olemassa muitakin tapoja ja sovelluksia testien jakamiseen. On olemassa sovelluksia, jotka mahdollistavat testien suorittamisen virtuaalisilla koneilla, samaan tietoverkkoon kuuluvilla tietokoneilla tai pilvipalveluissa. Monet näistä sovelluksista ovat kuitenkin maksullisia. Koska Selenium Grid on ilmainen ja se mahdollistaa testien jakamisen helposti ja tehokkaasti, ei ole syytä olla käyttämättä sitä.

4.3 Jatkuvan integraation palvelimet

Jatkuvan integraation (Continuous Integration, CI) palvelimet ovat sovelluksia, jotka on luotu jatkuvan integraation periaatetta noudattavaa ohjelmistokehitystä varten, kertoo Sommerville. Sommervillen mukaan jatkuvan integraation ideana on, että sovelluksesta luodaan jokaisen pienen lähdekoodiin tehdyn muutoksen jälkeen uusi versio. Kun uusi versio on luotu, automatisoidut testit käynnistyvät ja tarkistavat toimiiko kaikki niin kuin pitää. Toimiakseen jatkuva integraatio tarvitsee automatisoidut testit, koska lähdekoodin muutoksia voi tulla päivittäin useita. Jatkuvan integraation palvelimet ovat kehitetty hoitamaan tätä prosessia. Niiden avulla on mahdollista automatisoida uuden version luomiseen ja testaukseen tarvittavat toimenpiteet. (Sommerville 2016, 742–743.)

Toimeksiantajallani on käytössä Jenkins, joka on jatkuvan integraation palvelin, mutta varsinaisesti jatkuvan integraation prosessi ei ole vielä käytössä. Vaikka prosessi ei ole käytössä, palvelimet kuuluvat osaksi työtäni, koska ne tarjoavat paljon ominaisuuksia automatisoitujen testien suorittamiseen. Jatkuvan integraation palvelimen avulla on mahdollista muun muassa ajastaa testien käynnistymistä, ketjuttaa testejä toisiinsa ja saada hälytyksiä kaatuneista testeistä.

4.3.1 Jenkins

Jenkins on omavarainen ja avoimen lähdekoodin omaava jatkuvan integraation sovellus, jonka käyttäminen on täysin ilmaista. Sovelluksen avulla on mahdollista automatisoida esimerkiksi ohjelmiston kääntämiseen, käyttöönottoon ja testaamiseen liittyviä tehtäviä. (Jenkins documentation n.d.)

Jenkins sai alkunsa vuonna 2004 Koshuke Kawaguchin toimesta. Jenkinsillä on aktiivinen käyttäjäyhteisö, minkä avulla sovellus on kasvanut hyvin. Jenkinsin käyttö perustuu liitännäisiin, joita on olemassa tällä hetkellä yli 1200. Näiden liitännäisten avulla Jenkins voidaan liittää melkein kaikkiin olemassa oleviin teknologioihin. Jenkinsiin on olemassa liitännäiset muun muassa Robot Frameworkille ja Selenium Gridille. (About Jenkins n.d.)

Jenkins on yksi käytetyimmistä ja pidetyimmistä jatkuvan integraation sovelluksista. Joulukuussa 2016 Jenkinsillä oli yli 133 000 asennusta ja arviolta yli miljoona käyttä-

jää eri puolella maailmaa (About Jenkins n.d.). Sovellukselle on myönnetty useita palkintoja, joista mainittakoon voitto avoimen lähdekoodin sovelluksissa DevOps Dozen -kilpailussa vuonna 2016 sekä voitto jatkuvan integraation sovelluksissa vuoden 2014 Geek Choice Awardseissa (Awards 2017).

4.3.2 Muut jatkuvan integraation palvelimet

Jatkuvan integraation palvelimista on tarjontaa laajalti. Joukkoon mahtuu täysin ilmaisia sekä maksullisia sovelluksia. Joistakin maksullisista sovelluksista voi olla saatavilla ominaisuuksiltaan rajoitettu versio ilmaiseksi. Palvelimissa on eroja muun muassa alustoissa, käännöstyökaluissa, liitännäismahdollisuuksissa ja ilmoitustyypeissä. Vaihtoehtoihin tutustuttaessa Jenkinsiä kukistavaa sovellusta ei löytynyt. Varmasti muutama muukin sovellus olisi toiminut oikein hyvin, mutta Jenkinsin maksuttomuus, liitännäisten ja käyttäjien määrä sekä se, että se oli jo toimeksiantajalla käytössä, puolsivat Jenkinsiä. Liitännäisten avulla Jenkins on mahdollista yhdistää myös yrityksen JIRA-tehtävienhallintajärjestelmän ja BitBucket-versionhallintajärjestelmän kanssa.

5 Työkalujen asennus

Tässä vaiheessa työtä oli selvää, mitä työkaluja käytettäisiin testausympäristön luomiseen. Oli siis aika asentaa kyseiset työkalut ja muodostaa niistä toimiva kokonaisuus.

5.1 Robot Frameworkin asennus

Robot Frameworkin asentaminen vaatii, että tietokoneelle on asennettu joko Python-, Jython- tai IronPython-ohjelmointikieli. Asennettavaksi valikoitui Python, koska Robot Framework on toteutettu sillä ja sen käyttöä suositellaan käyttöohjeessa. (Robot Framework User Guide 2017.)

Pythonista asennettiin ensiksi uusin versio (3.6.1), mutta kun huomattiin, ettei RIDE tue uusinta versiota Pythonista, asennettiin Pythonista versio 2.7.13. Pythonin asennusta puolsi myös se, ettei RIDE tue muita ohjelmointikieliä (Installation Indroduction 2017). Pythonin asentamisen jälkeen, sen asennuskansion sijainti lisättiin Windowsin ympäristömuuttujien Polku-järjestelmämuuttujaan, jotta ohjelmaa on mahdollista käyttää Windowsin komentoriviltä. (Robot Framework User Guide 2017.)

Kun Python oli asennettu, Robot Framework oli mahdollista asentaa käyttäen Pythonin standardia pip-pakettienhallintamoduulia. Asennus tapahtui yksinkertaisesti syöttämällä Windowsin komentoriville käsky *pip install robotframework* (ks. kuvio 4). (Mt.)

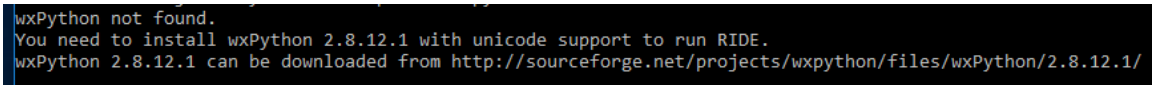
Kuvio 4. Komento, jolla asennetaan Robot Framework.

Robot Frameworkin sisältämät avainsanakirjastot asentuivat viitekehyksen asennuksen yhteydessä, mutta Selenium2Library täytyi asentaa erikseen. Asennus tapahtui ajamalla komento *pip install robotframework-selenium2library* (ks. kuvio 5). (Selenium2Library Installation 2017.)

Kuvio 5. Komento, jolla asennetaan Selenium2Library-avainsankirjasto.

Seuraavaksi oli aika asentaa testiskriptien muokkausohjelma RIDE. Myös sen asentaminen tapahtui käyttämällä Pythonin pip-moduulia, antamalla komentoriville käsky *pip install robotframework-ride* (ks. kuvio 6). Kun ohjelman asennus oli suoritettu, myös sen asennuskansion sijainti lisättiin Windowsin ympäristömuuttujien Polku-järjestelmämuuttujaan, jotta ohjelman käynnistäminen onnistuisi pelkästään suorittamalla komento *ride.py* komentoriviltä. Tästä huolimatta RIDE ei kuitenkaan käynnistynyt ensimmäisellä yrityksellä. Syynä tähän oli se, että RIDE:n käynnistyminen ei ole mahdollista, jos tietokoneelle ei ole asennettu wxPythonia, joka on Pythonin graafinen käyttöliittymä -kirjasto. Onneksi komentorivi kertoi tästä RIDE:n käynnistysyrityksen yhteydessä (ks. kuvio 7), joten epäonnistumisen syyn etsimiseen ei tarvinnut käyttää aikaa. Kun wxPythonin oli asennettu, RIDE käynnistyi komennolla *ride.py*. (Installation Indroduction 2017.)

Kuvio 6. Komento, jolla asennetaan RIDE-editori.



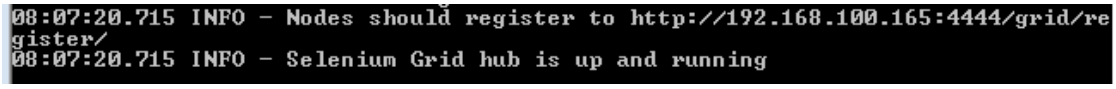
Kuvio 7. Komentorivin viesti puuttuvasta ohjelmasta.

5.2 Selenium Gridin asennus

Selenium Gridin käyttämistä varten tarvitaan Selenium Server jar -tiedosto, jonka voi ladata osoitteesta <http://www.seleniumhq.org/download/>. Tiedosto ladattiin kah-

delle eri tietokoneelle ja tiedoston sijainti lisättiin molemmissa Polku-ympäristömuuttujaan. Tämän jälkeen tietokoneelle jolla testit sijaitsevat, voitiin käynnistää hub komennolla *java -jar selenium-server-standalone-2.53.0.jar -role hub* (ks. kuvio 8). Komennon ajamisen jälkeen komentorivi kertoi käynnistymisen onnistuneen kuvion 9 mukaisesti.

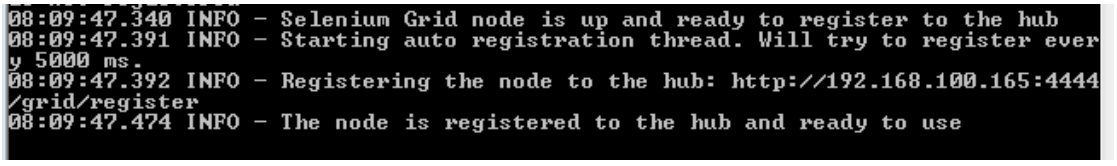
Kuvio 8. Komento, jolla käynnistetään Selenium Gridin hub.



Kuvio 9. Komentorivin ilmoitus siitä, että hub on käynnissä.

Tietokoneelle, jolla testit haluttiin suoritettavan, käynnistettiin node komennolla *java -jar selenium-server-standalone-2.53.0.jar -role node -hub http://192.168.100.165:4444/grid/register* (ks. kuvio 10). Komento yhdisti tietokoneen nodeksi hub-tietokoneeseen (ks. kuvio 11).

Kuvio 10. Komento, jolla yhdistetään node hubiin.



Kuvio 11. Komentorivin ilmoitus siitä, että node on rekisteröity hubiin.

5.3 Jenkinsin asennus

Seuraava vaihe oli asentaa jatkuvan integraation palvelimeksi valikoitunut Jenkins. Jenkins oli jo asennettu yhdelle toimeksiantajan servereistä, mutta asennus suoritettiin myös toiselle tietokoneelle harjoituksen vuoksi. Jenkinsin asentaminen vaatii minimissään Java 7 -ohjelman, 256 MB vapaata välimuistia ja 1 GB levytilaa (Jenkins user handbook n.d.). Jenkinsin asennus osoittautui erittäin helpoksi toimenpiteeksi. Jenkinsin Windows-asentajan saa ladattua osoitteesta <https://jenkins.io/download/>. Kun on ladannut ja purkanut tiedoston, ei tarvitse kuin ajaa tiedosto asentaakseen sovelluksen. Kun asennus oli valmis, Jenkins käynnistyi automaattisesti selaimen osoitteeseen <http://localhost:8080>. Ensimmäisen käyttäjätunnuksen luomiseen vaadittiin salasana, joka löytyi yhdestä asennuskansion tiedostosta. Käyttäjän luomisen jälkeen Jenkins ehdotti suosittujen liitännäisten asentamista - liitännäisten asennuksen valmistuttua Jenkinsin asennus oli valmis ja sovellukseen oli mahdollista kirjautua sisään.

Kun Jenkins oli käyttövalmis, siihen oli asennettava Robot Framework -liitännäinen, jotta sovellus pystyy luomaan ja julkaisemaan suoritetun Robot Framework -testin tulokset. (Robot Framwork Plugin 2016). Jos Jenkinsin avulla suoritetaan Robot Framework -testejä toisilla tietokoneilla, itse Robot Frameworkin tarvitsee olla asennettuna vain sillä koneella, missä Jenkinskin on.

Tässä vaiheessa tarvitsi enää yhdistää olemassa oleva Selenium Gridin asennus Jenkinsiin. Tutustumalla Jenkinsin Selenium-liitännäiseen selvisi, että operaatio on helpointa tehdä kyseisen liitännäisen avulla, sillä liitännäinen käynnistää automaattisesti asentamisensa jälkeen Selenium Grid hubin porttiin 4444. Kun hub on pystyssä, siihen voidaan yhdistää node-koneita opitulla tavalla. Liitännäisen asennuksen jälkeen hub ei kuitenkaan käynnistynyt automaattisesti, koska portti oli jo käytössä aikaisemmin käynnistetyllä hubilla. Jenkinsin asetuksista olisi voinut määrittää hubille uuden portin, mutta sille ei ollut tarvetta, koska aikaisemmin luotu hub-node -yhteys oli sama purkaa. Kun yhteys oli purettu, hub käynnistyi Jenkinsissä. Seuraavaksi oli mahdollista yhdistää testikone Jenkinsin hubiin. (Selenium Plugin 2017.)

6 Automatisoidun käyttöliittymätestin tekeminen ja suorittaminen

Kun työkalut oli asennettu ja yhdistetty toisiinsa, oli aika tehdä ensimmäinen automatisoitu testi ja kokeilla sen suorittamista luodussa ympäristössä.

6.1 Testin kirjoittaminen

RIDE:llä testit luodaan aina testijoukon (testsuite) alle. Testijoukko voi nimensä mukaisesti sisältää useita testitapauksia. Testijoukkoon liitetään ne avainsanakirjastot, joita testeissä halutaan käyttää. Testijoukolle on mahdollista luoda muuttujia tai muuttujat voidaan liittää myös erillisinä tiedostoina. Testit luodaan käyttäen avainsanoja. RIDE kertoo, mitä kukin avainsana tarvitsee sovelluksen lähdekoodista kyetäkseen suoriutumaan itselleen määritetystä toimenpiteestä. Esimerkiksi Selenium2Library-avainsanakirjaston avainsana: ”Click Button” (paina näppäintä), sisältää yhden argumentin, johon määritetään painettavan näppäimen id-, nimi- tai arvotunniste. Ensimmäiseksi tehtiin yksinkertainen testi käyttäen selenium2library-kirjastoa, joka ainoastaan avaa Chrome-selaimen Googlen etusivulle.

6.2 Testin suorittaminen

Kun testiskripti oli valmis, täytyi testata toimiiko se. Testin ajaminen ei kuitenkaan vielä ensimmäisellä kerralla onnistunut. RIDE, jolla testi käynnistettiin, ilmoitti seuraavaa: ”Message: 'chromedriver' executable needs to be in PATH. Please see https://sites.google.com/a/chromium.org/chromedriver/home”. Virheilmoitus johtui siitä, että Selenium2Library-kirjaston tarvitsema selain-ajuri oli unohtunut asentaa. Chrome-selaimen käyttämisen mahdollistama web-ajuri on virheilmoituksessa mainittu ChromeDriver. Kun ChromeDriver oli ladattu ja asennuskansion sijainti lisätty Polku-järjestelmämuuttujaan, testin suorittaminen onnistui. (ChromeDriver n.d). Samalla asennettiin myös GeckoDriver-ajuri Mozillan Firefoxia varten ja InternetExplorerDriver-ajuri Windowsin Internet Exploreria varten. (Selenium Downloads n.d.)

Seuraavaksi testattiin, miten testin suorittaminen onnistuu Jenkinsin avulla. Jenkinsiin luotiin uusi projekti, johon laitettiin sama komento, jolla testin saa käynnistettyä

komentoriviltä käsin. Kun projekti käynnistettiin, Jenkins käynnisti testin tietokoneella onnistuneesti.

Kun ensimmäisen testin suorittaminen Jenkinsin avulla oli onnistunut, oli aika testata testin suorittaminen toisella tietokoneella node-hub -yhteyden välityksellä. Ennen kun yhteys pystyttiin testaamaan, täytyi oppia ohjaamaan RIDE:llä tehty testiskripti hubiin. Vastaus löytyi Selenium2Library-avainsanakirjaston käyttöohjeesta (2016). Kun avainsanan, joka aukaisee testissä selaimen (Open Browser), neljänteen argumenttiin (remote_url) määritetään hubin osoite, niin testi ohjautuu hubille. Hub välittää testin sitten sopivalle, vapaana olevalle nodelle.

Ensimmäisellä yrittämällä päivitetyn testin suorittaminen node-koneella ei onnistunut, koska node-koneelle asennetun selainajurin sijainti oli laitettu väärin koneen Polku-järjestelmämuuttujaan. Kun asia korjattiin, projektin käynnistäminen Jenkinsissä käynnisti testin toisella koneella.

6.3 Hyväksymistestien tekeminen

Kun tietotaito testien tekemiselle ja suorittamiselle oli hankittu, oli aika aloittaa ensimmäisten testien tekeminen yrityksen web-sovellukselle. Aivan ensimmäiseksi oli tarvetta testille, joka ainoastaan kirjautuu sovellukseen sisälle ja ulos. Tarkoituksena testin avulla on varmistaa sovelluksen pystyssä oleminen. Kun testi oli valmis ja toimiva, luotiin Jenkinsiin projekti, joka suorittaa testin kolmen minuutin välein. Jenkins määritettiin myös lähettämään kaikille ohjelmistotiimin jäsenille sähköpostiviesti, mikäli testin suorittaminen epäonnistuu. Testin ansiosta tieto sovelluksen mahdollisesta kaatumisesta saadaan nopeasti toisin kuin ennen testiä, jolloin tieto sovelluksen kaatumisesta tuli useimmiten vasta asiakkaalta.

Seuraavaksi aloitettiin varsinaisten versiopäivitysten yhteydessä suoritettavien hyväksymistestien tekeminen. Sovelluksen eniten käytetyille moduuleille haluttiin ensimmäiseksi testitapaukset, jotka navigoivat moduulit läpi tarkistaen jokaisen näkymän sisällön oikeellisuuden. Kun kyseiset testit olivat valmiit, luotiin Jenkinsiin jokaiselle testille oma projekti. Sovelluksen uuden version julkaisun kynnyksellä nämä projektit voidaan suorittaa nappia painamalla julkaisuehdokkaalle. Testeistä tehtiin

sellaisia, että niihin on mahdollisimman helppo liittää tulevaisuudessa jatkoksi yksilöllisempiä testitapauksia.

6.4 Esimerkki hyväksymistestistä

Toimeksiantajan toiveesta heidän sovellukselleen suunnattua testiskriptiä ei julkaista tässä työssä. Havainnollistamisen vuoksi työn liitteeksi tehtiin kuitenkin esimerkkitesti toiselle web-sovellukselle. Testistä haluttiin sellainen, joka kuvaa mahdollisimman hyvin, sitä kaikkea, mitä on mahdollista automatisoida. Testin kohteena päätettiin käyttää Googlen sovelluksia, koska ne ovat yleisesti tunnettuja. Testin eri vaiheet hahmoteltiin ensiksi paperille, ja päädyttiin lopulta testiin, jossa on seuraavat vaiheet:

1. Kirjautuminen aikaisemmin luodulle Google-tilille.
2. Haku Kartat-sovelluksessa käyttäen Jamk:in osoitetta.
3. Jamk:in pihamaalla otetun 360-asteen kuvan avaaminen ja siitä neljän näyttökuvan ottaminen, yksi jokaisesta ilmansuunnasta.
4. Otettujen näyttökuvien tallentaminen Drive-pilvipalveluun.
5. Kuvien katsominen Drive-pilvipalvelussa.
6. Sähköpostin lähetys kyseiset kuvat viestiin liitettynä.
7. Kirjautuminen ulos Google-tililtä.

Tätä testitapausta tehtäessä vastaan tuli tilanne, jossa oli tarvetta käyttää näppäimistön painikkeita. Ratkaisua etsittäessä löytyi ImageHorizonLibrary-avainsanakirjasto. Kirjasto sisältää Press combination -avainsanan, joka mahdollistaa näppäimistön käyttämisen testeissä. (ImageHorizonLibrary 2015.)

Testatessa avainsanan toimivuutta samalla tietokoneella Robot Frameworkin asennuksen kanssa ongelmia ei esiintynyt. Kun luotiin Jenkinsiin projekti, joka suoritti saman testin toisella tietokoneella Selenium Grid -yhteyden avulla, ongelmia ilmeni. Testi kaatui jokaisella suoritusyrityksellä ensimmäisen ImageHorizonLibrary-kirjaston avainsanan kohdalle.

Vastaus ongelmaan löytyi vasta kysymällä apua muilta käyttäjiltä Google Ryhmät -palvelun robotframework -ryhmässä. Kävi ilmi, että kaikki avainsanakirjastot eivät tue testin suorittamista toisella tietokoneella. Keskustelufoorumilla neuvottiin ratkai-

semaan ongelma käyttämällä Robot Frameworkin Remote Library Interface -ominaisuutta. (Tatu Aalto 2017.)

Robot Frameworkin käyttöohjeen mukaan kyseisen ominaisuuden avulla on mahdollista käyttää testissä toiselle tietokoneelle asennettua avainsanakirjastoa. Eli kirjastoa, joka ei tue etäkäyttöä, voidaan käyttää, kun se asennetaan samalle tietokoneelle, jolla testi suoritetaan. Kirjastoon saadaan yhteys Robot Frameworkin Remote Library Interface -ominaisuuden avulla. Ominaisuus toimii siten, että koneelle, jossa avainsanakirjasto sijaitsee, asennetaan etäpalvelin (remote server). Tähän palvelimeen otetaan sitten yhteys Robot Frameworkin sisäänrakennetun Remote Library -kirjaston avulla. Remote Library -kirjasto ei sisällä avainsanoja, vaan se toimii välityspalvelimena viitekehyksen ja toiselle koneelle asennetun avainsanakirjaston välillä. (Robot Framework User Guide 2017.)

Toiminnallisuuden käyttäminen vaati siis etäpalvelimen testikoneelle. Mahdollisia etäpalvelimia Robot Frameworkille on useita, joista yksi on PythonRemoteServer. Koska tähänkin asti oli käytetty Pythonia, PythonRemoteServer valikoitui käytettäväksi etäpalvelimeksi. Asennus oli yksinkertainen suorittaa; täytyi vain ajaa komento *pip install robotremoteserver*. (PythonRemoteServer n.d.)

Ennen palvelimen käynnistämistä testikoneelle täytyi asentaa tarvittava avainsanakirjasto, joka siis oli tässä tapauksessa ImageHorizonLibrary-kirjasto. Koska testikoneelle ei ollut asennettu vielä Robot Frameworkiä, täytyi toistaa kappaleessa 5.1 kerrotut vaiheet Robot Frameworkin asentamisesta, minkä jälkeen kirjaston asennus onnistui komennolla *pip install robotframework-imagehorizonlibrary*. Tässä vaiheessa oli aika käynnistää etäpalvelin. Se tapahtui ajamalla kuviossa 13 näkyvä komento Pythonin komentorivillä. Komento käynnisti etäpalvelimen osoitteeseen <http://192.168.100.163:5550>. Jos osoitetta ja porttia ei määritä itse, palvelin käynnistyy osoitteeseen <127.0.0.1:8270>. (Mt.)

```
>>> from robotremoteserver import RobotRemoteServer
>>> from ImageHorizonLibrary import ImageHorizonLibrary
>>> RobotRemoteServer(ImageHorizonLibrary(),host='192.168.100.163', port=5550)
Robot Framework remote server at 192.168.100.163:5550 starting.
```

Kuvio 12. Komento, jolla käynnistetään etäpalvelin Robot Frameworkille.

Jotta testi osasi ottaa yhteyttä pystytettyyn palvelimeen, täytyi vain lisätä testiin Remote-kirjasto ja laittaa sen ensimmäiseksi argumentiksi etäpalvelimen osoite, eli <http://192.168.100.163:5550> (Robot Framework User Guide 2017).

Ensimmäisellä yrittämällä tämän ominaisuuden testaaminen ei onnistunut, vaan testi kaatui taas ensimmäisen ImageHorizonLibrary-kirjaston avainsanan kohdalla. Ongelman löytämiseksi testattiin testikoneen palomuurin osuutta asiaan. Siksi toista testikertaa varten kytkettiin palomuuri pois päältä. Tällä kertaa etäpalvelimen ja sen sisältämän avainsanakirjaston käyttö onnistui, joten oli selvää, että epäonnistuminen johtui palomuurista. Tästä syystä palomuuriin määritettiin sääntö, joka sallii toisen tietokoneen yhteydenoton testitietokoneen Python-ohjelman kanssa (Python, koska etäpalvelin oli Pythonin ylläpitämä). Määrityksen jälkeen etäpalvelimen käyttö onnistui myös palomuurin ollessa päällä.

Valmiin testin testiskripti näkyy liitteen 1 kuvissa. Kuvat on otettu tekstitiedostosta, jollaiseksi RIDE tallentaa sillä tehdyt testit.

7 Tulokset ja jatkokehitys

Työn tuloksena saatiin rakennettua toimiva ympäristö web-sovelluksen hyväksymistestauksen automatisoinnille. Työssä käytettiin kahta tietokonetta, joille asennettiin tarvittavat ohjelmat testien luomiselle, hajauttamiselle ja suorittamiselle. Koneelle, jolla testit luodaan, asennettiin Python, Robot Framework, Selenium2Library- ja ImageHorizonLibrary-avainsanakirjasto, wxPython, RIDE sekä Jenkins. Jenkinsiin asennettiin liitännäiset sekä Robot Frameworkille, että Selenium Gridille. Koneelle, jolla testit suoritetaan, asennettiin Python, Robot Framework, selain-ajurit, ImageHorizonLibrary -avainsanakirjasto ja RobotRemoteServer. Kummallekin tietokoneelle ladattiin myös Selenium server jar -tiedosto, joista loppujen lopuksi tarvittiin vain testikoneelle ladattua, koska Jenkinsin Selenium-liitännäinen sisälsi kyseisen tiedoston.

Pythonin, Robot Frameworkin, wxPythonin, RIDE:n ja Slenium2Libraryn avulla automatisoituja testejä on mahdollista luoda ja Jenkinsin avulla niiden suorittamista voidaan hallita. Robot Framework -liitännäisen avulla Robot Frameworkin luomat kattavat raportit saadaan luotua ja julkaistua Jenkinsissä. Selenium-liitännäisen ja Selenium server jar -tiedoston avulla saatiin muodostettua hub-node -yhteys tietokoneiden välille, mikä mahdollisti testien suorittamisen node-koneella. Testit avautuvat node-koneella selaimiin asennettujen selain-ajureiden ansiosta ja RobotRemoteServer-etäpalvelimen avulla on mahdollista käyttää testeissä avainsanakirjastoja, jotka eivät tue etäkäyttöä.

Hyväksymistestien tekeminen saatiin alulle ja sovelluksen pystyssä olemisen varmistava testi käyttöön. Hyväksymistestien luomista jatketaan niin, että kaikki sovelluksen perusominaisuudet tulisivat testattua automatisoiduilla testeillä. Myös testausprosessia on tarkoitus viedä tulevaisuudessa eteenpäin. Tarkoituksena on ottaa käytäntöön jatkuva integrointi mahdollisimman pian. Kun koko ketju lähdekoodin muutoksesta uuden version luomiseen ja automatisoitujen testien käynnistymiseen saadaan valmiiksi, voidaan puhua täysin automatisoidusta testauksesta.

8 Pohdinta

Testauksen automatisointi tuo helpotusta testausvaiheessa usein vallitsevaan ajanpuutteeseen. Kokonaan testausta ei voi kuitenkaan jättää automaatiotestauksen harteille. Yrityksen web-pohjainen sovellus on niin moniulotteinen ja laaja, ettei kaikkia testitapauksia ole mahdollista automatisoida. Automatisoituihin testeihin ei pystytä saamaan tarpeeksi vaihtuvuutta, jotta kaikki käyttötapaukset saataisiin niiden avulla testattua. Automatisointi on kuitenkin järkevää usein testattavien perusominaisuuksien kohdalla.

Haasteena testausautomaatiossa on saada testeistä toimivia. Testien tekemisessä tulee olla todella huolellinen ja perusteellinen, jotta testit eivät kaadu testeissä olevien virheiden takia sovelluksessa olevien virheiden sijasta. Isoin tekijä tähän on testaajan osaaminen ja huolellisuus, mutta myös sovelluksen koodilla on oma merkityksensä. Koodista ei aina löydy suoraan tarvittavia tunnisteita avainsanoja varten, eivätkä kaikki elementit välttämättä ole sitä, miltä ne näyttävät käyttöliittymässä. Onneksi koodia on kuitenkin aina mahdollista päivittää ja selkeyttää. Internet-yhteyden hetkelliset katkokset ovat myös ongelma testausautomaatiossa, koska ne voivat aiheuttaa testien epäonnistumisia ja sitä kautta turhia hälytyksiä. Testausautomaation haasteet ovat kuitenkin helppo antaa anteeksi, koska hyviä puolia on niin paljon. Automaation avulla saadaan nopeutettua testausta merkittävästi, helpotettua uudelleentestausta ja vähennettyä manuaalisen työn tarvetta.

Työssä käytetyt työkalut mahdollistavat automatisoinnin kehittämisen tulevaisuudessa. Robot Framework on toimiva viitekehys testien kattavaan automatisointiin. Se, miten monipuolisia ja helposti ylläpidettäviä testitapauksista saa, on täysin kiinni testitapausten tekijästä – viitekehyksen rajat eivät tule heti vastaan. Viitekehys sisältää paljon muitakin ominaisuuksia kuin ne, jotka tulivat ilmi tässä työssä. Selenium Grid toimii hyvin testien hajauttamiseen - hub-node -yhteys ei ole katkeillut ja uusia nodeja on helppo lisätä hubiin tulevaisuudessa, mikäli testikoneita tulee lisää. Myös Jenkins osoittautui oikein hyvin toimivaksi työkaluksi. Jenkinsin avulla automatisointia on mahdollista kehittää tulevaisuudessa paljon. Kaikki nämä sovellukset ovat hyvin suosittuja ja eteenpäin vietyjä säännöllisten päivitysten ansiosta. Jos sovellukselle ei ilmesty uusia päivityksiä säännöllisesti, eikä siitä käydä keskustelua, sen kehitys on

todennäköisesti loppunut. Tämä johtuu yleensä siitä, että joku on tehnyt saman asian paremmin ja vienyt käyttäjät.

Kokonaiskuvaa ajatellen työn tulosten avulla saatiin toimeksiantajayrityksessä otettua iso askel kohti järjestelmällisempää testausta. Työssä toteutettua testausmenetelmää käytetään ja tullaan käyttämään jatkossa yhä enemmän isona osana yrityksen web-sovelluksen laadunvarmistusta.

Lähteet

Aalto, T. 2017. Keskustelu robotframework-users -ryhmässä Google Ryhmät -palvelussa. Viitattu 4.4.2017. https://groups.google.com/forum/#!topic/robotframework-users/eVwQxRWpheo

About Jenkins. N.d. Artikkeli Cloudbeessin www-sivuilla. Viitattu 17.3.2017. https://www.cloudbees.com/jenkins/about

Awards. 2017. Listaus Jenkins-sovelluksen voittamista palkinnoista. Viitattu 8.4.2017. https://wiki.jenkins-ci.org/display/JENKINS/Awards

Berglund, M. 2015. Miten käyttöliittymä testataan automaattisesti –artikkeli Develoren www-sivuilla. Viitattu 25.2.2017. http://www.develore.com/artikkeli/miten-kayttoliittyma-testataan-automaattisesti/

ChromeDriver. N.d. ChromeDriverin kotisivu. Viitattu 13.3.2017. https://sites.google.com/a/chromium.org/chromedriver/

Ghahrai, A. 2015. Test automation advantages and disadvantages -artikkeli Testing Excellencen www-sivuilla. Viitattu 20.3.2017. http://www.testingexcellence.com/test-automation-advantages-and-disadvantages/

Grid2. 2.8.2016. Selenium Gridin esittely Githubin www-sivuilla. Viitattu 27.3.2016. https://github.com/SeleniumHQ/selenium/wiki/Grid2

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

How To Use Selenium Grid. N.d. 52. artikkeli Elemental Seleniumin www-sivuilla. Viitattu 3.4.2017.

ImageHorizonLibrary. 2015. Avainsanakirjaston käyttöohje. Viitattu 4.4.2017. http://eficode.github.io/robotframework-imagehorizonlibrary/doc/ImageHorizonLibrary.html

Installation Introduction. N.d. RIDE-ohjelman asennusohje. Viitattu 20.3.2017. https://github.com/robotframework/RIDE/wiki/Installation-Instructions

Jenkins documentation N.d. Jenkinsin esittely dokumentaatiot-osiossa Jenkinsin www-sivuilla. Viitattu 17.3.2017. https://jenkins.io/doc/

Jenkins user handbook. N.d. Viitattu 8.4.2017. https://jenkins.io/user-handbook.pdf

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. 1. p. Jyväskylä: Docendo

Kelly, M. 2003. Choosing test automation framework. Viitattu 5.5.2017. https://www.ibm.com/developerworks/rational/library/591-pdf.pdf

Lewis, W. E. 2004. Software testing and Continuous Quality Improvement. USA: Auerbach

Most popular test automation frameworks. 2016. Artikkeli Software Testing Helpin www-sivuilla. Viitattu 23.3.2017. http://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/

Pettichord, B. 2001. Succes with Test Automation -artikkeli. Viitattu 5.5.2017. http://ceng557.cankaya.edu.tr/uploads/files/Success%20with%20Test%20Automation.doc

PythonRemoteServer N.d. Python-etäpalvelimen käyttöohje. Viitattu 4.4.2017. https://github.com/robotframework/PythonRemoteServer

RIDE – How to. 2012. RIDE:n ominaisuuksien esittely Githubin www-sivuilla. https://github.com/robotframework/RIDE/wiki/How-To

Robot Framework. N.d. Viitekehyksen kuvaus Robot frameworkin www-sivuilla. Viitattu 20.2.2017. http://robotframework.org

Robot Framework Plugin. 2016. Robot Framework Jenkins liittännäisen www.sivu. Viitattu 28.3.2017. https://wiki.jenkins-ci.org/display/JENKINS/Robot+Framework+Plugin

Robot Framework User Guide. 2017. Robot Frameworkin käyttöohje. Viitattu 17.3.2017. http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

Selenium downloads. N.d. Listaus ladattavista web-drivereista Selenium HQ:n www-sivuilla. Viitattu 26.3.2017. http://www.seleniumhq.org/download/

Selenium grid. 2017. Selenium gridin käyttöohje dokumentaatiot-osiossa SeleniumHQ:n www-sivuilla. Viitattu 17.3.2017. http://www.seleniumhq.org/docs/07_selenium_grid.jsp

Selenium IDE. 2017. Selenium IDE:n käyttöohje dokumentaatiot-osiossa SeleniumHQ:n www-sivuilla. Viitattu 17.3.2017. http://www.seleniumhq.org/docs/02_selenium_ide.jsp

Selenium Introduction. 2017. Introduction-artikkeli dokumentaatiot-osiossa SeleniumHQ:n www-sivuilla. Viitattu 17.3.2017. http://www.seleniumhq.org/docs/01_introducing_selenium.jsp#

Selenium2Library. 2016. Selenium2Library-avainsanakirjaston käyttöohje Robot Frameworkin www-sivuilla. Viitattu 28.3.2017. http://robotframework.org/Selenium2Library/Selenium2Library.html

Selenium2Library Installation. 2017. Avainsanakirjaston asennusohje Githubin www-sivuilla. Viitattu 26.3.2017. https://github.com/robotframework/Selenium2Library/blob/master/INSTALL.rst

Selenium Plugin. 2017. Selenium liitännäisen kotisivu. Viitattu 3.4.2017. https://wiki.jenkins-ci.org/display/JENKINS/Selenium+Plugin

Sensire yrityksenä. 2017. Yritys esittely Sensire Oy:n www-sivuilla. Viitattu 20.2.2017. http://www.sensire.fi/fi/yritys

Sommerville, I. 2016. Software engineering. 10. p. Englanti: Pearson Limited.

Testing Trends in 2017. 2017. Ohjelmisto-alan ammattilaisille toteutettu kyselytutkimus. Viitattu 6.5.2017. http://info.saucelabs.com/rs/468-XBT-

687/images/Sauce%20Labs%20-%20State%20of%20Testing%20Survey%20Results%20Jan,%202017.pdf

Vingrys, K. 2010. Acceptance test automation. Viitattu 7.4.2017. https://www.thoughtworks.com/insights/blog/acceptance-test-automation

What is automation testing? N.d. Artikkeli Software Testing Classin www-sivuilla. Viitattu 22.3.2017. http://www.softwaretestingclass.com/what-is-automation-testing/

Why automated testing. N.d. Artikkeli Smartbearin www-sivuilla. Viitattu 20.3.2017. https://support.smartbear.com/articles/testcomplete/manager-overview/

Wikipedia, RobotFramework. 7.3.2017. Robot Frameworkin Wikipedia-sivu. Viitattu 2.4.2017. https://en.wikipedia.org/wiki/Robot_Framework

Liitteet

Liite 1. Google-testin testiskripti

```
*** Settings ***
Library           Selenium2Library
Library           BuiltIn
Library           Remote    http://192.168.100.163:5550


*** Test Cases ***
Google
    Kirjaudu Google-tilille
    Etsi Jamk kartalta
    Google drive
    Lähetä sähköposti
    Kirjaudu ulos
    [Teardown]    Close Browser

*** Keywords ***
Kirjaudu Google-tilille
    Open browser    http://www.google.fi    chrome    none    http://192.168.100.103:4444/
wd/hub
    Maximize browser window
    Set selenium speed    0.5
    Wait Until Element Is Visible    id=gb_70
    Click Element    id=gb_70
    Wait Until Element Is Visible    id=Email
    Input Text    id=Email    Automaatio1
    Click Element    id=next
    Wait Until Element Is Visible    id=Passwd
    Input Password    id=Passwd    250893kvI
    Click element    id=signIn

Etsi Jamk kartalta
    Wait Until Element Is Visible    id=gbwa    30
    Click Element    id=gbwa
    Wait Until Element Is Visible    id=ogbkddg:2    30
    Click Element    id=ogbkddg:2
    Wait Until Element Is Visible    id=searchboxinput    30
    Input Text    id=searchboxinput    Jyväskylän ammattikorkeakoulu, Rajakatu 35
Jyväskylä
    Click Element    id=searchbox-searchbutton
    Wait Until Element Is Visible    id=widget-expand-button    30
    Click Element    id=widget-expand-button
    Wait Until Element Is Visible    //*[@id="runway"]/div/div/div[5]/div/ul/span[1]    30
    Click Element    //*[@id="runway"]/div/div/div[5]/div/ul/span[1]
    Set selenium speed    1s
    Wait Until Element Is Visible    //*[@id="compass"]/div/button[2]
    Click Element    id=widget-expand-button
    Capture Page Screenshot
    Click Element    //*[@id="compass"]/div/button[2]
    Capture Page Screenshot
    Click Element    //*[@id="compass"]/div/button[2]
    Capture Page Screenshot
    Click Element    //*[@id="compass"]/div/button[2]
    Capture Page Screenshot
    Click Element    //*[@id="compass"]/div/button[2]
    Click element    //*[@id="titlecard"]/div[1]/button
    Set Selenium Speed    0.5
```

```
Google drive
    Wait Until Element Is Visible    id=gbwa    30
    Click Element    id=gbwa
    Wait Until Element Is Visible    id=ogbkddg:6    30
    Click Element    id=ogbkddg:6
    Wait Until Element Is Visible    id=:1c.0B3J2--auMmx6aHZlVzE5cDc5UVk    15s
    Sleep    2s
    Double Click Element    id=:1c.0B3J2--auMmx6aHZlVzE5cDc5UVk
    Sleep    3s
    Press Combination    key.right
    Sleep    2s
    Press Combination    key.right
    Sleep    2s
    Press Combination    key.right
    Sleep    2s
    Press Combination    key.esc

Lähetä sähköposti
    Wait Until Element Is Visible    id=gbwa    15
    Click Element    id=gbwa
    Wait Until Element Is Visible    id=ogbkddg:5    15
    Click Element    id=ogbkddg:5
    Get Window Titles
    Select Window    title=Postilaatikko - automaatio1@gmail.com - Gmail
    Wait Until Element Is Visible    id=:3b    15
    Click Element    id=:3b
    Wait Until Element Is Visible    id=:8e    15
    Input Text    id=:8e    jjvayrynen@gmail.com
    Input Text    name=subjectbox    Kuvia Jamkista
    Input Text    id=:90    Hei, Tässä muutama kuva Jamkista. T: Automaatio
    Sleep    1s
    Click Element    id=:9s
    Sleep    2s
    Select Frame    xpath=//div[@class='KA Kj-JD picker-dialog']/div/iframe
    Press Combination    KEY.CTRL    key.a
    Click Element    id=:h
    Click element    id=picker:ap:0
    Sleep    10s
    Click element    id=:7o

Kirjaudu ulos
    Wait Until Element Is Visible    //*[@id="gb"]/div[1]/div[1]/div[2]/div[4]/div[1]/a
    Click Element    //*[@id="gb"]/div[1]/div[1]/div[2]/div[4]/div[1]/a
    Wait Until Element Is Visible    id=gb_71
    Click Element    id=gb_71
    Wait Until Element Is Visible    id=rsi-card
```