Lauri Leirost

AVOIMEN LÄHDEKOODIN TESTIAUTOMAATIOJÄRJESTELMÄN VALINTA

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Tekniikan ja liikenteen ala
Tietotekniikan koulutusohjelma
Kevät 2014

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Tekniikan ja liikenteen ala	Koulutusohjelma Tietotekniikka
Tekijä(t) Lauri Leirost	
Työn nimi Avoimen lähdekoodin testiautomaatiojärjestelmän valinta	
Vaihtoehtoiset ammattiopinnot	Toimeksiantaja Elektrobit
Aika Kevät 2014	Sivumäärä ja liitteet 39

Tämän opinnäytetyön tavoitteena oli vertailla avoimen lähdekoodin testiautomaatiojärjestelmiä työn tilanneelle Elektrobitille. Vertailusta saatujen tulosten ja johtopäätösten perusteella Elektrobit voi arvioida tulevaisuudessa sopivan testiautomaatiojärjestelmän tilannekohtaisesti.

Vertailussa pyrittiin löytämään vastaukset seuraaviin testiautomaatiojärjestelmiin liittyviin asioihin: tekniset ominaisuudet, käytettävyys, ylläpidettävyys, taloudellisuus ja käyttöönotto. Tutkimukseen kuului haastatteluosuus, jossa haastateltiin testiautomaatiokokemusta omaavia Elektrobitin työntekijöitä. Haastatteluiden tuloksia käytettiin opinnäytetyön johtopäätöksien muodostamiseen.

Testiautomaatio on ohjelmistotestauksen osa. Siinä erityisohjelmisto hallinnoi testien suorittamista ja vertailee testeistä saatuja tuloksia odotettuihin tuloksiin. Testiautomaation päätarkoituksena on varmistaa jo testattujen osien toimivuus. Tämä on osa tuotteen laadunvalvontaprosessia.

Vertailtavat testiautomaatiojärjestelmät olivat geneerinen ymmärrettävien testitapausten luonnin mahdollistava Robot Framework, verkkoliikenneyrityksille suunniteltu Twister Framework ja vertaisympäristöä hyödyntävä Software Testing Automation Framework (STAF).

Vertailun mukaan Twister Framework oli testatuista järjestelmistä kypsin. Se oli lähes jokaisella tutkitulla osa-alueella verrokkeja pitemmälle viety. Robot Framework oli sitä parempi testitapausten luomiseen. STAF ei sisältänyt samassa mittakaavassa valmiita ominaisuuksia kuin Twister Framework, vaan ne oli jätetty käyttäjän tehtäväksi ja muokattavaksi.

Kieli	Suomi
Asiasanat	Avoin lähdekoodi, testiautomaatio, ohjelmistotestaus
Säilytyspaikka	☒ Verkkokirjasto Theseus ☐ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS ABSTRACT

School Engineering	Degree Programme Information Technology
Author(s) Lauri Leirost	
Title Choosing an Open Source Test Automation Framework	
Optional Professional Studies	Commissioned by Elektrobit
Date Spring 2014	Total Number of Pages and Appendices 39

The purpose of this Bachelor's thesis was to make a comparison on available Open Source Test Automation Frameworks for Elektrobit. Based on the results and conclusions of this thesis Elektrobit will have the possibility of choosing the most appropriate Test Automation Framework depending on their needs.

The comparison was done in order to find answers to technical features, the usability, maintainability, economic efficiency and installation process of Test Automation Frameworks. One part of this thesis was to interview Elektrobit employees who have experience in Test Automation and use their opinion to form the results and conclusions of the comparison.

Test Automation is part of Software Testing. It is specialized software that runs tests on target software automatically with predefined test cases and expected outcomes. The objective of test automation is to ensure that already tested software components will not break. Test Automation is part of the quality assurance of the product.

Test Automation Frameworks included in this research were Robot Framework, Twister Framework and Software Testing Automation Framework (STAF). Robot Framework is a generic test automation framework that aims at easy test case creation. Twister Framework is designed to be used with multiple communications-specific test devices. STAF is a highly extensible framework used in peer environment.

The comparison revealed that Twister Framework is more mature than the other tested frameworks and beats them in almost every category. For test case creation Robot Framework is best of the three. STAF does not include as much predefined parts as Twister Framework, as it is supposed to be used as an expendable platform.

Language of Thesis	Finnish
Keywords	Open Source, Test Automation, Software Testing
Deposited at	☒ Electronic library Theseus ☐ Library of Kajaani University of Applied Sciences

ALKUSANAT

Haluan kiittää Elektrobitiä ja Jussi Heikkistä työn mahdollistamisesta, haastateltavia mielipiteistään, Pentti Romppaista työn valvomisesta, Eero Soinista kielellisestä ohjauksesta ja Seija Heikkistä abstractin ohjauksesta.

Kajaanissa 22.04.2014
Lauri Leirost

SISÄLLYS

SYMBOLILUETTELO

FSF	Free Software Foundation
HTML	HyperText Markup Language
OSI	Open Source Initiative
SSH	Secure Shell
STAF	Software Testing Automation Framework
STAX	STAf eXecution engine
SUT	System Under Test
XML	Extensible Markup Language

1 JOHDANTO

Ohjelmistovirheet eivät koskaan ole ilmaisia rahallisesti tai imagollisesti. Niiden olemassaolo pyritään tästä syystä kaikin keinoin minimoimaan. Testiautomaation avulla pyritään varmistamaan jo testattujen osioiden toimivuus kehityksen jatkuessa. Se on tärkeä osa ohjelmistoprojektin laadunvalvontaa ja tästä syystä opinnäytetyön tilannut Elektrobit on asiasta kiinnostunut.

Elektrobit Oyj on Oulussa kotipaikkaa pitävä julkinen osakeyhtiö [1]. Yritys toimii kahdeksassa maassa ja vuonna 2013 se työllisti 1648 ihmistä [2][3]. Elektrobit-konserni on jaettu toimialoittain kahteen liiketoimintasegmenttiin (alakonserniin), jotka ovat Automotive ja Wireless [1]. Automotive-segmentissä Elektrobit tarjoaa ohjelmistotuotteita ja tuotekehityspalveluita autoteollisuudelle. Wireless-segmentissä tarjotaan langattoman tietoliikenteen ratkaisuja viranomais-, puolustus- ja turvallisuusmarkkinoille sekä teollisuuteen. [4.] Elektrobitin Kajaanin yksikkö on osa Wireless-segmenttiä.

Tämän opinnäytetyön tavoitteena oli vertailla avoimen lähdekoodin testiautomaatiojärjestelmiä Elektrobitille. Tulosten ja johtopäätösten perusteella yrityksellä on mahdollisuus valita itselleen tilannekohtaisesti sopivin testiautomaatiojärjestelmä.

Vertailussa huomioitiin seuraavat asiat: tekniset ominaisuudet, käytettävyys, taloudellisuus, ylläpidettävyys ja käyttöönotto. Tulosten ja johtopäätösten muodostamiseen tarvittiin mielipiteitä henkilöiltä, joilla on kokemusta testiautomaatiosta. Tätä tarkoitusta varten haastateltiin asian kanssa tekemisissä olevia Elektrobitin työntekijöitä.

2 KESKEISET KÄSITTEET

Seuraavien alaotsikoiden alla käsitellään tämän opinnäytetyön keskeiset käsitteet eli avoin lähdekoodi ja testiautomaatio.

2.1 Avoin lähdekoodi

Avoimella lähdekoodilla tarkoitetaan tietokoneohjelmaa, joka on lisensoitu avoimen lähdekoodin lisenssillä [5, s. 273]. Lisenssi tai tässä asiayhteydessä tarkennettuna ohjelmistolisenssi on oikeudellisesti sitova sopimus, joka ilmoittaa sovelluksen käyttöoikeudet. Se määrittää sekä ohjelman luojan että käyttäjän oikeudet [6]. Avoimen lähdekoodin lisenssin kriteerit on luonut Open Source Initiative (OSI). Se on Kaliforniassa vuonna 1998 perustettu voittoa tavoittelematon järjestö, jonka tehtävänä on arvioida ja hyväksyä lisenssejä avoimen lähdekoodin kriteereiden mukaisiksi. Lisäksi OSI myös kouluttaa ja tiedottaa ihmisiä ei-suljetun lähdekoodin hyödyistä. [7] Saavuttaakseen OSI:n hyväksynnän tulee lisenssin täyttää seuraavat kymmenen kriteeriä:

1. Ohjelman tulee olla vapaasti levitettävissä ja välitettävissä.

2. Ohjelman lähdekoodi tulee olla vapaasti saatavilla.

3. Lisenssin täytyy sallia johdettujen ohjelmien luominen ja levittäminen.

4. Ohjelman lähdekoodin levitystä muokattuna voidaan rajoittaa ainoastaan niin, että korjaustiedostojen levittäminen on sallittua. Johdettu ohjelma voi joutua käyttämään eri nimeä ja versionumerointia kuin alkuperäinen.

5. Yksilöiden ja ryhmien eriarvoistaminen ei ole sallittua.

6. Käyttötarkoitusten rajoittaminen ei ole sallittua.

7. Ohjelman mukana tulleet oikeudet koskevat kaikkia, jotka ovat sen saaneet käsiinsä.

8. Ohjelman oikeudet eivät saa riippua laajemmasta ohjelmistokokonaisuudesta. Mikäli ohjelma irrotetaan kokonaisuudesta, saa ohjelma alkuperäisen ohjelmistokokonaisuuden oikeudet.

9. Ei saa asettaa rajoituksia muihin mukana toimitettaviin ohjelmiin.

10. Tulee olla teknologioiden suhteen neutraali. Varauksia ei saa olla yksittäisten teknologioiden ja käyttöliittymien varjolla. [8.]

Yksinkertaistettuna avoin lähdekoodi tarkoittaa kaikkien nähtävissä, käytettävissä ja vapaasti muokattavissa olevaa ohjelmakoodia, jonka levittämistä ja käyttämistä ei ole rajoitettu.

OSI:n hyväksymiä avoimen lähdekoodin lisenssejä on 70 [9], joista tunnetuimmat ja käytetyimmät ovat GNU General Public License (GPL), GNU Lesser General Public License (LGPL), New BSD License, Common Development and Distribution License, Common Public License, MIT License, Apache License 2.0, Mozilla Public License 2.0 ja Eclipse Public License [10].

Lisenssien suuri lukumäärä johtuu niiden eri käyttötarkoituksista sekä eri ideologioista niiden taustalla. Ohjelmistoliiketoiminnan sopimukset -kirjassaan Välimäki on jakanut lisenssien ominaisuudet kuuteen osaan, jotka ovat: vapaa levitys, vapaa käyttö, avoin koodi, pysyvä, tarttuva ja verkkokäyttö [5, s. 205–207]. Näistä kuudesta kriteereistä avoimen lähdekoodin lisenssit sisältävät vähintään vapaan levityksen ja käytön sekä avoimen koodin. Lisenssit, jotka sisältävät vain nämä kolme ehtoa, ovat niin sanottuja yliopistolisenssejä, joiden päätarkoitus on antaa ohjelmiston käyttäjälle mahdollisimman paljon oikeuksia ja mahdollisimman vähän vastuuta. Yksi tunnetuimmista yliopistolisensseistä on yhdysvaltalaisen Massachusetts Institute of Technology -yliopiston luoma MIT License. Se on niin vapaa ettei se vaadi, että ohjelman lähdekoodi on avoinna ja saatavilla. Täten ohjelmaa voidaan levittää konekielisenä ilman lähdekoodia. Ainoa vaatimus MIT Licensessa on, että ohjelmasta ei saa koskaan poistaa MIT-lisenssiä, tekijöiden nimiä ja vastuuvapautuslauseketta. MIT-lisenssin vaatimukset ovat niin kevyet, että MIT-lisensoitu ohjelma voidaan lisätä osaksi kaupallista suljetun lähdekoodin ohjelmaa. [5, s. 208.]

Toinen lisenssityyppi yliopistolisenssien rinnalla on vastavuoroisuutta vaativat lisenssit eli copyleft-lisenssit. Vastavuoroisuutta vaativat lisenssit pyrkivät pitämään lisenssin jatkossakin avoimena. Tunnetuin vahvaa vastavuoroisuutta käyttävä lisenssi on GNU General Public License (GPL). Vahva vastavuoroisuus tarkoittaa, että GPL-lisensoitujen ohjelmien muunnelmat tulee lisensoida saman lisenssin alle, jolloin ei ole mahdollista muokata GPL-lisensoidusta ohjelmasta suljetun lähdekoodin versiota. [5, s. 210–212.]

Käytettäessä GPL-lisensoitua koodia toisen lisenssin kanssa tulee varmistaa lisenssien yhteensopivuus. Käytettäessä GPL-lisenssiä toisen lisenssin kanssa tilanne usein tulkitaan muunnelluksi työksi, joka tulee lisensoida GPL:n alle. Mikäli lisenssit ovat ristiriidassa, ei tuotosta voida julkaista lainkaan. [5, s. 224–225.]

GNU Lesser General Public License (LGPL) on niin sanotusti heikkoa vastavuoroisuutta vaativa lisenssi, joka on ensisijaisesti suunniteltu ohjelmakirjastojen lisensointiin. LGPL:llä lisensoiduista ohjelmista muokatut ja johdetut teokset täytyy lisensoida LGPL:llä tai GPL:llä. LGPL-lisenssi ei vaadi siihen linkitetyn ohjelmiston lisenssiltä mitään, jolloin sitä voidaan käyttää myös suljetun lähdekoodin sovelluksissa, vaikkakaan sitä ei saa lisensoida muun lisenssin alle. [5, s. 217.]

Ennen vuotta 1998 oli tietokoneohjelmia, joiden lähdekoodi oli avointa, mutta tuolloin ne tunnettiin paremmin nimellä ”vapaat ohjelmistot”. Tämä termi otettiin käyttöön vuonna 1983 Richard M. Stallmanin perustaman GNU-projektin toimesta, ja vuonna 1985 vapaa ohjelmisto -liikkeen virallinen kattojärjestö Free Software Foundation (FSF) aloitti toimintansa [11]. Vuonna 1989 Free Software Foundation ja GNU-projekti ottivat käyttöönsä GNU General Public Licensen, joka oli ensimmäinen merkittävä vapaan ohjelmiston lisenssi [5, s. 205].

Vapaa ohjelmisto ja avoimen lähdekoodin ohjelmisto eivät eroa suuresti toisistaan, vaan niiden suurimmaksi eroksi voidaan nostaa vapaan ohjelmiston taustalla oleva eettinen lataus [5, s. 148] ja hieman tiukemmat kriteerit lisenssille [12]. Eettisyys vapaan ohjelmiston taustalla tarkoittaa Free Software Foundationin pyrkimyksiä kohti sananvapauden kaltaisia oikeuksia ohjelmistoille ja taistelua suljettua lähdekoodia vastaan. [13.]

Open Source Initiative perustettiin vuonna 1998, kun GNU/Linux alkoi saada valtavirran suosiota ja Netscape julkaisi verkkoselaimensa lähdekoodin avoimen lähdekoodin lisenssin alla [5, s. 148–149]. Syitä toisen lähes vastaavanlaisen termin luomiselle ei Open Source Initiative anna suoraan. Richard Stallman arvioi artikkelissaan Open Source Initiativen kehittäneen termin avoin lähdekoodi, jotta vapaa ohjelmisto erottuisi ohjelmiston eettisestä taustasta ja koska englannin kielen sana ”free” mielletään ilmaiseksi eikä vapaaksi. Stallmanin mukaan näitä välttämällä avoin lähdekoodi saisi yritysmaailman tuen. [14].

Iso osa lisensseistä on sekä avoimen lähdekoodin että vapaan ohjelmiston kriteereiden mukaisia sekä Free Software Foundationin ja Open Source Initiativen hyväksymiä.

Esimerkiksi GNU General Public License ja Apache License ovat molempien hyväksymiä, kun taas Nasa Open Source Agreement on vain Open Source Initiativen hyväksymä. [15][16.]

2.2 Testiautomaatio

Testiautomaatio on ohjelmistotestauksen muoto, jossa erillinen automatisoitu ohjelmisto suorittaa testejä halutulle ohjelmistolle/järjestelmälle (System Under Test, SUT). Sen avulla pyritään poistamaan toistuvasti suorittavien testien ajamista käsin ja vapauttamaan resursseja muualle. Automaation kehittämiseen kuluvasta suuresta aikamäärästä johtuen, sillä ei voida korvata manuaalitestaajia. Kertaluontoisia tai pari kertaa käytettyjä testejä ei ole kannattavaa automatisoida. On huomioitava, että testiautomaatio ei ole kirjaimellisesti testaus- vaan laadunvalvontajärjestelmä. [17, s. 76–78.]

Ohjelmistotestaamiselle on kasvava tarve, koska maailmanlaajuisesti valmistetaan jatkuvasti laajempia ohjelmistoja. Tämä näkyy testiautomaatiojärjestelmien määrässä sekä niiden huomioimisessa projekteissa. Cambridgen yliopiston tekemän tutkimuksen mukaan maailmassa käytetään vuosittaiseen ohjelmistovirheiden korjaamiseen 312 miljardia Yhdysvaltain dollaria [18]. Yksittäisten ohjelmistovirheiden aiheuttamat kustannukset saattavat kasvaa huomattavan suuriksi, kuten vuonna 2012, kun Knight Capital Groupin osakkeita ostava ja myyvä ohjelmisto alkoi toimia viallisesti, jonka seurauksena yhtiö hävisi 440 miljoonaa dollaria puolessa tunnissa [19].

Testiautomaation merkitys korostuu jo testattujen komponenttien testaamisessa. Tätä kutsutaan regressiotestaukseksi. Sen avulla pyritään varmistamaan olemassa olevien komponenttien toiminta. Regressiotestauksen tarve korostuu projekteissa, joissa käytetään jatkuvaa integrointia (Continuous Integration). Se on ohjelmistokehityksessä käytetty tapa, jossa tiimien jäsenet integroivat työtään useasti versionhallintaan. Erillinen automatisoitu ohjelmisto kääntää versionhallintaan integroidut tuotokset. Kerran päivässä tehty käännös tunnetaan daily buildina. [20.] Ajamalla testiautomaatio daily buildia vasten saadaan regressiotestattua jokaisen päivän integraation tulos. Tällöin saadaan jatkuvasti testattua uuden ja vanhan koodin yhteensopivuus.

Testiautomaatiojärjestelmiä on moniin eri tarpeisiin. Graafisten käyttöliittymien ja nettisivujen testauksessa niiden käyttö on yleistynyt. Tällaisissa järjestelmissä voidaan

nauhoittaa ihmisten käyttäytymistä ja suorittaa samat toiminnot kuin ihminen. Yksi tähän tarkoitukseen tarkoitetuista testiautomaatiojärjestelmistä on Selenium. [21.]

Testiautomaation kehittäminen vaatii runsaasti sille osoitettuja resursseja. Automatisoitujen testitapausten kehittäminen voi olla lähes yhtä haastavaa kuin kehitettävälle tuotteelle tehtävä ohjelmointityö. Lisäksi aikaa kuluu automaatiojärjestelmän toiminnallisuuden kehittämiseen ja testaukseen. Näistä syistä testiautomaation kustannukset ovat lähtötasoltaan korkeammat kuin käsin testauksessa. Jos testitapaus suoritetaan kuitenkin satoja tai tuhansia kertoja, jää suoritettavan testin hinnaksi huomattavasti pienempi arvo. [17, 76–79.]

3 HAASTATTELU

Haastattelu on laadullisen opinnäytetyön aineiston tärkein keräysmenetelmä. Haastattelun etuna on se, että siinä vastauksia voidaan tulkita enemmän kuin esimerkiksi kyselyssä. Haastattelu voidaan valita silloin, kun tutkimusaihetta on tutkittu vähän, tutkimustulokset halutaan sijoittaa laajempaan kontekstiin, aihe voi tuottaa monitahoisia vastauksia, halutaan selventää vastauksia, halutaan syventää tietoa tai tutkitaan arkoja tai haasteellisia aiheita. [22, s. 192.] Tässä opinnäytetyössä haastattelun perusteluna oli kerätä asiantuntijalausuntoja sekä selventää ja syventää niitä. Testiautomaatiosta on julkaistu kahdeksan ammattikorkeakoulutasoista opinnäytetyötä ja kuusi yliopistotasoista tutkimusta. Testiautomaatiosta löytyy asiantuntija-artikkeleita, kirjoja ja tutkimuksia. Siitä löytyy kirjallisuutta erityisesti maksullisten palveluiden kautta.

Haastattelutyyppejä on kolme: lomakehaastattelu eli strukturoitu haastattelu, avoin haastattelu ja teemahaastattelu. Lomakehaastattelussa käytetään esivalmistettuja lomaketta. Siihen on kirjattu esitysjärjestyksessä kysymykset ja väittämät. Avoin haastattelu on strukturoimaton haastattelumuoto, jolla ei ole kiinteää rakennetta. Tällöin haastattelu on luonteeltaan keskustelumuotoinen, jossa aihekin voi muuttua. Teemahaastattelu on kahden aiemman välimuoto, jossa haastatteluun kuuluvat teema-alueet ovat etukäteen suunniteltuja, mutta tarkka järjestys ja kysymysten asettelu tapahtuvat vasta haastattelun aikana. [22, s. 195–197.] Haastattelutyypiksi tähän opinnäytetyöhön valikoitui teemahaastattelu. Sen avulla haastattelutilanteet pystyttiin pitämään tiukasti asiassa, rajoittamatta kuitenkaan haastateltavilta saatavaa informaatiota. Lomakehaastattelun katsottiin rajoittavan haastattelutilannetta liiaksi, ja huonosti suunnitellut kysymykset olisivat voineet poistaa haastattelun mielekkyyden. Avoin haastattelu nähtiin liian laajaksi ja tarpeettomaksi tämän kaltaisessa tutkimuksessa.

Haastattelu on mahdollista toteuttaa joko yksilöhaastatteluna, parihaastatteluna tai ryhmähaastatteluna. Yleisin näistä kolmesta on yksilöhaastattelu. Valinta näiden kolmen vaihtoehdon kesken tulee tutkijan tehdä sen mukaan, miten hän arvioi saavansa parhaan lopputuloksen. [22, s. 197.] Tässä opinnäytetyössä haastattelut toteutettiin yksilöhaastatteluina. Sen avulla arvioitiin saavan parhaat tulokset, sillä jokainen haastattelu painotti haastateltavien eri osaamisalueita.

Haastateltavat valittiin siten, että mahdollisimman moniin testiautomaatioon liittyviin osa-alueisiin saataisiin mielipiteitä. Täten pääteltiin, että tarvitaan kolme haastateltavaa. Heistä yksi antoi mielipiteensä testitapausten kirjoittajan näkökulmasta, toinen ylläpitäjän näkökulmasta ja kolmas esimiehen näkökulmasta. Kaikki kolme haastateltavaa olivat vapaaehtoisesti mukana.

3.1 Ensimmäinen haastattelu

Ensimmäisellä haastateltavalla on kokemusta testitapausten kirjoittamisesta. Hänelle suunnatut kysymykset oli rajattu hänen osaamisaluettaan mukaileviksi.

Haastateltavan mukaan tärkeitä ominaisuuksia testiautomaatiossa ovat testitapausten kirjoittamiseen käytetyn ohjelmointikielen helppous sekä mahdollisimman suuri määrä lisäkirjastoja. Mahdollisimman helppo ohjelmointikieli nopeuttaa testien tekemistä ja niiden tulkitsemista sekä helpottaa oppimista. Suurella määrällä lisäkirjastoja on mahdollista säästää aikaa runsaasti ja saada ohjelma toimimaan vakaammin ja tehokkaammin.

Haastateltava ei osannut antaa mielipidettään ominaisuuksista, joita järjestelmän ei tulisi sisältää, koska hänellä ei ollut omasta mielestään tarpeeksi kokemusta testiautomaatiojärjestelmistä. Dokumentoinnin puutteet hän näki yleisiksi koko ohjelmistoalalla.

Aiemmin esille tulleen ohjelmointikielen helppouden ja nopeuden vuoksi haastateltavan mielestä testien ohjelmoimiseen käytettävä kieli tulisi olla skriptikieli, kuten Perl, Ruby tai Python. Haastateltavan mukaan ei ole ylitsepääsemätöntä, että käytettävä kieli olisi luotu kyseistä järjestelmää varten ja hieman eroaisi perusmuotoisista kielistä. Pääasia on, että kieli on helppo käyttää ja oppia. Ohjelmointikielen tärkeisiin perusominaisuuksiin haastateltavan mukaan kuuluvat mahdollisuudet hallita tietokantoja, käyttää useita yleisiä verkkoprotokollia, hyvät valmiudet tekstinkäsittelyyn sekä kaikenlainen mahdollinen lisälogiikka, kuten verkon yli ohjattavat pistorasiat. Virhetilanteiden simuloiminen on myös toivottu ominaisuus.

Ohjelmointia helpottavia työvälineitä haastateltavan mukaan ovat versionhallinta, debuggaustyökalut ja jonkinlainen versio testiautomaatiojärjestelmästä, johon ohjelmoija pääsee suoraan itse käsiksi. Tämä tarkoittaa sitä, että automatisoitujen testien kehittäjällä olisi mahdollisuus suoraan testata ohjelmiensa toimintaa ja saada niistä palautetta ilman välikäsiä.

Tämä nopeutta testiautomaation kehittämistä ja vähentää automaation ylläpitäjään kohdistuvaa kuormitusta.

Ajankäytön jakautumisen haastateltava arvioi jakautuvan niin, että monimutkaiset, erillistä logiikkaa vaativat testit vievät 95 % ohjelmointiin käytetystä ajasta. Loput 5 % kuluvat yksinkertaisiin avainsana-tyyppisiin testeihin. Kokonaistyöajasta saattaa iso osa kulua niin sanottuun turhaan työhön erillisten kirjastojen käytön ja asentamisen kanssa. Joskus saattaa pahimmillaan mennä jopa viikko tämänkaltaisen kamppailun kanssa.

3.2 Toinen haastattelu

Toinen haastateltava oli kokenut testiautomaation ylläpitäjä ja kehittäjä. Kysymykset keskittyivät testiautomaatioon ja sen ylläpitoon.

Testiautomaatiojärjestelmän tärkeimmät ominaisuudet ovat haastateltavan mukaan järjestelmän stabiilius ja käyttökelpoinen kehitysympäristö. Systeemin stabiilius vaikuttaa suoraan järjestelmän luotettavuuteen. Järjestelmä ei saa itse aiheuttaa virhetilanteita, koska se vaikeuttaa entisestään virheiden havaitsemista, kun joudutaan tutkimaan, johtuiko virhe automaatiosta vai itse koodista. Käyttökelpoinen ja automaatiota vastaava kehitysympäristö nopeuttaa ja helpottaa testitapausten luomista sekä niiden integroimista automaatioon. Tällöin myös testit ovat luotettavampia. Monipuolinen käyttöliittymä on kohtuullisen tärkeässä osassa, mahdollistaen testiautomaation tarjoamien ominaisuuksien kattavan käytön. Lisäksi laaja konfiguroitavuus parantaa järjestelmän käytettävyyttä sallimalla esimerkiksi erilaisia testikonfiguraatioita, testien ajastuksia ja eri alustojen käyttöä. Testiautomaatiossa käytetyt ohjelmointikielet tulisivat haastateltavan mukaan olla käyttöjärjestelmän valmiiksi tukemia.

Testiautomaatiojärjestelmän kuluttamat resurssit niin tietokoneessa itsessään kuin sähkönkulutuksen kannalta eivät haastateltavan mukaan ole erityisen relevantteja huolenaiheita, sillä testiautomaatiosta saatu hyöty on niin suuri, että se ajaa niiden ohi.

Haastateltava arvioi käyttävänsä viikkotyötunneistaan 20–25 % testiautomaatiojärjestelmän ylläpitoon. Järjestelmän kehittämiseen hän arvioi käyttävänsä saman verran aikaa. Loppuaika hänellä kuluu pääasiassa testitapausten integroimiseen järjestelmään. Oleellisin tekijä ylläpitoon kuluvan ajan vähentämiseen on haastateltavan mukaan automaation

komponenttien ja testattavan järjestelmän, Software Under Test (SUT), tarpeeksi kypsä taso. Automaation omille komponenteille saadaan kypsä taso regressiotestaamalla ne kunnolla ennen käyttöönottoa. Mikäli testattava järjestelmä on hajalla, joutuu ylläpitäjä jatkuvasti puuttumaan testeihin, jotta ne saadaan ajettua läpi.

Tärkeitä testiautomaation kanssa käytettyjä työkaluja ovat automaattikääntäjä, testimanageri ja jonkinlainen versionhallinta. Automaattisen kääntäjän tehtävänä on syöttää ohjelmia testiautomaation testattavaksi. Testimanagerin tulee pystyä rakentamaan erilaisia testisekvenssejä suoritettavista testeistä sekä tehdä niille omia testikonfiguraatioita. Lisäksi testimanagerissa testitulokset tulee välittyä helposti vaatimustyökaluun.

Ylläpitäjän näkökulmasta testiautomaatiojärjestelmässä tulee olla hyvät automaation toiminnan seuraamiseen käytettävät työkalut ja selkeä testitulosten raportointi. Automaation toiminnan seuraaminen on erittäin tärkeää, sillä automaation toimintavarmuus on pystyttävä todentamaan. Toimintaa tulee täten pystyä seuraamaan erittäin tarkalla tasolla, mieluiten kaikkea olisi mahdollista tutkia ja tarkkailla, jotta pienimmätkin mahdolliset virheet saadaan kiinni. Lisäksi tulosten sekä toiminnan seuraaminen tulisi olla mahdollista tehdä nopeasti, mieluiten yhdellä toiminnolla.

Haastateltava huomauttaa hyvin toimivan testiautomaatiojärjestelmän olevan puolet testiautomaation kokonaiskuvasta, loppuosan muodostuessa testitapauksista. Automaatiossa tulee tarkoin huomioida testitapaukset ja niiden toiminta, sillä huonot testitapaukset eivät välttämättä testaa mitään, vesittäen koko testiautomaation idean. Tämä myös tarkoittaa, että uusien testitapausten integroiminen järjestelmään on erittäin tärkeässä asemassa, ja ne tulee tarkastaa ja testata kunnolla ennen automaatioon laittamista. Haastateltava arvioi, että mikäli automaatio kokonaisuudessaan toimii hyvin, voitaisiin sitä käyttää regressiotestauksen lisäksi myös hyväksymistestauksessa.

3.3 Kolmas haastattelu

Kolmas haastateltava toimi esimiestehtävissä ja hänen yhtenä vastuualueena on testiautomaatiotiimin johtaminen. Haastattelun aiheet keskittyivät hänen osaamisalueeseensa.

Haastateltava arvioi testiautomaatiojärjestelmän tärkeimmiksi ominaisuuksiksi luotettavuuden, riittävän suorituskyvyn, helposti ymmärrettävän käyttäjärajapinnan,

joustavuuden ja monipuolisuuden. Luotettavuus tarkoittaa sitä, että virheet eivät johdu automaatiosta. Tämä on ensisijaisen tärkeätä. Helposti ymmärrettävä ja yksinkertainen käyttäjärajapinta helpottaa testiautomaation kanssa työskentelyä ja sen oppimista. Joustavuus ja monipuolisuus takaavat mahdollisuudet laajentaa järjestelmää jälkikäteen tarvittavaan suuntaan. Joustavuus on erittäin tärkeä osa testiautomaatiojärjestelmää.

Taloudellisuutta ajatellessa työtuntien näkökulmasta haastateltava arvioi tarpeellisen tarkastelun alkavan testiautomaatiojärjestelmän valintavaiheessa. Helppokäyttöinen ja helposti ylläpidettävä järjestelmä säästää huomattavan paljon aikaa isolta joukolta työntekijöitä. Joustava järjestelmä, jossa on helppoa lisätä toiminnallisuutta, mahdollistaa suuremman määrän automaation kehittämiseen osallistuvia henkilöitä. Mikäli toiminnallisuutta ei pystytä riittävän helposti kehittämään, tulee järjestelmään liittyvä osaaminen keskittää muutamalle henkilölle.

Testiautomaatiojärjestelmän valintaan tulee haastateltavan mukaan ottaa kohtuullisen suuri ryhmä ihmisiä. Ryhmässä tulee olla osaajia laajalta alueelta. Valintavaiheessa tulee saada kaikkien ryhmän jäsenten mielipiteet kuuluviin. Historian vaikutus valintaan tulee tasapainottaa, niin ettei kehitysaskelten eteen astuta sellaisista syistä kuin ”näin on ennenkin tehty”. Mikäli kuitenkin on valmiiksi laajasti osaamista jostakin asiasta, tulee tarkoin harkita, kumpi on parempi valinta.

Haastateltava ehdotti myös harkitsemaan jotakin pientä ja yksinkertaista testiautomaatiojärjestelmää, sillä iso ja monipuolinen ei välttämättä ole aina jokaiseen tilanteeseen paras. Järjestelmän kaikkien osien ei tarvitse olla hienoja ja automatisoituja, vaan painotus pitäisi olla toimintavarmuudessa ja muissa kriittisissä ominaisuuksissa, silläkin uhrauksella, että joitakin asioita joutuu tekemään käsin. Oli järjestelmä sitten iso tai pieni, olisi hyvä, jos järjestelmä olisi yleisesti käytössä ja todettu toimivaksi muidenkin toimesta.

3.4 Haastattelujen tulokset

Haastatteluista saatujen tietojen perusteella testiautomaation tärkeimmät ominaisuudet saatiin tiivistetysti jaettua kuuteen osaan. Niihin kuuluvat stabiilius, konfiguroitavuus, monitoroitavuus, ymmärrettävyys, testitapausten luontiin helppo kieli ja runsas määrä lisäkirjastoja. Taulukossa 1 on esitetty yhteenveto haastattelujen tuloksista, sisältäen aiemmin mainitut kuusi kriteeriä ja niiden sisällöt.

Taulukko 1. Yhteenveto haastattelujen tuloksista.

Haastattelujen tulokset	
Vaatimus	*Lisätietoja*
Stabiilius	Järjestelmän tulee olla vakaa. Syntyneet virheet eivät saa johtua automaatiosta. Synnyttää luotettavuutta.
Konfiguroitavuus	Parantaa järjestelmän käytettävyyttä ja mahdollistaa monipuolisemman käytön.
Monitoroitavuus	Automaation toiminnan seurattavuus ja selkeät testi-raportit parantavat käytettävyyttä.
Ymmärrettävyys	Selkeä käyttäjärajapinta mahdollistaa suuremman joukon käyttäjiä osallistumaan automaatioprosessiin.
Testitapausten luontiin helppo kieli	Skriptikieli tai muu vastaava helppo ohjelmointikieli testitapausten luomiseen. Helpottaa testitapausten tulkitsemista, opettelua ja kehittämistä.
Suuri määrä valmiita lisäkirjastoja	Mahdollisimman suurella määrällä lisäkirjastoja pyritään estämään riippuvaisuuksista johtuvat ongelmat. Lisää myös testitapausten luotettavuutta.

4 TESTIAUTOMAATIOJÄRJESTELMÄT

Erilaisia testiautomaatiojärjestelmiä löytyy useita ja eri käyttötarkoituksiin. Suuri osa niistä on suunniteltu erityisesti web-sovellusten ja verkkosivujen testaamiseen. Tähän käyttöön tarkoitetut järjestelmät jätettiin suoraan vertailun ulkopuolelle. Vaaditut kriteerit järjestelmälle olivat avoimen lähdekoodin lisäksi riittävä dokumentointi sekä mahdollisuus käyttää sitä sulautettujen järjestelmien testaukseen. Yleiskäyttöisiä, sulautettujen järjestelmien testaamisen mahdollistavia, avoimen lähdekoodin testiautomaatiojärjestelmiä löydettiin neljä kappaletta. Näistä yksi oli puutteellisesti dokumentoitu ja näin ollen jäi työstä pois. Näillä perusteluilla tässä työssä ovat vertailtavina seuraavat kolme testiautomaatiojärjestelmää: Robot Framework, Twister Framework ja Software Testing Automation Framework.

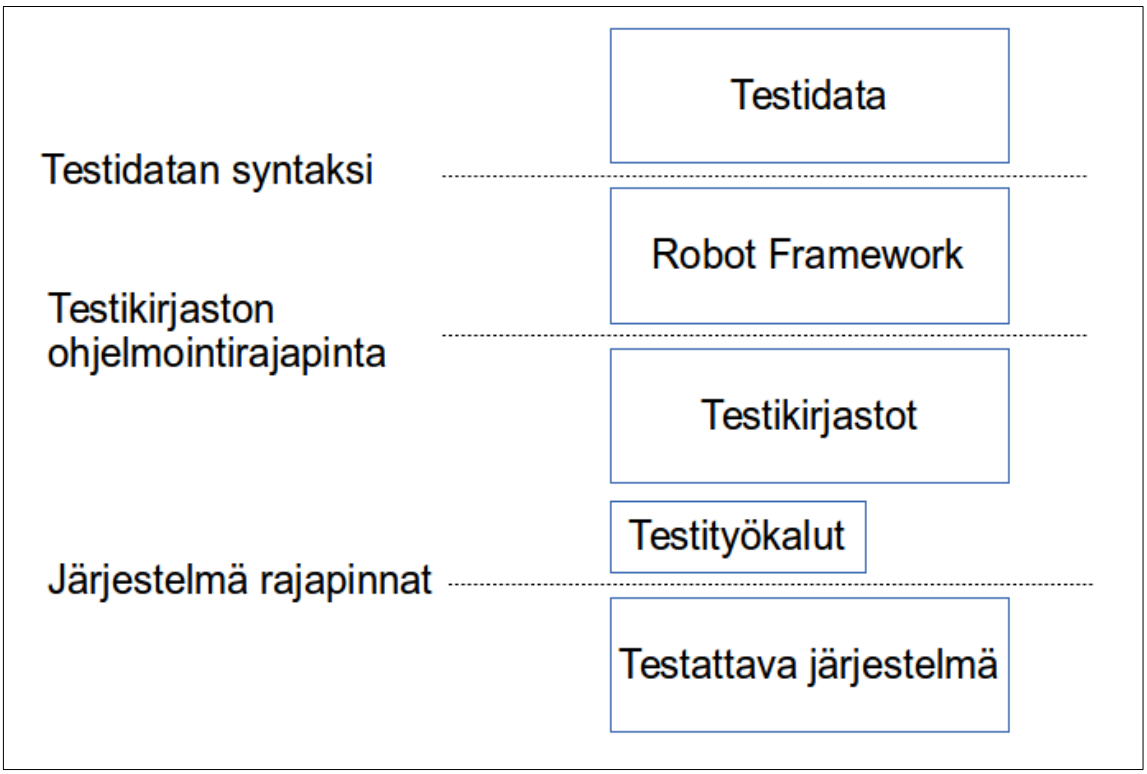
4.1 Robot Framework

Robot Framework on geneerinen hyväksymistestaamiseen suunniteltu testiautomaatio-ohjelmisto. Hyväksymistestauksessa ohjelman toimintaa verrataan käyttäjätason vaatimuksiin, eli testit suoritetaan käyttäjän näkökulmasta [23]. Se on suunniteltu avainsana-testaamiseen, mutta ohjelmiston testausominaisuudet ovat laajennettavissa. Se on toteutettu Python-ohjelmointikielellä. [24]

Robot Frameworkin perusteet syntyivät Pekka Klärckin (os. Laukkanen) ”Aineisto- ja avainsanaohjatut testiautomaatiojärjestelmät”-nimisessä pro gradu-tutkielmassa [24]. Klärck toimii edelleen ohjelmiston pääkehittäjänä [25]. Robot Frameworkin kehittäminen alkoi tuolloisen Nokia Networksin, nykyinen Nokia Solutions and Networks (NSN), sisäisenä projektina, joka myöhemmin julkaistiin avoimen lähdekoodin ohjelmistona. Se on vielä tänäkin päivänä NSN:n ylläpitämä ja rahoittama sekä projektin ydinhenkilöstö toimii NSN:n tiloissa Espoossa [26]. Robot Framework on lisensoitu Apache License 2.0:n alle [24].

Robot Frameworkissä testit suoritetaan testitapauksissa esitetyn datan sekä erillisten kirjastojen avulla, jotka toimivat rajapintana järjestelmän ja SUT:n välillä. Testidata voidaan esittää kolmessa eri syntaksissa, joista Robot Framework poimii käytettävät asiasanat. Niiden avulla Robot Framework etsii testikirjastosta käytettävät komennot, ja valinnanvaraisesti myös hyödyntää ulkoisia testityökaluja, joiden avulla se suorittaa halutut komennot

testattavassa järjestelmässä. Tämän jälkeen Robot Framework vertaa testidatassa esitettyjä odotettuja vastauksia saatuihin vastauksiin ja tuottaa HTML (HyperText Markup Language)- ja XML (Extensible Markup Language)-dokumentit tuloksista. HTML on kuvauskieli jota käytetään verkkosivujen rakenteen ja ulkoasun esittämiseen [27]. XML on World Wide Web Consortiumin (W3C) kehittämä strukturoitu tekstiformaatti [28]. Kuvassa 1 on esitetty kaaviokuva Robot Frameworkistä ja sen kanssa yhteistyössä toimivista osista.

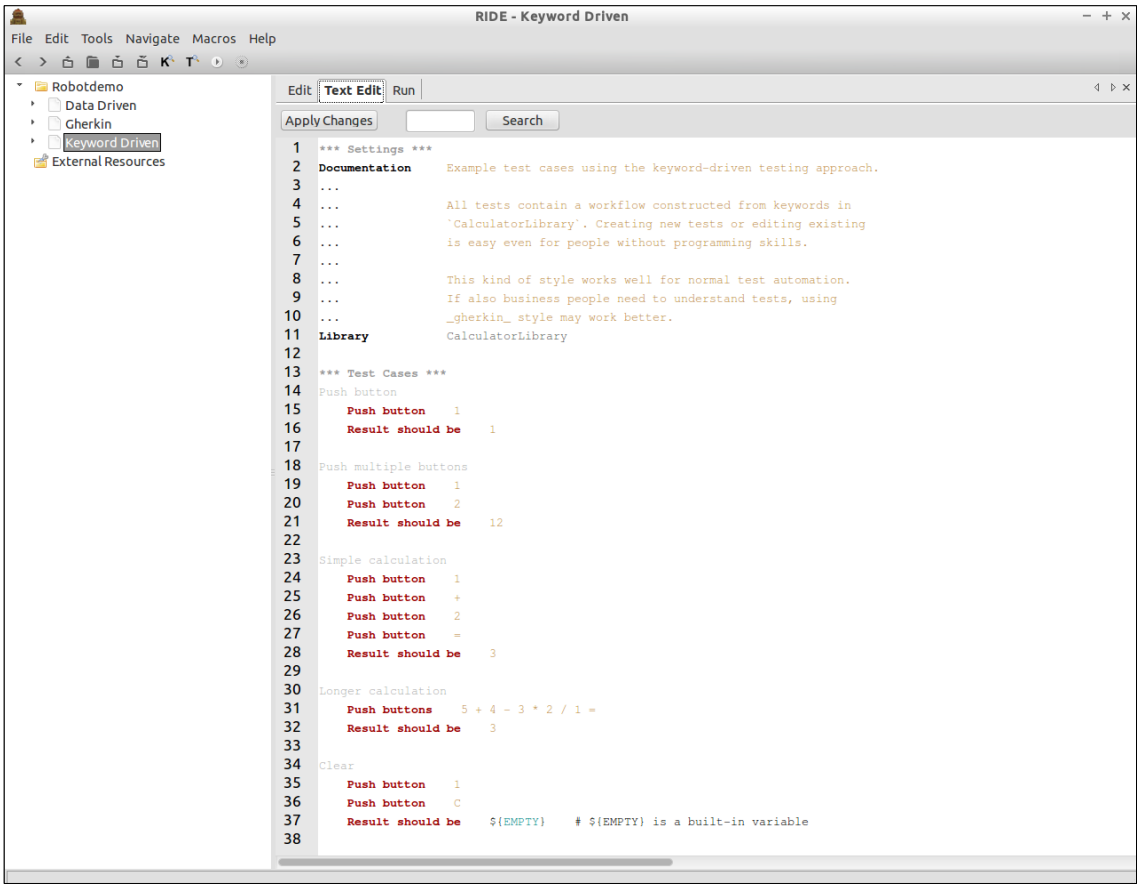


Kuva 1. Robot Framework ja sen ympärillä olevat moduulit.

Robot Frameworkiä on mahdollista laajentaa kirjastojen ja työkalujen avulla. Kirjastot on tarkoitettu antamaan lisätoimintoja testitapauksille. Niitä on mahdollista luoda Python- ja Java-ohjelmointikielillä. Valmiisiin ulkoisiin kirjastoihin kuuluu web-testauksen mahdollistava Selenium2Library ja verkkoliikenneprotokolla SSH:n (Secure Shell) hyödyntämisen mahdollistava SSHLibrary. Työkalut ovat erillisiä ohjelmia, joita voidaan käyttää tiettyjen toimintojen suoritukseen. Yksi valmiista ulkoisista työkaluista on DbBot, joka asettaa testitulokset SQLite-tietokantaan. Valmiita sisäisiä kirjastoja on kymmenen kappaletta. Yksi niistä on OperatingSystem-kirjasto, joka mahdollistaa käyttöjärjestelmäkomentojen suorittamisen. Remote on sisäinen kirjasto, joka mahdollistaa testien suorittamisen verkon yli järjestelmällä, johon Robot Frameworkiä ei ole asennettu. Testikirjastojen täytyy olla tällöin testattavalla järjestelmällä. [24.]

Käytettäessä kirjastoja ja työkaluja jotka on toteutettu Javalla, tulee Robot Framework suorittaa käyttäen Jythonia [24]. Se on Python-ohjelmointikielestä toteutettu JVM:llä (Java Virtual Machine) suoritettava versio, joka on kirjoitettu Javalla. Perinteisesti Pythonista puhuttaessa tarkoitetaan C-kielellä toteutettua CPythonia. Python-ohjelmointikieli on sama niin Jythonissa kuin CPythonissa. [29]

Robot Frameworkin käyttämiä testitapauksia voidaan tehdä ja editoida tekstinkäsittelyohjelmilla. Mikäli halutaan helpottaa testitapausten tekemistä, voidaan käyttää Robot Frameworkin omaa Robot Integrated Development Environment (RIDE) -ohjelmointiympäristöä. Vaihtoehtoisesti mikäli RIDE:ä ei haluta käyttää, on Eclipse-ohjelmointiympäristöön sekä Vim-, Emacs-, TextMate- ja Sublime Text -tekstinkäsittelyohjelmiin ladattavissa lisäosa Robot Framework -testien tekemiseen. Nämä ohjelmat ja lisäosat helpottavat testitapausten luontia tarjoamalla koodin automaattista täydennystä, syntaksin mukaista väritystä ja monia muita ominaisuuksia. [24.] Kuvassa 2 on esitetty RIDE:n testitapausten muokkausikkuna.



Kuva 2. RIDE:n testitapausten muokkausikkuna.

Testitapauksien luontiin on mahdollista käyttää kolmea erilaista lähestymistapaa: avainsanaohjattu (keyword-driven), dataohjattu (data-driven) tai Gherkin-syntaksi [24]. Avainsanaohjatussa syntaksissa testitapaus muodostuu sille annetusta nimestä ja halutuista avainsanoista, joille on annettu testisyöte. Yksinkertainen laskimen testaaminen voisi toimia seuraavalla koodilla:

```
Simple calculation
    Push button    1
    Push button    +
    Push button    2
    Push button    =
    Result should be    3 [30].
```

Tässä koodissa Simple calculation tarkoittaa testin nimeä. Push button on avainsana, jonka toiminnallisuus löytyy testikirjastosta. Tässä esimerkissä Push buttonin toiminnallisuutena on simuloida määritetyn napin painamista, eli tässä tapauksessa nappien 1, +, 2 ja = painamista. Result should be on myös avainsana ja sen syötteenä on vastaus, joka tulisi saada Push button -avainsanalle annetuista syötteistä.

Dataohjatussa syntaksissa syötteet annetaan erillisessä osassa avainsanoista. Laskimen testaaminen dataohjatulla syntaksilla voisi sisältää seuraavanlaista koodia:

```
*** Test Cases ***    Expression    Expected
Addition                    12 + 2 + 2    16
                    2 + -3          -1
*** Keywords ***
Calculate
    [Arguments]    ${expression}    ${expected}
    Push buttons    C${expression}=
    Result should be    ${expected} [31].
```

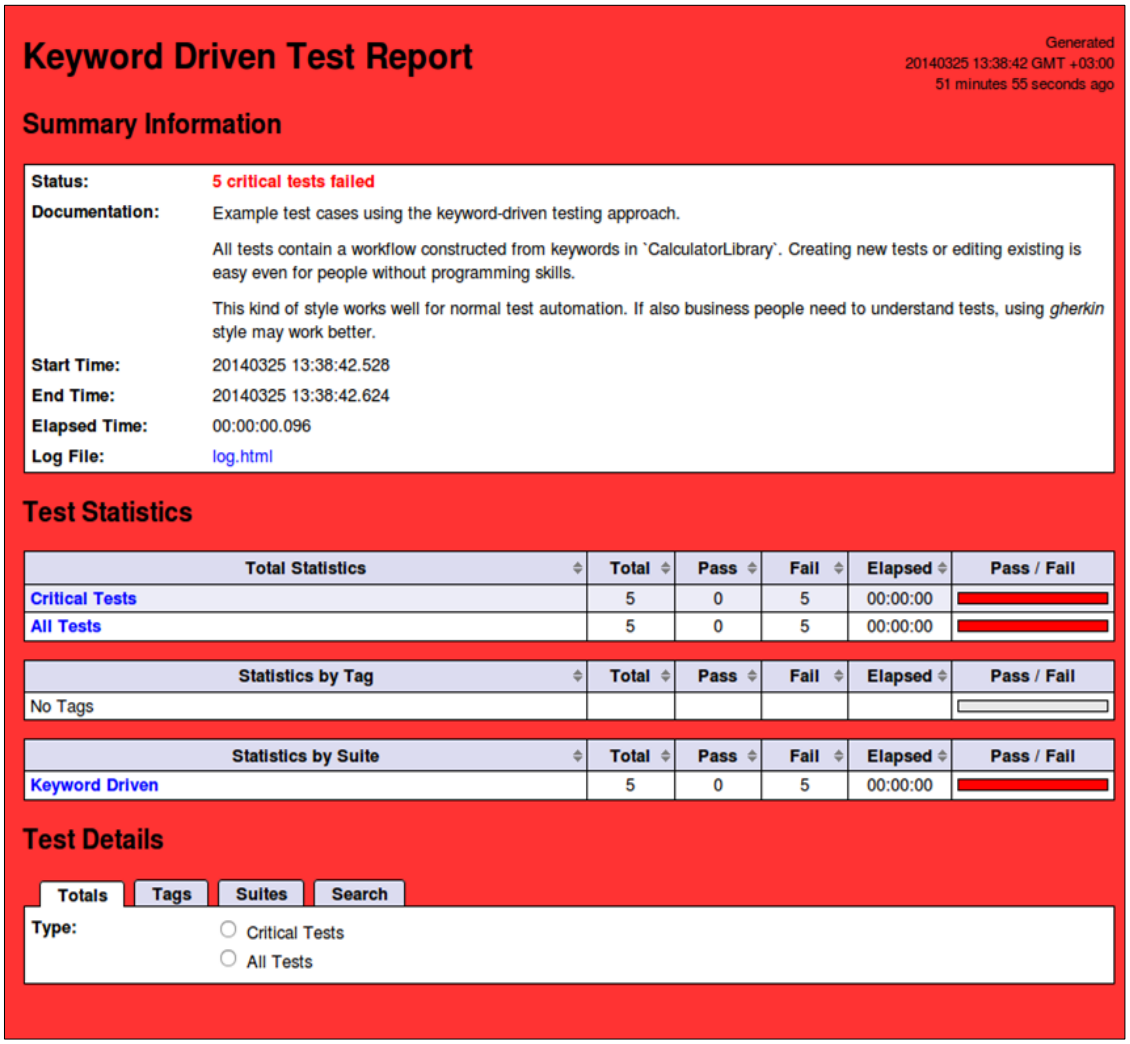
Yllä olevassa esimerkissä ensimmäisenä ilmoitetaan testitapauksen nimi, syötteet ja odotettu vastaus. Tämän jälkeen avainsana osiossa on määritetty, mitä testitapaukselle tehdään. Tässä tapauksessa Addition-testitapauksen ensimmäisenä osana painetaan nappeja 1, 2, +, 2, + ja 2, jolloin tuloksen tulisi olla 16.

Gherkin-syntaksi on luotu Cucumber-nimistä käyttäytymislähtöistä ohjelmistonkehitysympäristöä (Behavior Driven Development Framwork) ja testityökalua varten [32]. Gherkin-syntaksi on suunniteltu niin selväksi ja helpoksi, että ohjelmointia osaamattomatkin osaavat sitä lukea ja käyttää [33]. Gherkin-syntaksin mukainen Robot Frameworkin käyttämä testitapaus laskimen testaamiseen voisi olla seuraavan kaltainen:

```
*** Test Cases ***
Addition
    Given calculator has been cleared
    When user types "1 + 1"
    and user pushes equals
    Then result is "2"
*** Keywords ***
Calculator has been cleared
    Push button    C
User types "${expression}"
    Push buttons    ${expression}
User pushes equals
    Push button    =
Result is "${result}"
    Result should be    ${result} [34].
```

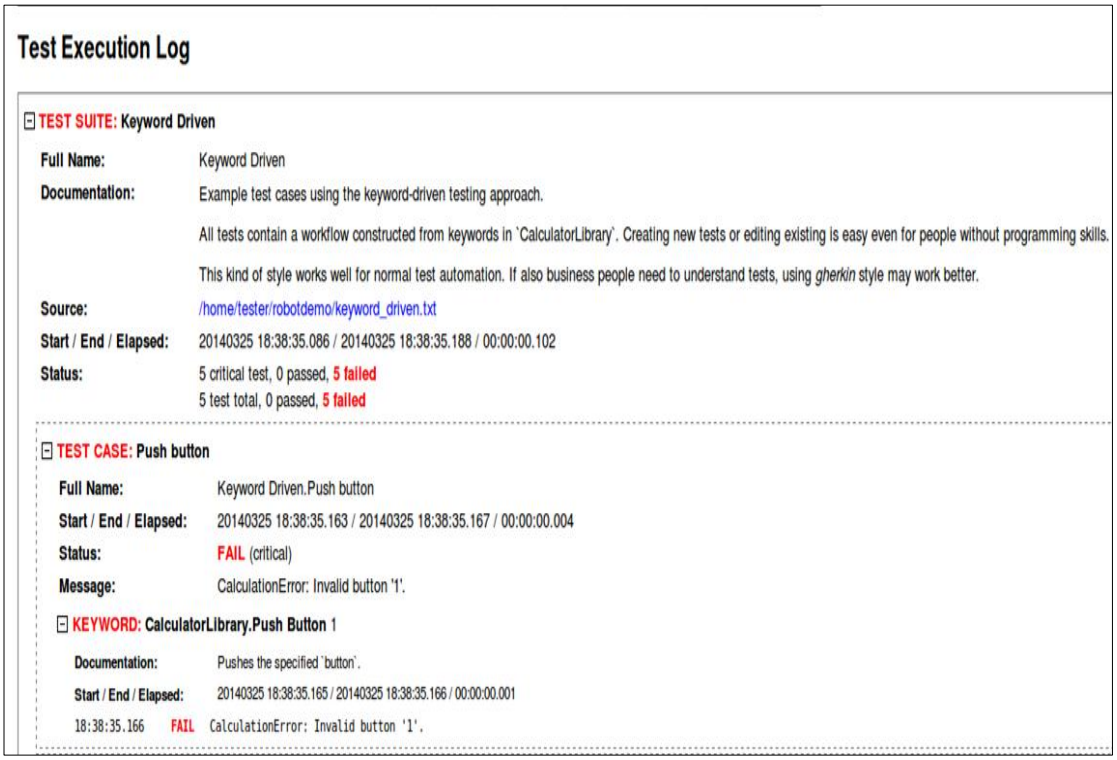
Test Cases -kohdan alla on määritetty Addition-niminen testitapaus, joka on kirjoitettu Gherkin-syntaksin mukaisesti sisältäen sen käyttämät Given-, When-, and- ja Then-avainsanat. Keywords-osiossa Robot Frameworkin ja testikirjaston avainsanat yhdistetään yllä olevien Gherkin-syntaksin mukaisten lauseiden kanssa toimivaksi.

Robot Framework tuottaa testeistä automaattisesti HTML- ja XML-dokumentit, jotka on mahdollista laittaa palvelimelle nähtäväksi. Testitulokset raportoidaan kahdessa osassa: Testiraportissa ja testilokissa. Testiraportissa annetaan suoritetuista testeistä yleiskuva, jossa näkyy testien lukumäärä, aloitus- ja lopetusaika sekä kuinka moni testeistä on mennyt läpi. Kuvassa 3 on valmiista esimerkistä saatu HTML-muotoinen testiraportti.



Kuva 3. Robot Frameworkin tuottama testiraportti.

Testilokissa on sama statistiikkaosio kuin testiraportissa, mutta sen lisäksi siitä löytyy myös testiajoloki. Tässä lokissa tulokset ovat tarkasteltavissa testikirjasto-, testitapaus-, kuin myös yksittäisellä asiasanatasolla, jolloin mahdollisten virheiden paikantaminen on selkeämpää. Kuvassa 4 on esitetty testiajoloki samasta testistä kuin kuvassa 3.



Kuva 4. Robot Frameworkin tuottama testiajoloki

Robot Framework on siis yleiskäyttöinen hyväksymistestaamiseen ja hyväksymistestauslähtöiseen kehitykseen suunniteltu testiautomaatiojärjestelmä. Sen vahvuudet ovat testitapausten luonnissa ja testiraporteissa. Laajentaminen on mahdollista lisäosilla ja kirjastoilla, joita on valmiina runsas määrä. Taulukossa 2 on esitetty yhteenveto Robot Frameworkistä.

Taulukko 2. Yhteenveto Robot Frameworkistä.

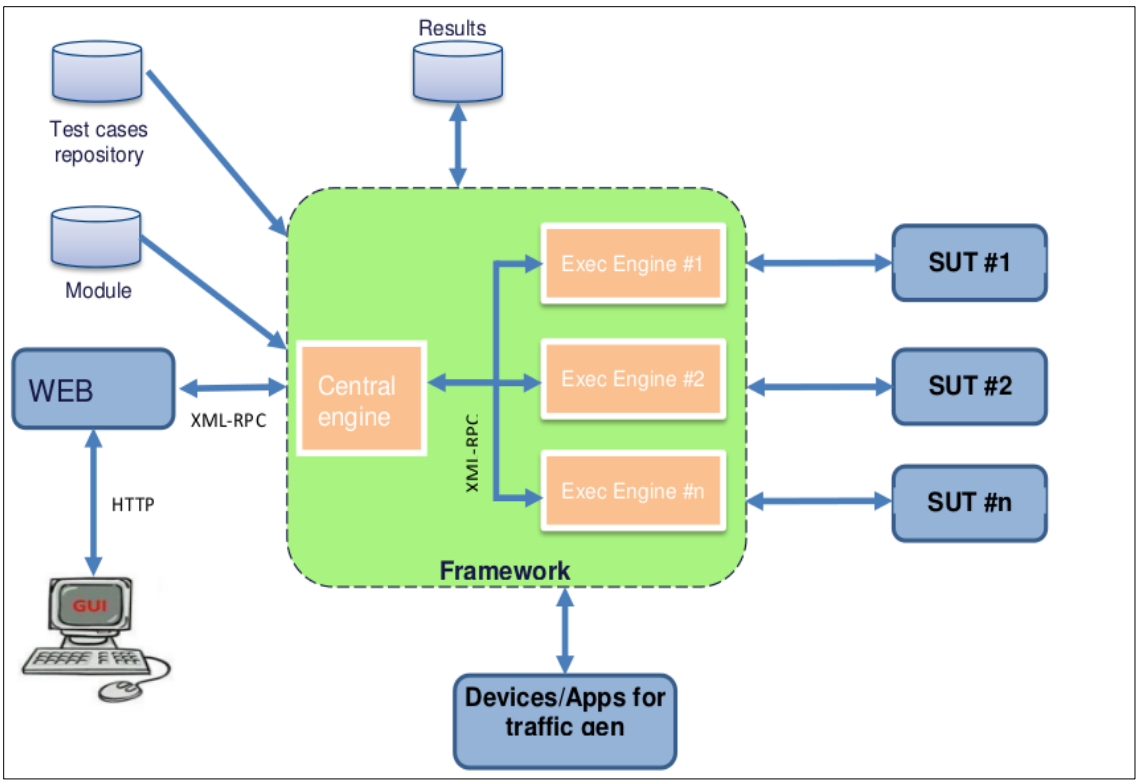
Robot Framework	
Yleiskuvaus	Geneerinen hyväksymistetaukseen suunniteltu järjestelmä.
Lisenssi	Apache License 2.0
Kirjoitettu	Pythonilla
Testitapaukset	Kirjoitetaan yhdellä kolmesta erikoissyntaksista, jotka hyödyntävät asiasanoja.
Ylläpidettävyys	Ei ylläpidettävyyttä lisääviä ominaisuuksia.
Testiraportit	Generoituvat automaattisesti XML- ja HTML-muodossa.
Laajennettavuus	Pythonilla ja Javalla on luotavissa kirjastoja ja lisäosia. Valmiina ovat mm: - DbBot-lisäosa testitulosten siirtämiseksi tietokantaan. - Lisäosa jatkuvan integraation Jenkins-työkaluun. - SSH-verkkoprotokollan hyödyntämisen mahdollistava SSHLibrary-kirjasto - Web-testauksen mahdollistava Selenium2Library
Muuta	Sisäisen RemoteLibraryn avulla on mahdollista kommunikoida toisten laitteiden XML-RPC -protokollan avulla. Tällöin Robot Frameworkin ei tarvitse olla asennettuna kohdelaitteelle.

4.2 Twister Framework

Twister Framework on venäläisen Luxoftin kehittämä testiautomaatiojärjestelmä [35]. Luxoft on Moskovassa vuonna 2000 perustettu ohjelmistoyritys, joka on venäläisen IT-palveluyhtiö IBS Groupin tytäryhtiö [36]. Twister Framework on lisensoitu Apache License 2.0:n alle. Se on kehitetty Python- ja Java-ohjelmontikielillä. Se toimii vain Linux-alustalla. Testitapausten kirjoittaminen onnistuu Python-, Tcl- ja Perl-ohjelmointikielillä. [37.]

Twister Framework on ensisijaisesti suunniteltu useiden verkkoliikennetekniikkaa käyttävien laitteistojen rinnakkaiseen testaukseen, ja ei ole näin ollen tarkoitettu esimerkiksi perinteisten tekstinkäsittelyohjelmien testaamiseen. Tästä huolimatta Twister Frameworkin avulla on mahdollista tehdä graafiselle käyttöliittymälle tarkoitettuja esinauhoitettuja testejä Selenium-, Sikuli- ja TestComplete-ohjelmistojen avulla. [37][38.]

Twister Framework rakentuu käyttäjän näkökulmasta graafisesta käyttöliittymästä ja/tai komentorivikäyttöliittymästä ja keskusmoottorista (Central Engine). Graafiseksi käyttöliittymäksi voidaan valita joko puhdas selainsovellus tai selaimessa toimiva Java-sovelma (Java applet). Keskusmoottori on kokoelma palveluita jotka muodostavat Twister-palvelimen. Näihin palveluihin kuuluvat suoritusmoottorin/-moottoreiden (Execution Engine) kanssa keskustelu, sähköpostin lähetys, tietokantaan talletukset ja liitännäisten suorittaminen. Suoritusmoottori on skripti, joka käynnistää suorittajan (Runner) jonka päätehtävänä on suorittaa testitapaus. Ne ajetaan ennakkoon valitulla SUT:lla. [37.] Kuvassa 5 on nähtävissä Twister Frameworkin korkean tason rakenne.



Kuva 5. Twister Frameworkin korkean tason rakenne [37].

Twister Framework vaatii toimiakseen palvelinkoneen, josta löytyy Linux-käyttöjärjestelmä, Apache HTTP-palvelinohjelma, MySQL-tietokanta ja Python-ohjelmointikieli. Mikäli käyttöliittymäksi halutaan Java-sovelmaa käyttävä näkymä, tulee palvelinkoneelle asentaa myös Java 1.7. Lisäksi Twister-palvelin vaatii runsaan määrän Python-ohjelmia toimiakseen, joihin kuuluu muun muassa PyCrypto-kryptografiatyökalu ja Paramiko SSHv2-protokollakirjasto. [37.]

Twister Framework on usean käyttäjän järjestelmä, jossa jokaisella käyttäjällä on ryhmiä ja rooleja. Usean käyttäjän järjestelmä mahdollistaa käyttäjille mahdollisuuden suorittaa valitsemiaan testejä silloin kuin haluavat. Keskusmoottori pitää huolta, että jokainen käyttäjä saa yhden tai useamman suoritusmoottorin omaan käyttöönsä. Tämä tarkoittaa, että keskusmoottori on kaikille käyttäjille sama, mutta suoritusmoottorit ovat jokaisella erillisiä. [37.]

Käyttöliittymävaihtoehtoja on kolme kappaletta: web-käyttöliittymä, Java-sovelma ja komentorivinäkymä. Web-käyttöliittymästä nähdään keskusmoottoriin liittyviä tietoja, kuten sen tila ja loki, voidaan hallinnoida prosesseja ja tarkastella käyttäjälokeja. Komentorivikäyttöliittymä on tarkoitettu testien suorittamiseen. Se näyttää testitapausten tilat ja mahdollistaa ajonaikaisesti testien lisäämisen testikokoelman perään. Java-sovelma on

monipuolinen selaimessa käytettävä käyttöliittymä, jossa on mahdollista seurata ja muokata kaikkia automatisointiin ja sen ympärille liittyviä ominaisuuksia. Siitä on mahdollista muokata SUT:eja, testikokoelmia ja testikohtaisia asetuksia. Java-sovelmassa ja web-käyttöliittymässä testitulokset ovat nähtävissä samasta paikasta. [37.]

Testitapaukset Twister Frameworkissa toimivat normaaleilla Tcl-, Perl- ja Python-ohjelmointikielillä, käyttäen niiden standardisyntaksia. Perl-kieltä on kuitenkin osittain rajoitettu. Vaadittu Python-versio on 2.7, ja Tcl:llä vaadittu versio on 8.5. Testitapausten luonnin helpottamiseksi Twister Framework sisältää oletuksena Pythonilla kirjoitetut SSH-, Telnet-, FTP-, Threads- ja UnitTest-kirjastot. Järjestelmä lisää testitapauksiin automaattisesti kahdeksan apumuuttujaa ja 12–13 apufunktiota (yksi enemmän Tcl:lle kuin Pythonille ja Perlille). Nämä apumuuttujat ja -funktiot sisältävät ominaisuuksia, joiden avulla voidaan kirjoittaa viestejä erikoislokiin ja vaikuttaa resursseihin ja SUT:eihin. Niiden hyödyntäminen on kuitenkin vapaaehtoista. [37.]

Testitapausten suorittamista voidaan seurata reaaliaikaisesti sekä Java-sovelmassa että komentorivillä. Suoritettavien testitapausten ja -kokoelmien asetuksia voidaan muokata tarpeen mukaan. Testikokoelmissa muokattavia asioita ovat: käytettävät SUT:t, etu- ja jälkikäteen suoritettavat skriptit, testitapausten väliset viiveet ja muut vastaavat. Testitapaukselle on valittavissa suoritettava- (Runnable), vapaaehtoinen- (Optional), alustustiedosto- (Setup file) ja puhdistustiedostoasetus (Teardown file). Suoritettava-asetuksella merkattu testitapaus suoritetaan. Jos sitä ei valita, testitapaus siirtyy suoritusmoottorille, mutta sitä ei suoriteta. Vapaaehtoinen-asetuksella varustetut testitapaukset eivät epäonnistuessaan lopeta koko testikokoelman suorittamista. Alustustiedostoasetuksella merkityt testitapaukset suoritetaan aina, ja epäonnistuessaan lopettavat koko testikokoelman suorittamisen. Testitapaukset, joilla on puhdistustiedostoasetus suorittavat testikokoelman puhdistuksen. Ne suoritetaan, vaikka alustustiedostoasetuksella merkityn testin epäonnistumisen johdosta testikokoelman suoritus olisi loppunut. Asetus- ja puhdistustiedostoasetuksella merkittyjä testitapauksia ei tallenneta tietokantaan. [37.]

Testitulokset menevät suoraan MySQL-tietokantaan säilöttäväksi. Muoto, jossa tulokset tallennetaan, on mahdollista muokata asetuksista. Saadut tulokset ovat nähtävissä selaimessa olevan työkalun Results-osiossa. Lisäksi on mahdollista asettaa testitulosten automaattinen lähettäminen sähköpostin välityksellä halutuille ihmisille. [37.] Kuvassa 6 on esitetty

esimerkkitapaus Twister Frameworkin antamasta yksityiskohtaisesta testiraportista. Tässä raportissa nähdään versionumerointi, käännöksen numero, testikokoelman nimi, testitapauksen nimi, aloitusaika, testin kesto, testin tulos ja linkki tarkempiin testitapauskohtaisiin tietoihin.

of_version	build_number	suite_name	tc_name	start_date	duration	status	tc_log
1.0	31	Suite_1	basicEcho.py	2013-01-30T15:42:51.833870	3	PASS	Click for details
1.0	31	Suite_1	basicEchoWithData.py	2013-01-30T15:42:55.258789	7	PASS	Click for details
1.0	31	Suite_1	basicFlowMod.py	2013-01-30T15:43:03.277510	7	PASS	Click for details
1.0	31	Suite_1	basicFlowStatsGet.py	2013-01-30T15:43:11.238864	7	PASS	Click for details
1.0	31	Suite_1	basicPacketIn.py	2013-01-30T15:43:19.258403	11	FAIL	Click for details
1.0	31	Suite_1	basicPacketOut.py	2013-01-30T15:43:30.286554	6	PASS	Click for details
1.0	31	Suite_1	basicPortConfigMod.py	2013-01-30T15:43:36.390734	6	PASS	Click for details
1.0	31	Suite_1	basicSimpleDataPlane.py	2013-01-30T15:43:43.326679	10	PASS	Click for details
1.0	31	Suite_1	basicSimpleProtocol.py	2013-01-30T15:43:54.297225	4	PASS	Click for details
1.0	31	Suite_1	basicTableStatsGet.py	2013-01-30T15:43:59.261475	7	PASS	Click for details
of_version	build_number	suite_name	tc_name	start_date	duration	status	tc_log

Kuva 6. Esimerkki Twister Frameworkin luomasta yksityiskohtaisesta testiraportista [39].

Twister Frameworkin toimintaa voidaan laajentaa joko valmiilla tai itse tehdyillä lisäosilla. Twister Frameworkin mukana toimitetaan valmiit lisäosat seuraavien ulkoisten ohjelmien hyödyntämiseen: Git-, SVN- ja ClearCase-versionhallintaohjelmistot, Jenkins/Bamboo-jatkuvan integroinnin työkalu ja Jira-tehtävänhallintaohjelmisto. Lisäksi mukana tulee myös Twister Frameworkin sisäisten lisäosat verkkoliikenteen tarkkailuun (Packet sniffer), keskusmoottorin aikataulutukseen (Scheduler) ja palveluiden lokien tarkkailuun (Service console) [39]. Lisäosat tehdään Javalla, mutta joitakin lisämetodeja voidaan tarpeen vaatiessa tehdä Pythonilla [40].

Yhteenvetona Twister Framework on suunniteltu useiden verkkoliikennetekniikoita hyödyntävien laitteiden, mahdollisesti samanaikaiseen, testaukseen. Sen tarjoamat käyttöliittymät selkeyttävät järjestelmän ylläpitoa ja mahdollistavat testikonfiguraatioiden mielekkään muokkauksen. Testitapaukset tehdään yleisillä ohjelmointikielillä käyttäen niiden standardisyntaksia. Testiraportit ovat muokattavissa halutun kaltaisiksi. Järjestelmän toiminta on laajennettavissa lisäosien avulla, joita valmiiksi saatavilla kattava joukko. Taulukossa 3 on esitetty tiivistetty yhteenveto Twister Frameworkistä.

Taulukko 3. Yhteenveto Twister Frameworkistä

Twister Framework	
Yleiskuvaus	Telekommunikaatioalan yrityksille suunniteltu järjestelmä.
Lisenssi	Apache License 2.0
Kirjoitettu	Pythonilla ja Javalla (Java-sovelma)
Testitapaukset	Kirjoitetaan Pythonilla, Tcl:llä tai Perlillä (rajoitettu) perussyntaksia käyttäen.
Ylläpidettävyys	Keskusmoottoria ja testejä voidaan reaaliaikaisesti monitoroida.
Testiraportit	Muokattava halutunlaisiksi. Tarkasteltavissa web-sovelluksesta ja tietokannasta.
Laajennettavuus	Pythonilla ja Javalla on luotavissa lisäosia. Valmiina ovat mm: - Lisäosa Jira-tehtävänhallintaohjemistoon. - Lisäosat Git, SVN ja ClearCase-versionhallintaohjelmistoihin - Lisäosa jatkuvan integraation Jenkins-työkaluun. - Aikataulutukseen Scheduler-lisäosa. - Verkkoliikenteen tarkkailuun Packet sniffer -lisäosa.
Muuta	Tarjoaa kolme käyttöliittymävaihtoehtoa. SUT- ja testikohtaiset säädöt tehty helposti muokattaviksi.

4.3 Software Testing Automation Framework

Software Testing Automation Framework (STAF) on IBM:n vuonna 1998 julkaisema testiautomaatiojärjestelmä [41]. Se on lisensoitu Eclipse Public License V1.0:n alle. STAF on suunniteltu kevyeksi, useita käyttöjärjestelmiä ja ohjelmointikieliä tukevaksi sekä helposti laajennettavaksi. STAF tukee 22:ta käyttöjärjestelmää, joihin kuuluu Linux, Windows 8, Mac OS X ja FreeBSD. Tuettuja ohjelmointikieliä ovat Java, C, C++, Python, Perl, Tcl ja Rexx. [42.]

STAF on rakennettu uudelleenkäytettävien komponenttien ympärille, joita kutsutaan palveluiksi (services). Sisäänrakennettuihin palveluihin kuuluvat lokiin kirjoittaminen, monitorointi ja resurssienhallinta. [43] Lisäksi valmiita ladattavissa olevia palveluita on 12, joihin kuuluvat HTTP-, FTP- ja FSExt-palvelut. HTTP-palvelu mahdollistaa testiraporttien julkaisemisen verkossa. FTP-palvelun avulla voidaan siirtää tiedostoja laitteelta toiselle. FSExt-palvelu mahdollistaa joidenkin tiedostojärjestelmäkomentojen käytön. [44]. Omien palveluiden kirjoittaminen on mahdollista Java-, C++- ja Perl-ohjelmointikielillä käyttäen STAF:in mukana toimitettavia kirjastoja [45].

STAFProc on tietokoneella, toisin sanoen STAF Clientillä, ajossa oleva prosessi, joka käsittelee pyynnöt ja ohjaa ne oikealla palvelulle. Nämä pyynnöt voivat tulla paikalliselta koneelta tai toiselta STAF Clientiltä. Näin ollen STAF toimii vertaisympäristössä, jossa koneet voivat tehdä pyyntöjä toisilla koneilla olevilta palveluilta. Palvelupyyntöjen

lähettäminen vaatii, että prosessi on hankkinut itselleen käsittelijän (handle). Se on prosessille uniikki tunnistetieto, jonka avulla STAF-ympäristössä jokainen prosessi pystytään tunnistamaan. Käsittelijä sisältää seuraavat tiedot: Käsittelijän (kuvaava) nimi, viimeisimmän pyynnön aikaleima, mahdollinen käyttäjän todennus ja prioriteettijonotusarvo prosessienväliselle viestinnälle. Ennen kuin prosessi sulkeutuu, tulee sen vapauttaa käsittelijänsä käyttämät resurssit ilmoittamalla STAF:lle käsittelijän perumisesta. [42.]

STAX (STAf eXecution engine) on XML:ään perustuva suoritusmoottori (execution engine), joka on toteutettu ulkoisena STAF-palveluna [46]. XML:ssä tieto esitetään elementeillä. Niiden avulla rakenne ja sisältö pysyvät tarkasti järjestettynä. Elementeillä on aloitus- ja lopetusmerkkaukset, joilla ilmoitetaan mitkä asiat kuuluvat minkäkin elementin sisään. Aloitus- ja lopetusmerkinnät kirjoitetaan hakasulkeiden sisään siten, että aloitusmerkinnän sisään asetetaan vain elementin nimi ja lopetusmerkinnän asetetaan elementin nimen eteen vinoviiva. Esimerkiksi seuraava on hyväksyttävää XML:ää: [tervehdys]Hei![/tervehdys]. Tässä esimerkissä ”[tervehdys]” on tervehdys-elementin alkumerkintä, ”[/tervehdys]” saman elementin loppumerkintä ja ”Hei!” on kyseisen elementin sisältö. [28.]

Mikäli sisältöä halutaan muokata jollakin tavalla, löydetään se viittaamalla sisällön elementtiin. STAX on suunniteltu helpottamaan testien ja testiympäristöjen automatisointia. STAX ottaa vastaan tehtävämäärityksiä XML-dokumentteina. Nämä tehtävämääritykset mahdollistavat tehtävän suorittamiseen tarvittavien prosessejen ja STAF-komentojen määrittämisen. STAX sisältää toimintoja joiden avulla voidaan toteuttaa, hallinnoida, seurata ja tarkastella tehtäviä. [46.]

STAX käyttää muuttujien ja lausekkeiden arviointiin Python-ohjelmointikieltä, jota voidaan suoraan lisätä käytettyihin XML-dokumentteihin. Python-koodi suoritetaan Jythonilla. Pythonia käyttämällä STAX saa käyttöön tehokkaat työkalut joita ei XML:ssä ole otettu käyttöön. Näiden Python-kielen ominaisuuksien avulla voidaan esimerkiksi etsiä tiettyä merkkijonoa tiedostosta. Lisäksi STAX:iin on lisätty valmiita XML-elementtejä, joiden avulla voidaan XML-dokumentissa toteuttaa esimerkiksi funktioita, silmukkarakenteita sekä poikkeusten ja signaalin käsittelyä. Seuraava STAX esimerkkikoodi sisältää Pythonin ja STAX:in oman XML-elementin käyttöä: [46.]

```
<script>machList = []</script>

<script>clientPool = 'ClientMachinePool'</script>

<loop var="i" from="1" to="10">

 <sequence>

  <stafcmd>

   <location>'server1.austin.ibm.com'</location>

   <service>'RESPOOL'</service>

   <request>'REQUEST POOL %s' % (clientPool)</request>

  </stafcmd>

 <script>machList.append(STAFResult)</script>

 </sequence>

</loop>
```

[46.]

Yllä olevassa STAX-koodissa jokainen elementti on esimääritelty. Script-elementtien sisälle asetetaan Python-koodia. Loop-elementti on silmukkarakenteen luova rakenne, joka tässä esimerkissä suorittaa sisällä olevan koodin kymmenen kertaa. Sequence-elementin sisällä olevat STAX-elementit suoritetaan sekvenssissä, eli peräkkäin, kirjoitetussa järjestyksessä. Stafcmd-elementin sisälle asetetaan STAF-palvelupyyntö ja pyynnön kohde. Location-elementin sisään asetetaan STAF-pyynnön kohde, service-elementin sisällä palvelun nimi ja request-elementin sisälle itse palvelulle esitettävä pyyntö. Koodissa kirjoitetaan machList-listaan kymmenen tietokoneen nimeä, joilta voidaan tehdä STAF-pyyntöjä. [46.]

Testitapauksia voidaan tehdä useilla eri tavoilla ja ohjelmointikielillä. Käytettävissä olevat ohjelmointikielet ovat: C++, Java, Perl, Python ja Tcl. On myös mahdollista käyttää Ant- ja Shell-skriptejä. Lisäksi testitapaukset on myös mahdollista toteuttaa STAX:in oman XML-pohjaisen syntaksin avulla. Kaikkien näiden kielien kanssa yhteistä on, että paras ja selkein testijärjestelmä saadaan käyttämällä STAF/STAX:in mukana tulevia valmiita kirjastoja. Näiden kirjastojen avulla testitapaukset ovat yhteydessä koko STAF/STAX-ympäristön

kanssa. Tällöin esimerkiksi käyttämällä monitorointi- ja lokipalveluita löytyy verkosta tieto, mitä testitapaus parhaillaan tekee ja miten aiemmat testit ovat menneet. [45]

Testiraporttien muodostaminen tehdään kirjaamalla suoritettujen testien nimet ja niiden tila (hyväksytty/hylätty) omaan dokumenttiin. Eri palveluita käyttämällä voidaan esimerkiksi tarkkailla haluttujen tiedostojen olemassaoloa ja määritettyjen merkkijonojen esiintymistä tiedostoissa. Nämä palveluilta saadut tiedot voidaan lisätä testiraporttiin. Valmis testiraportti on mahdollista lähettää sähköpostina Email-palvelun avulla määritetyille henkilöille. HTTP-palvelun avulla on mahdollista julkaista testiraportti verkossa. [45.]

Osa Javalla toteutetuista palveluista kuten STAX sisältävät omat graafiset käyttöliittymät [46]. Eclipse-ohjelmointiympäristöön on myös ladattavissa lisäosa STAF:n käyttöön [47]. Lisäksi on myös saatavilla QuickSTAF-niminen suljetun lähdekoodin graafinen käyttöliittymä. Sen rajoittamaton versio on maksullinen. QuickSTAF mahdollistaa kaikkien komentorivillä tehtävien asioiden graafisen käytön. [48]

Yhteenvetona STAF on siis laajennettavaksi tarkoitettu vertaisympäristössä toimiva testiautomaatiojärjestelmä. Se on suunniteltu kevyeksi, tehokkaaksi ja monialustaiseksi. STAF on tarkoitettu pohjaksi, jonka päälle asetetaan haluttuja toimintoja palveluiden avulla. Automaation kannalta tärkein palvelu on STAX. Se STAF:ia varten suunniteltu suoritusmoottori ja on lähestulkoon pakollinen. Testiraportit ja niiden sisältö konfiguroidaan kokonaan itse. Taulukossa 4 on esitetty tiivistetty yhteenveto STAF:ista.

Taulukko 4. Yhteenveto STAF-järjestelmästä.

Software Testing Automation Framework (STAF)	
Yleiskuvaus	Vertaisympäristössä toimiva modulaarinen järjestelmä
Lisenssi	Apache License V1.0
Kirjoitettu	C++:lla ja Javalla
Testitapaukset	Kirjoitetaan C/C++:lla, Javalla, Perlillä, Pythonilla, Tcl:llä, Ant- tai Shell-skriptillä.
Ylläpidettävyys	Monitorointipalvelun avulla voidaan seurata järjestelmän toimintaa
Testiraportit	Vapaasti muokattavissa halutunlaisiksi. Ei valmista mallia.
Laajennettavuus	Järjestelmä on suunniteltu laajennettavaksi palveluiden avulla. Omia palveluita Voidaan luoda C++:lla, Perlillä ja Javalla. Ladattavissa olevia palveluita ovat mm: - STAX, testien automatisointiin käytettävä suoritusmoottori. - HTTP-palvelu, jonka avulla tulokset voidaan lähettää verkkosivulle. - FTP-palvelu, jolla voidaan siirtää tiedostoja - FSExt-palvelu, sisältää joitakin tiedostojärjestelmään liittyviä työkaluja
Muuta	Järjestelmä tarjoaa pohjan, josta voidaan palveluiden ja omien lisäyksien avulla Tehdä tarpeisiin sopiva testiautomaatiojärjestelmä. STAX-palvelu on käytännössä pakollinen tähän.

5 VERTAILU

Ohjelmistojen vertailu toteutettiin käyttämällä testeistä ja dokumentaatioista kerättyä aineistoa. Ohjelmistot testattiin siten, että ne asennettiin GNU/Linux-käyttöjärjestelmällä varustetulle tietokoneelle, jonka jälkeen ohjelmalla suoritettiin saatavilla olleita esimerkkitestitapauksia. Työn alussa määritetyistä kriteereistä taloudellisuus muodostuu haastatteluista saatujen tietojen mukaan ylläpidettävyydestä ja testitapausten luomisesta. Koska ylläpidettävyys on jo yksi vertailtavista aiheista, taloudellisuus on korvattu testitapausten luonti -aiheella. Vertailu on siis toteutettu seuraavien testiautomaatiojärjestelmiin liittyvien aiheiden ympärille: tekniset ominaisuudet, käytettävyys, testitapausten luonti, ylläpidettävyys ja käyttöönotto.

5.1 Tekniset ominaisuudet

Twister Framework ja STAF ovat suunniteltu lähtökohtaisesti testausympäristöön, jossa on useita käyttäjiä, tietokoneita ja testattavia systeemejä. Robot Framework pystyy käyttämään myös tämänkaltaista mallia Remote-kirjaston avulla.

Robot Framework ja STAF toimivat käytännössä kaikilla yleisimmillä ja muutamalla vähemmän käytetyllä käyttöjärjestelmällä. Twister Framework toimii käytännössä ainoastaan Linux-alustalla, vaikkakin joitakin testejä voidaan suorittaa Windowsilla portable-tilassa, jossa ohjelmistoa ei ole asennettu tietokoneelle.

Kaikki kolme järjestelmää on suunniteltu laajennettaviksi. Twister ja Robot Frameworkissa laajennukset tehdään pääosin lisäosien avulla. Robot Frameworkiin on saatavissa jatkuvaa integrointia varten Jenkins-lisäosa ja testitulosten tietokantaan siirtämistä varten DbBot-niminen työkalu. Twister Frameworkin mukana toimitetaan lisäosat esimerkiksi jatkuvaan integrointiin (Jenkins), versionhallintaan (Git, SVN, ClearCase), bugien seurantaan (Jira). STAF:in laajennettavuus perustuu sen palveluita käyttävään rakenteeseen. Saatavissa olevat palvelut mahdollistavat esimerkiksi sähköpostin ja ajastusten käytön. Myös STAF:ia varten on Jenkinsiin luotu lisäosa. Kaikkien kolmen ohjelmiston laajennettavuus on toki siltä kannalta automaattista, että niiden jokaisen lähdekoodi on vapaasti saatavilla ja muokattavissa.

5.2 Käytettävyys

Robot Frameworkin käyttöä ja testien kehittämistä varten on mahdollista ottaa käyttöön graafinen kehitysympäristö RIDE. Siinä testitapausten kehittämistä on helpotettu esimerkiksi syntaksin värityksellä. Testien suorittaminen onnistuu nappia painamalla ja suoritusasetuksia on mahdollista säätää eri valintapainikkeilla. Vaihtoehtoisesti RIDE:n tilalle voidaan ottaa Eclipse-ohjelmointiympäristö, johon on ladattavissa lisäosa Robot Frameworkille. Eclipse-lisäosa ei kuitenkaan tarjoa testin suorittamiseen minkäänlaisia toimintoja, vaan toimii lähinnä hienostuneena tekstieditorina.

Twister Framework tarjoaa mahdollisuuden käyttää kahta erilaista graafista käyttöliittymää. Vaihtoehdot ovat web-käyttöliittymä tai Java-sovelma. Web-käyttöliittymä on tarkoitettu keskusmoottorin tarkasteluun ja prosessien käynnistämiseen sekä sulkemiseen. Java-sovelma on monipuolinen käyttöliittymä, joka mahdollistaa lähestulkoon kaikkien Twister Frameworkistä löytyvien ominaisuuksien muokkaamisen ja tarkastelun. Se on kätevä työkalu testattavien systeemien hallinnoimiseen, suoritettavien testien valitsemiseen ja testien suorituksen tarkkailemiseen.

STAF:in mukana ei tarjota yleispätevää graafista käyttöliittymää. STAX -palvelun mukana on Javalla toteutettu graafinen käyttöliittymä, jolla voidaan hallinnoida tehtäviä ja testitapauksia sekä seurata testien suoriutumista. Vaihtoehtoisesti voidaan ottaa käyttöön Eclipseen luotu STAF-lisäosa. Siitä on mahdollista käyttää kaikkia STAF-komentoja. Lisäosa tarjoaa myös mahdollisuuden tarkkailla graafisesti käynnissä olevia palveluita ja pyyntöjä sekä mahdollistaa STAF-muuttujien tarkastelun ja muokkauksen.

Kaikkia kolmea järjestelmää on mahdollista käyttää komentoriviltä. STAF:in ja Robot Frameworkin tapauksessa se on kattavin käyttöliittymä. Twister Frameworkin Java-sovelma on kattavampi kuin sen komentorivikäyttöliittymä. Twister Frameworkissä komentoriviä ei ole tarkoitettu ylläpitotehtäviin vaan ainoastaan monitorointiin ja käyttäjätoimintoihin. Kaikissa kolmessa ohjelmassa komentorivin käyttö on dokumentoitu sen verran kattavasti, että ohjelmiston käytettävyys ei siitä kärsi. Komentorivi ehkä hidastaa ohjelmiston käytön oppimista, mutta pitkällä tähtäimellä on helppo ja tehokas tapa käyttää ohjelmistoa.

5.3 Testitapausten luonti

Twister Framework tukee testitapauksia, jotka on kirjoitettu Python-, Tcl- tai Perl-ohjelmonitikielellä. Perlin käytettävissä olevia ominaisuuksia on hieman rajoitettu perusversiosta. Pythonilla ja Tcl:llä testitapaukset voidaan tehdä käyttäen täysin standardisyntaksia. Mitään erillisiä testitapausten kehittämistä vaikeuttavia tai helpottavia asioita järjestelmä ei sisällä.

STAF:issa testitapaukset voidaan tehdä C++:lla, Javalla, Perlillä, Pythonilla, Tcl:llä, STAX-koodilla ja Ant- tai Shell-skriptauksella. Testitapaukset kirjoitetaan näiden kielien perussyntaksin mukaisesti samoin kuin Twister Frameworkissä. Jotta STAF-testeissä voidaan käyttää STAF-komentoja ja -palveluita, tulee testitapauksissa käyttää STAF:in mukana toimitettavia kirjastoja. Näistä kirjastoista löytyy valmiit käytettävissä olevat funktiot eri STAF:in osioiden kanssa kommunikointiin. Järjestelmä ei sisällä muita testitapauksen kehittämiseen vaikuttavia asioita.

Testitapausten kehittäminen Robot Frameworkille eroaa paljon STAF:ille ja Twister Frameworkille kehittämisestä. Robot Frameworkille testitapauksia voidaan tehdä kolmella erilaisella syntaksilla, joista kaksi on luotu varta vasten Robot Frameworkiä varten. Näiden syntaksien on tarkoitus helpottaa testitapausten kirjoittamista ja parantaa niiden luettavuutta. Testitapausten luettavuus on pyritty saamaan sellaiselle tasolle, jolla testitapauksen toiminta selviää ohjelmointitaidottomalle henkilölle. Tämä syntaksin helppous ei kuitenkaan tarkoita sitä, että testitapausten luominen olisi kokonaisuudessaan helpompaa Robot Frameworkille kuin STAF:ille tai Twister Frameworkille. Robot Frameworkille on luotava testitapausten ja testattavan ohjelmiston välille testikirjasto. Siinä on määritetty menetelmät, joilla testitapausten käyttämät avainsanat kommunikoivat testattavan ohjelmiston eri osien kanssa. Testikirjastot voidaan kirjoittaa joko Pythonilla tai Javalla. Niiden tekeminen on verrattavissa STAF:lle ja Twister Frameworkille tehtävien testitapausten kehittämisen kanssa. Testitapausten ja testikirjastojen luomista on mahdollista helpottaa ladattavilla kirjastoilla.

5.4 Ylläpidettävyys

Määritellään ylläpidettävyyteen vaikuttavaksi osa-alueiksi järjestelmän toimintavarmuus, toimivuuden tarkastelu ja testien raportointi.

Twister Frameworkissä järjestelmän toimivuutta ja testien suorittamista voidaan seurata web-käyttöliittymässä, Java-sovelmassa ja komentorivillä. Web-käyttöliittymän puolella voidaan seurata keskusmoottorin ja käyttäjien toimintaa lokien avulla sekä tarkastella testiraportteja. Niiden ulkoasu ja siihen liitetyt tiedot riippuvat siitä, miten raportointi on konfiguroitu. Testiraporteista on mahdollista tuottaa Excelissä avattavia XLS-tiedostoja. Valmistaja Luxoft on testannut järjestelmän (Central Enginen ja raportoinnin) toimintavarmuutta 750 samanaikaisella yhteydellä, jolloin ei tapahtunut yhtään kaatumista tai yhteyden katkeamista [34].

Robot Framework ei sisällä erillisiä monitorointityökaluja, koska se on käytännössä pelkästään testejä suorittava ohjelma. Testejä monitoroidaan tällöin testitapauksiin, testikirjastoihin ja erillisiin skripteihin lisättyjen tulostus ja lokiinkirjoitusfunktioiden avulla. Näiden funktioiden tuloksia voidaan seurata komentoriviltä tai erillisistä lokitiedostoista. Toisin sanoen Robot Frameworkin monitoroitavuus riippuu siitä, minkälaisia tulostuksia testit ja niitä käynnistävät skriptit tekevät. Testituloksia voidaan tarkastella kahdessa eri muodossa: XML-tiedostona ja HTML-tiedostoina. XML-tiedostosta löytyy kaikki testeistä saadut tulosteet. Siitä luodaan kaksi HTML-dokumenttia: raportti ja loki. Raportissa on selkeässä muodossa ilmoitettu testien yhteenveto. Lokista löytyy hyvin jäsennettynä kaikkien testitapausten tiedot ja tulokset, joihin epäonnistuneissa testeissä on ilmoitettu virheen aiheuttaja ja syy. Dokumenteissa ei ole esitetty toimintavarmuudelle tutkimuksia. Ilmoitettuja vikoja on 22 kappaletta, joista kahdella on korkea prioriteetti [49].

STAF:issa on järjestelmän tarkkailua varten oma monitor-palvelu. Palvelua voidaan pyytää esittämään järjestelmän sen hetkinen ja sitä edeltäneet tilat. Monitor-palvelun ilmoittamat tiedot muodostuvat testitapauksissa käytettyjen lokiinkirjoitusfunktioiden käytöstä. Tällä tavoin STAF on yhteneväinen Robot Frameworkin kanssa, jossa myöskin järjestelmän monitoroitavuus on pääosin testitapausten kehittäjien käsissä. Testiraporttien rakenne ja siihen kuuluvat osat ovat täysin muokattavissa. Raportti voidaan muodostaa esimerkiksi keräämällä STAX:in avulla XML-tiedostoon kaikki testitapauksista saatu data. Tämä XML-tiedoston avulla voidaan luoda vaikka HTML-tiedosto, jossa tulokset on jaoteltu selkeästi. STAF:in vakaudesta ei löydy erillistä todistelua, mutta järjestelmä on edelleen IBM:n käytössä ja kehityksessä, josta voidaan arvioida sen toimivuuden olevan vähintään riittävällä tasolla. Avoinna olevia bugiraportteja on 87 kappaletta, joista 23:een on tehty jonkinlaista päivitystä 2010-luvulla ja 18:ta tapaukseen uusimmat päivitykset ovat ennen vuotta 2005. Tästä päätellen suurin osa listalla olevista bugeista on harvinaisia tai sellaisia, joilla ei ole merkitystä [50].

5.5 Käyttöönotto

Ensimmäinen askel käyttöönotossa on asentaa järjestelmät ja niiden vaatimat ohjelmat. Robot Frameworkin asentaminen on mutkatonta käyttämällä pip-nimistä paketinhallintaohjelmistoa, joka on suunniteltu Pythonilla kirjoitettujen ohjelmistojen asentamiseen ja hallinnoimiseen. Asennus onnistuu pip:iä käyttämällä yhdellä komennolla komentoriviltä. Tämän jälkeen asennetaan ulkoiset kirjastot ja työkalut, joita halutaan käyttää.

Twister Frameworkin asentaminen on moniosaisempi prosessi kuin Robot Frameworkin asennus. Twister Framework vaatii Apache HTTP-palvelinohjelman ja MySQL-tietokantaohjelmiston toimiakseen, joten ne tulee asentaa ja konfiguroida ensin. Tämän jälkeen Twister ladataan internetistä koneelle, ja suoritetaan asennus valmiiden skriptitiedostojen avulla. Tämän jälkeen voidaan halutessa asentaa ja konfiguroida Java-sovelma toimintakuntoon.

STAF on Twister Frameworkin tapaan hieman hankalampi asentaa kuin Robot Framework. STAF asennetaan lataamalla verkosta sopivin asennustiedosto ja käynnistämällä sen mukana tuleva asennusohjelma (graafinen tai komentorivi) tai asennetaan kaikki käsin. Tämän jälkeen asennetaan halutut erilliset lisäpalvelut purkamalla ladatut paketit services-kansioon, jonka jälkeen ne konfiguroidaan toimimaan STAF:in kanssa.

Kun halutut ohjelmat ja lisäosat on asennettu alkaa konfigurointi. Kaikkiin kolmeen ohjelmistoon on määritettävä SUT:t ja niiden kanssa keskusteluun tarvittavat parametrit sekä asetettava oikeat asetukset. Automatisoinnin kannalta järjestelmissä tulee olla määritetty, mitkä käännökset otetaan testattavaksi ja mistä nämä käännökset ovat saatavilla. STAF:in ja Robot Frameworkin tapauksessa tulee myös luoda testien käynnistykseen ja ohjaukseen käytettävät skriptitiedostot. Twister Frameworkissa ja STAF:issa tulee lisäksi rakentaa/muokata testiraportointi sopimaan omiin tarpeisiin.

6 TULOKSET JA JOHTOPÄÄTÖKSET

Tässä luvussa käydään läpi vertailun perusteella saadut tulokset. Tulokset yhdistetään haastatteluissa nousseiden painotusten ja vaatimusten kanssa yhteen ja niistä muodostetaan johtopäätökset.

Lähtökohtaisesti Twister Framework on teknisiltä ominaisuuksiltaan kolmikon kattavin ohjelmisto. Sen suuri vahvuus on valmis verkon yli tapahtuvan toiminnan konfiguroitavuus. Lisäksi järjestelmän monitoroitavuus, SUT-kohtaisten asetusten säätö ja riittävä määrä tärkeitä lisäosia tekevät siitä kohtuullisen valmiin paketin suoraan asennuksesta.

Myös järjestelmän käytettävyydessä Twister Framework on toisia edellä. Sen tarjoama web-käyttöliittymä ja Java-sovelma helpottavat testien ja SUT:ien tarkkailua ja muokkausta. STAF:in tarjoama STAX:in hallinnointiin tarkoitettu käyttöliittymä ei ole yhtä kattava kuin Twister Frameworkin tarjoama Java-sovelma.

Testitapausten luontiin paras vaihtoehto näistä kolmesta on Robot Framework. Sen kolme valittavissa olevaa yksinkertaista ja selkeää syntaksia yhtenäistävät testitapaukset helposti kehitettäviksi ja tulkittaviksi. Vaikka näiden testitapausten toiminta mahdollisesti riippuu monimutkaisten testikirjastojen luomisesta, ei näiden testikirjastojen monimutkaisuus eroa merkittävästi STAF:in ja Twister Frameworkin testitapauksista.

Robot Frameworkissä ja STAF:issa toimivuuden tarkastelu määrittyy järjestelmän kehittyessä. Twister Frameworkissä toimivuutta voidaan tarkastella heti valmiiden työkalujen avulla. Selkeimmät ja monipuolisimmat testiraportit luo Robot Framework. STAF:issa ja Twister Frameworkissa testiraportit ovat kuitenkin muokattavissa tarpeita vastaaviksi. Ohjelmien suoritusvarmuutta ei voitu tarkastella niin kattavasti, että ne olisivat olleet aseteltavissa paremmusjärjestykseen.

Twister Frameworkin käyttöönottoprosessi on siinä mielessä helpoin, että se tarjoaa suoraan asennuksen jälkeen eniten ominaisuuksia. Monet näistä ominaisuuksista ovat sellaisia, joiden luomisessa muille järjestelmille kuluu runsaasti aikaa. On kuitenkin huomioitava, että tästä huolimatta myös Twister Framework vaatii paljon konfigurointia ennen kuin se on toimintavalmis.

Twister Framework täyttää lähes kaikki toivomukset, mitä haastatteluissa tuli esille. Se on monipuolinen, joustava, laajasti konfiguroitavissa ja sen toiminnan seuraaminen on helppoa. Lisäksi testitapaukset voidaan tehdä skriptikielellä sekä se on väitetysti erittäin stabiili. Twister Frameworkin ominaisuudet ovat luotu juuri Elektrobitin kaltaisen telekommunikaatioalalla toimivan yrityksen tarpeisiin.

STAF on askeleen Twister Frameworkiä enemmän muokattavissa oleva järjestelmä. STAF, johon on lisätty STAX ja muita palveluita, on pohja, jonka päälle on mahdollista rakentaa mitä monipuolisimpia testijärjestelmiä. Se on hieman rakennussarjamainen järjestelmä, joka toimittaa vain kriittisimmät asiat, jättäen kaiken muun järjestelmän käyttäjän käsiin.

Robot Framework on oikeastaan aivan erilainen järjestelmä kahteen muuhun verrattuna. Se on tarkoitettu hyväksymistestaukseen ja hyväksymistestauslähtöiseen ohjelmistokehitykseen, joka on siis testausta käyttäjän näkökulmasta. Periaatteessa mikään ei kuitenkaan estä Robot Frameworkin käyttöä muiden vaiheiden testien suorittamiseen. Sen käyttö saattaisi tehdä testitapauksista helpommin luettavia ja testikokonaisuuksien ymmärtämisestä selkeämpää.

Taulukkoon 5 on koottu yleiskatsaus haastattelujen asettamien kriteereiden ja testiautomaatiojärjestelmien ominaisuuksien välisestä vertailusta.

Taulukko 5. Haastatteluista saatuihin kriteereihin vertaileminen.

Vertailu haastattelujen asettamiin kriteereihin	
Vaatimus	*Tulos*
Stabiilius	Kehittäjä lupaa Twister Frameworkin olevan erittäin vakaa. Muista järjestelmistä ei tämän osalta erillistä dataa.
Konfiguroitavuus	Kaikissa kattava. Twister Frameworkin käyttöliittymä tekee konfiguroinnista helpompaa kuin verrokeilla.
Monitoroitavuus	Twister Frameworkissa kattavat työkalut järjestelmän ja testien monitoroimiseen. STAF monitor-palvelulla tarjoaa osittain samankaltaista toimintaa. Robot Framwork ei tarjoa ratkaisua.
Ymmärrettävyys	Yksikään vertailtavista ohjelmista ei ole täysin selkeä, mutta Twister Framework käyttöliittymällän on niistä selkein.
Testitapausten luontiin helppo kieli	Kaikissa järjestelmissä mahdollisuus käyttää helppoa kieltä testitapausten luontiin. Robot Frameworkissa omat erikseen helpotetut syntaksit testitapausten kirjoittamiseen.
Suuri määrä valmiita lisäkirjastoja	Robot Framework sisältää kohtuullisen määrän lisäkirjastoja testitapausten luontiin. Twister Frameworkissä ja STAF:ssa jotakin lisäosia ja palveluita voidaan käyttää testitapauksissa.

7 YHTEENVETO

Tietotekniikan ja ohjelmistojen kasvava merkitys ja määrä tekee ohjelmistovirheistä erittäin kalliita ja joissakin tilanteissa jopa hengenvaarallisia. Tästä syystä ohjelmistotestauksen merkitys on merkittävä niiden kehittäjille, niin taloudellisesti kuin imagollisesti. Osa ohjelmistotestausta on testiautomaatio. Se on osa ohjelmiston laadunvalvontajärjestelmää. Sillä pyritään varmistamaan jo testattujen osien toiminta uuden toiminnallisuuden kanssa.

Tämän opinnäytetyön tarkoituksena on ollut vertailla saatavilla olevia avoimen lähdekoodin testiautomaatiojärjestelmiä. Sen tulosten ja johtopäätösten perusteella Elektrobitillä on mahdollisuus valita itselleen tilannekohtaisesti sopivin järjestelmä.

Vertailtavat testiautomaatiojärjestelmät Robot Framework, Twister Framework ja STAF eroavat toisistaan monilla tavoin. Niiden tarjoamat ominaisuudet ja ratkaisut on pyritty esittämään työssä monipuolisesti ja selkeästi.

Järjestelmiä ei ollut mahdollista rajallisesta ajasta johtuen testata täysin kattavasti. Lisäksi siihen olisi vaadittu useita tietokoneita, testattava järjestelmä ja testitapauksia. Tällä tavoin olisi voitu kokeellisesti testata järjestelmien stabiliteettia. Järjestelmät testattiin suhteellisen pikaisesti esimerkkien ja dokumentaatioiden avulla.

Twister Framework vastaa vertailluista järjestelmistä parhaiten työn vaatimuksissa ja haastatteluiden perusteella määritettyjä kriteerejä. Sen valmiiksi tarjoamat ratkaisut ylläpidettävyyteen ja käytettävyyteen olivat vertailun parhaat. Työn lopputulos vastaa sille asetettuja tavoitteita ja vaatimuksia.

LÄHTEET

1. Hallinto- ja ohjausjärjestelmä, Elektrobit [WWW-dokumentti] Saatavilla: <http://www.elektrobit.com/keita_me_olemme/yritystiedot/hallinto_ja_ohjausjarjestelma> (Luettu 14.04.2014)

2. Work at EB, Elektrobit [WWW-dokumentti] Saatavilla: <http://www.elektrobit.com/who_we_are/work_at_eb> (Luettu 14.04.2014)

3. Avainlukuja, Elektrobit [WWW-dokumentti] Saatavilla: <http://www.elektrobit.com/keita_me_olemme/yritystiedot/avainlukuja> (Luettu 14.04.2014)

4. Strategiset linjaukset, Elektrobit [WWW-dokumentti] Saatavilla: <http://www.elektrobit.com/keita_me_olemme/yritystiedot/strategiset_linjaukset> (Luettu 14.04.2014)

5. Välimäki M. Oikeudet tietokoneohjelmistoihin. Helsinki: Talentum; 2009. ISBN 978-952-14-1344-5

6. Software Licensing, The University of North Carolina Greensboro, [WWW-dokumentti] Saatavilla: <https://its.uncg.edu/Software/Licensing/> (Luettu 2.3.2014)

7. About the Open Source Initiative, Open Source Initiative. [WWW-dokumentti] Saatavilla: <http://opensource.org/about> (Luettu 2.3.2014)

8. The Open Source Definition, Open Source Initiative [WWW-dokumentti] Saatavilla: <http://opensource.org/osd> (Luettu 1.3.2014)

9. Licenses by Name, Open Source Initiative [WWW-dokumentti] Saatavilla: <http://opensource.org/licenses/alphabetical> (Luettu 1.3.2014)

10. Report of License Proliferation Committee and draft FAQ, Open Source Initiative [WWW-dokumentti] Saatavilla: <http://opensource.org/proliferation-report> (Luettu 1.3.2014)

11. What is Free Software, Free Software Foundation [WWW-dokumentti] Saatavilla: <http://www.fsf.org/about/what-is-free-software> (Luettu 2.3.2014)

12. What is Free Software, GNU Project [WWW-dokumentti] Saatavilla: <https://www.gnu.org/philosophy/free-sw.html> (Luettu 14.3.2014)

13. What is Free Software, Free Software Foundation [WWW-dokumentti] Saatavilla:<http://www.fsf.org/about/what-is-free-software> (Luettu 2.3.2014)

14. Richard Stallman, Why Open Source misses the point of Free Software [WWW-dokumentti] Saatavilla: <https://www.gnu.org/philosophy/open-source-misses-the-point.html> (Luettu 3.3.2014)

15. Open Source Licenses by Name, Open Source Initiative [WWW-dokumentti] Saatavilla: <http://opensource.org/licenses/alphabetical> (Luettu 14.3.2014)

16. Various Licenses and Comments about them – Free Software Foundation [WWW-dokumentti] Saatavilla: <https://www.gnu.org/licenses/license-list.html> (Luettu 14.3.2014)

17. Kasurinen J.P. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo; 2013. ISBN 978-952-5912-99-9

18. Cambridge University Study States Software Bugs Cost Economy $312 Billion Per Year [WWW-dokumentti] Saatavilla: <http://www.prweb.com/releases/2013/1/prweb10298185.htm> (Luettu 5.3.2014)

19. Knight Shows How to Lose $440 Million in 30 Minutes [WWW-dokumentti] Saatavilla: <http://www.businessweek.com/articles/2012-08-02/knight-shows-how-to-lose-440-million-in-30-minutes> (Luettu 5.3.2014)

20. Fowler M. Continuous Integration [WWW-dokumentti] Saatavilla: <http://www.martinfowler.com/articles/continuousIntegration.html> (Luettu 6.3.2014)

21. What is Selenium, Selenium [WWW-dokumentti] Saatavilla: <http://docs.seleniumhq.org> (Luettu 14.3.2014)

22. Hirsjärvi S. - Remes P. - Sajavaara P. Tutki ja kirjoita. Helsinki: Tammi; 2000. ISBN 978-951-3148-36-2

23. Katara M. Ohjelmistojen testaus, Hyväksymistestaus, Tampereen teknillinen yliopisto [PDF-dokumentti] Saatavilla:<http://www.cs.tut.fi/~testaus/s2011/luennot/OHJ-3060_2011_110-170.pdf> (Luettu 14.04.2014)

24. Etusivu, Robot Framework [WWW-dokumentti] Saatavilla: <http://robotframework.org> (Luettu 12.3.2014)

25. Pekka Klärck, Homepage [WWW-dokumentti] Saatavilla: <http://eliga.fi/index.html> (Luettu 12.3.2014)

26. Janne, Robot Framework Newsletter, November 11 [WWW-dokumentti] Saatavilla: <http://hereberobots.blogspot.fi/2011/11/robot-framework-newsletter-november.html> (Luettu 12.3.2014)

27. Ragget D, Le Hors A, Jacobs I, HTML 4.01 Specification , W3C Recommendation [WWW-dokumentti] Saatavilla: <http://www.w3.org/TR/html4/> (Luettu 21.4.2014)

28. Bray, T. - Paoli, J. - Sperberg-McQueen, C. M. - Maler, E. - Yergeau, F. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation [WWW-dokumentti] Saatavilla: <http://www.w3.org/TR/REC-xml/> (Luettu 7.4.2014)

29. Jython General Information [WWW-dokumentti] Saatavilla: <https://wiki.python.org/jython/JythonFaq/GeneralInfo> (Luettu 7.4.2014)

30. Keyword-Driven RobotDemo [WWW-dokumentti] Saatavilla: <https://bitbucket.org/robotframework/robotdemo/src/master/keyword_driven.txt> (Luettu 6.4.2014)

31. Data-Driven RobotDemo [WWW-dokumentti] Saatavilla: <https://bitbucket.org/robotframework/robotdemo/src/master/data_driven.txt> (Luettu 6.4.2014)

32. Etusivu, Cucumber [WWW-dokumentti] Saatavilla: <http://cukes.info/> (Luettu 6.4.2014)

33. Gherkin, Cucumber [WWW-dokumentti] Saatavilla: <http://cukes.info/gherkin> (Luettu 6.4.2014)

34. Gherkin RobotDemo [WWW-dokumentti] Saatavilla: <https://bitbucket.org/robotframework/robotdemo/src/master/da ta_driven.txt> (Luettu 6.4.2014)

35. FAQ, Twister Framework [WWW-dokumentti] Saatavilla: <http://www.twistertesting.com/faqs/> (Luettu 7.4.2014)

36. History, Luxoft [WWW-dokumentti] Saatavilla: <http://www.luxoft.com/luxoft-overview/history/> (Luettu 7.4.2014)

37. Twister User Guide [PDF-tiedosto] Saatavilla: <http://www.twistertesting.com/twister-user-guide/> (Luettu 7.4.2014)

38. Application (GUI) Testing, Twister Framework [WWW-dokumentti] Saatavilla: <http://www.twistertesting.com/testing-solutions/application-gui-testing/> (Luettu 7.4.2014)

39. Product, Twister Framework [WWW-dokumentti] Saatavilla: <http://www.twistertesting.com/product/> (Luettu 7.4.2014)

40. Twister Plugins Guide [PDF-tiedosto] Saatavilla: <http://ww.twistertesting.com/twister-plugins-guide/> (Luettu 7.4.2014)

41. Software Testing Automation Framework, Wikipedia (English) [WWW-dokumentti] Saatavilla: <https://en.wikipedia.org/wiki/Software_Testing_Automation_Framework> (Luettu 7.4.2014)

42. Frequently Asked Questions About STAF V3, STAX, and STAF Services [WWW-dokumentti] Saatavilla: <http://staf.sourceforge.net/current/STAFFAQ.htm> (Luettu 7.4.2014)

43. Homepage, Software Testing Automation Framework [WWW-dokumentti] Saatavilla: <http://staf.sourceforge.net/index .php> (Luettu 7.4.2014)

44. Download Services for STAF V3, Software Testing Automation Framework [WWW-dokumentti] Saatavilla: <http://staf.sourceforge.net/getservices.php> (Luettu 7.4.2014)

45. Hands-on Automation with STAF/STAX Part 3 [PowerPoint-esitys] Saatavilla: <http://staf.sourceforge.net/educ/STAF-STAX-HandsOn-Part3.ppt> (Luettu 7.4.2014)

46. STAX Service User's Guide [WWW-dokumentti] Saatavilla: <http://staf.sourceforge.net/current/STAX/staxug.html> (Luettu 7.4. 2014)

47. Getting Started Guide, STAF Eclipse Plug-ins [WWW-dokumentti] Saatavilla:<http://staf.sourceforge.net/current/eclipsegs.htm> (Luettu 15.4.2014)

48. QuickSTAF Features, Testformation Inc [WWW-dokumentti] Saatavilla:<http://quickstaf.testformation.com/features/> (Luettu 15.4.2014)

49. Robot Framework List of Issues, Google Code [WWW-dokumentti] Saatavilla: <https://code.google.com/p/robotframework/issues/list?can=2&q=type=Defect&colspec=ID%20Type%20Status%20Priority%20Target%20Owner%20Summary%20Stars> (Luettu 9.4. 2014)

50. SW Test Automation Framework Bug List, Sourceforge [WWW-dokumentti] Saatavilla: <http://sourceforge.net/p/staf/bugs/limit=100> (Luettu 9.4.2014)