

Joni Lindgren

Automaatio kontekstilähtöisessä ohjelmistotestauksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan ko

Insinööriytyö

19.11.2013

Tekijä(t) Otsikko	Joni Lindgren Automaatio kontekstilähtöisessä ohjelmistotestauksessa
Sivumäärä Aika	44 sivua + 2 liitettä 19.11.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Simo Silander Ohjelmistoinsinööri Valtteri Konttinen
<p>Insinööriyössä oli tavoitteena tutkia ja hyödyntää automaatiota Intermarketing Oy:n ohjelmisto- ja laiteratkaisujen testauksessa. Erityisesti tavoitteena oli tehostaa testausta järjestelmissä, joihin kuuluu lähes aina manuaalisesti käsiteltäviä käteistä rahaa käsitteleviä fyysisiä laitteita.</p> <p>Ensin tutkittiin erilaisia näkökantoja ja lähestymistapoja ohjelmistotestaukseen. Tutkituista ajatusmalleista kontekstiohjautunut testaus osoittautui työn kannalta hyväksi näkökulmaksi ohjelmistotestaukseen ja koko työhön. Tämän jälkeen tutkittiin testauksen automatisoinnin yleisiä ongelmia ja mahdollisuuksia. Tutkimuksen perusteella päätettiin, että testauksen täydelliseen automaatioon ei kannata pyrkiä, ja työn kannalta automaattisten testien hyödyllisin käyttötarkoitus on olla tukena manuaaliselle tutkivalle testaukselle.</p> <p>Seuraavaksi tutustuttiin automatisoinnissa käytettäviin skriptaustekniikoihin, joista valittiin lopulta hyvinkin laajamittaisen ja ylläpidettävän testiautomaatoratkaisun mahdollistava avainsanaohjattu testaus. Tämän jälkeen esiteltiin automatisoinnissa käytettävä työkalu, Robot Framework -testiautomaatiokehys, joka soveltuu erinomaisesti avainsanaohjattujen testauksen toteuttamiseen. Seuraavaksi toteutettiin konseptin todennus ohjelmisto- ja laiteratkaisuun, joka koostuu setelinlaskimesta/lajittelijasta ja Swing-käyttöliittymällisestä Java-sovelluksesta. Järjestelmällä suoritetaan laskentaeriä, joiden tulokset tallennetaan XML-muodossa. Lopputuloksena oli yksi onnistuneen laskentaerän suorittava avainsanaohjattu testi, joka voidaan suorittaa puoliautomaattisesti tai täysin automaattisesti. Automaattinen testiajo suoritetaan simulaattoria vasten ja puoliautomaattinen fyysisen laitteen kanssa. Avainsanaohjattu testi on molemmissa sama, mutta puoliautomaattisessa suorituksessa lisätään aikaviivettä pakollisen manuaalisen toiminnon suorituksen kohdalle. Aikaviiveen käyttö oli yksinkertainen ja tehokas ratkaisu manuaalisen osuuden hoitamiseksi.</p> <p>Insinööriyön tutkimuksen ja konseptin todennuksen pohjalta työn toimeksiantaja sai paljon tärkeää tietoa testauksesta sekä testiautomaation haasteista ja mahdollisuuksista. Intermarketing Oy:n ohjelmistokehityksessä tullaan jatkossa ottamaan testiautomaatiota käyttöön työn pohjalta. Työn tutkimusosuudessa mainittuja automaatiomenetelmiä, konseptin todennusta, ja jatkokehitysideoita tullaan tutkimaan ja hyödyntämään yrityksen ohjelmistokehityksessä.</p>	
Avainsanat	testauksen koulukunnat, kontekstiohjautunut testaus, testiautomaatio, Robot Framework

Author(s) Title	Joni Lindgren Automation in context-driven software testing
Number of Pages Date	44 pages + 2 appendices 19 November 2013
Degree	Bachelor of Engineering
Degree Programme	Computer Engineering
Specialisation option	Software Engineering
Instructor(s)	Simo Silander, Senior Lecturer Valtteri Konttinen, Software Engineer
<p>The aim of this Bachelor's thesis was to examine and utilize automation in the testing of Intermarketing's software and hardware solutions. In particular, the goal was to improve the testing of systems, which almost always include manually operated physical cash handling devices.</p> <p>First, a variety of viewpoints and approaches to software testing were examined. Of the examined thought patterns, context-driven testing turned out to be a good viewpoint to software testing and to the whole study. The common problems and opportunities of test automation were examined next. Based on the study, it was decided that a complete automation of testing should not be the goal. Instead, considering the goal of the thesis, the most useful use of test automation was to be a support for manual exploratory testing.</p> <p>The different scripting techniques used in automation were examined next. Finally, keyword-driven testing was chosen, because it enables test automation solutions that can be very large-scale and maintainable. Next, the test automation tool to be used, Robot Framework, was introduced. Robot Framework is a test automation framework that is ideally suited for keyword-driven testing. The next phase was the design and implementation of a POC (Proof Of Concept). The POC was done for a software and hardware solution, which consists of a currency sorter/counter and a Java Swing application. The system is used to perform batch transactions and the transaction results are saved in XML format. The end result of the POC was one keyword-driven test that is used to perform a successful transaction of one batch. The test can be carried out semi-automatically or fully automatically. The automatic test runs against a simulator, and the semi-automatic against the physical device. The keyword-driven test is the same in both cases, but in the semi-automatic execution, there is an additional time delay for the mandatory manual operation. The time delay was a simple and effective solution to perform the manual part of the test.</p> <p>On the basis of the study and the POC realised, the subscriber of the thesis got a lot of important information on testing and test automation challenges and opportunities. On the basis of the thesis, test automation will be taken in to use in Intermarketing's software development. Furthermore, the automation methods, POC and the ideas for further development, studied in the thesis, will be examined and utilized in Intermarketing's software development, too.</p>	
Keywords	schools of testing, context-driven testing, test automation, Robot Framework

Sisällys

1	Johdanto	1
2	Ohjelmistotestaus	3
2.1	Perinteinen ja ketterä testaus	3
2.2	Testauksen koulukunnat	4
3	Testiautomaatio	9
3.1	Ongelmat	9
3.2	Mahdollisuudet	10
4	Testiskriptausta	12
4.1	Nauhoitus ja toisto	14
4.2	Lineaarinen skriptausta	16
4.3	Modulaarinen skriptausta	17
4.4	Aineisto-ohjattu testaus	19
4.5	Avainsanaohjattu testaus	21
4.6	Robot Framework	23
5	Konseptin todennus	26
5.1	Määritelmä	26
5.2	Toteutus	28
5.3	Tulokset ja jatkokehitys	36
6	Yhteenveto	39
	Lähteet	42
	Liitteet	
	Liite 1. Laskentaerän suoritus	
	Liite 2. Konseptin todennuksen avainsanat	

1 Johdanto

Työn toimeksiantaja Intermarketing Oy tarjoaa teknologia- ja palveluratkaisuja pankin ja kaupan alalle, julkiselle sektorille ja liikenteelle. Asiakasprojekteissa panostetaan tehokkuutta parantaviin rahankäsittely-, asiakasohjaus- ja turvallisuusratkaisuihin sekä asiakaslähtöisiin ohjelmistoratkaisuihin. Ohjelmistoratkaisut ovat usein räätälöityjä ja integroitava laitteisto on vaihtelevaa. Laitteisiin kuuluu muun muassa useita erilaisia talletus- ja tilitysautomaatteja, turvakassoja, rahanvaihettajia sekä kolikon- ja setelinlaskimia ja lajittelijoita.

Intermarketing Oy:n ohjelmistotestauksessa järjestelmä- ja hyväksymistestauksella on tärkeä rooli. Järjestelmätestaus on yleensä vaatimuksiin ja suunnitteluratkaisuihin perustuvaa koko järjestelmän testausta. Intermarketing Oy:n tapauksessa kuhunkin asiakaskohtaiseen järjestelmään kuuluu yleensä aina graafisella käyttöliittymällä käytettävä sovellus, jolla hallinnoidaan jotain fyysistä laitetta. Järjestelmätestausta tehdään mahdollisimman tarkkaan asiakkaan ohjelmisto- ja laiteratkaisuja vastaavassa ympäristössä. Hyväksymistestaus tapahtuu myös usein järjestelmätestauksen tasolla, mutta siinä keskitytään erityisesti varmentamaan asiakkaan vaatimusten täytyminen. Nykyään järjestelmä- ja hyväksymistestaus tapahtuu lähes kokonaan manuaalisesti. Tätä työtä varten haastateltiin Intermarketing Oy:n ohjelmisto- ja laiteratkaisujen kehitykseen osallistuvaa henkilöstöä [1], ja erityisesti seuraavat nykyiset testauksen haasteet nousivat esiin:

- manuaalisesti tehtävä tuotteiden laadun ja toimivuuden varmistaminen kuormittaa pientä ohjelmistotiimiä turhan paljon
- usein juuri testauksesta joudutaan karsimaan aikaa tiukassa aikataulussa
- samoja hitaita manuaalisia järjestelmä- ja hyväksymistestejä joudutaan suorittamaan uudestaan eri asiakasratkaisuja testatessa, vaikka käyttöliittymä olisi käytännössä sama ja tehdyt muutokset ”pinnan alla” konfiguraatioissa ja laitemalleissa
- selkeidenkin ongelmien ja ”rikkinäisten” julkaisu ehdokkaiden (*Release Candidate*) havaitsemiseen menee turhan kauan
- ohjelmiston julkaisuvalmiutta pitäisi pystyä seuraamaan paremmin
- ohjelmiston yleistä laatua pitäisi pystyä parantamaan sekä valvomaan nykyistä paremmin

- useista erilaisista laitteista johtuen testausta ei edes voitaisi automatisoida täysin, koska fyysisiä laitteita joudutaan käsittelemään käsin.

Tavoitteena olisi hyödyntää automatisointia yrityksen ohjelmistoprojektien testauksessa siinä määrin kuin on järkevää ja mahdollista. Tarkoitus olisi päästä eroon usein toistettavasta ja jopa turhasta manuaalisesta työstä sekä nopeuttaa ja kehittää erityisesti regressiotestausta. Regressiotestaus on ohjelmiston testaamista millä tahansa tasolla tarkoituksena varmistaa, ettei jo toteutetut ja toimivat ohjelmiston osat ole menneet rikki esimerkiksi uuden ominaisuuden toteutuksen myötä. Regressiotestauksella paljastetaan siis ohjelmistoregressiota, yleensä käyttäen vanhoja ohjelmiston ominaisuuksia testaavia tarkastuksia. Kyseisiä ohjelman ominaisuuksia ja vaatimuksia testaavia toimintasarjoja kutsutaan myös testitapauksiksi (*Test Case*). Useasta testitapauksesta voidaan muodostaa testikokoelmia (*Test Suite*), joita suoritetaan tällä hetkellä siis järjestelmä- ja hyväksymistesteissä manuaalisesti. Kyseiset regressiotestit suoritetaan usein samalla tavalla ja ne vievät usein paljon aikaa ja vaativat jatkuvaa keskittymistä.

Kun usein toistettavia tarkastuksia saadaan automatisoitua, regressiotestaus nopeutuu ja voidaan keskittyä myös enemmän tutkivaan testaukseen, joka on harjoittelematonta ja improvisoitua testausta. Tutkiva testaus on erittäin tärkeää, sillä usein juuri manuaalisella tutkivalla testauksella löydetään uusia ja odottamattomia ongelmia ja virheitä ohjelmistosta, kun taas regressiotestauksella pyritään lähinnä estämään tiedossa olevia mahdollisia ongelmia. Automaatiolla pyritään siis tukemaan manuaalista tutkivaa testausta. Testauksesta olisi yleisesti tarkoitus tulla entistä kattavampaa, tarkempaa ja ylläpidettävämpää.

Testiautomaatio on siis tietokoneen suorittamaa testausta, joka on ihmisen suorittamaa testausta nopeampaa, toistettavampaa ja parhaimmillaan myös tarkempaa. Testiautomaatioon liittyy myös riskejä, kuten liiallinen luottaminen automaattisiin testeihin ja epärealistiset odotukset automaatiolta sekä mahdollisesti suureksi paisuvat toteutus- ja ylläpitokustannukset. Insinööriyön tavoitteena on tutkia testiautomaatiota ja erityisesti työn toimeksiantajan tarpeeseen sopivia testiautomaation lähestymistapoja ja työkaluja, sekä suunnitella ja toteuttaa konseptin todennus (*Proof of Concept*) tutkimuksen pohjalta. Konseptin todennuksessa tavoitteena on siis kehittää yhden tietyn ohjelmiston ja laiteratkaisun testausta automatisoinnin avulla. Ensisijaisena kehityskohteena on käyttöliittymättestaus ja käyttöliittymän läpi tehtävä järjestelmätestaus. Tarkoituksena olisi jatkossa ottaa insinööriyön tulosten perusteella hyväksi todettuja menetelmiä,

mikäli sellaisia on löydetty, käyttöön myös muissa yrityksen ohjelmisto- ja laitteistoratkaisuissa.

2 Ohjelmistotestaus

Ohjelmistotestaus on erittäin oleellinen osa ohjelmistojen tuottamista. Se on nykyään jatkuvasti kehittyvä, merkittävämpi ja myös kiistelty ohjelmistokehityksen osa-alue, johon on useita erilaisia lähestymistapoja. Jopa käsitteiden ja termien käytöstä ja tulkinnoista on paljon erilaisia näkemyksiä. Eri näkemysten ja kokemusten tutkiminen on tärkeä osa testausprosessien kehityksessä ja erityisesti automaatiota suunniteltaessa. Tässä työssä ei syvennyttä tarkastelemaan kaikkia erilaisia testauksen tekniikoita ja kehitystapoja, sillä testauksen maailma on hyvin laaja eikä läheskään kaikki mahdu tähän työhön. Tässä työn osiossa käydään pääpiirteittäin läpi eri näkökulmia testaukseen ja valitaan juuri tämän työn tavoitteen ja toimeksiantajan kannalta paras mahdollinen lähestymistapa.

2.1 Perinteinen ja ketterä testaus

Testauksesta ja siihen liittyvistä menettelytavoista puhuttaessa ohjelmistotestaus jaetaan yksinkertaisimmillaan perinteisiin ja ketteriin menetelmiin. Usein perinteisen menetelmän esimerkkinä käytetyssä vesiputousmallissa [2] ohjelmistokehityksen vaiheet ovat järjestyksessä lueteltuna:

- *määrittely*
- *suunnittelu*
- *toteutus*
- *integraatio*
- *testaus*
- *toimitus*
- *ylläpito.*

Vesiputousmallisessa ohjelmistokehityksessä testaus on erillinen ohjelmistokehitysprojektin vaihe, jossa testataan projektin lopuksi valmiin ohjelmiston toimintaa ennen toimi-

tusta asiakkaalle. Usein ohjelmistokehitysprojektit saattavat kuitenkin muotoutua projektin edetessä useampaankin kertaan esimerkiksi asiakkaan vaatimusten muuttuessa testausvaiheessa. Vesiputousmallissa mahdolliset muutokset määrätyksiin etenkin projektin testausvaiheessa aiheuttavat helposti paljon lisää työtä, koska ohjelmistoa joudutaan mahdollisesti muuttamaan määrittelystä lähtien. Lisäksi myöhään tehtävä testausvaihe aiheuttaa helposti myös sen, että tehdyt virheet paljastuvat myöhään ja virheen aiheuttama haitta on mahdollisesti levinnyt laajemminkin ohjelmistoratkaisuun. Vesiputousmalli saattaa olla toimiva ratkaisu, kun ohjelmisto on hyvin yksinkertainen ja määritykset ovat hyvin selvillä, mutta usein monimutkaisemman ohjelmistoprojektin täydellinen määrittely ja suunnittelu etukäteen on lähes mahdotonta [2].

Ketterä kehitys tarkoittaa yleensä itseohjautuvaan ja nopeaa tiimityöhön perustuvaa ohjelmistokehitystä, jossa dokumentaatio pyritään pitämään kevyenä ja ohjelmistoa kehitetään nopeissa sykleissä, joita kutsutaan iteraatioiksi. Yhden iteraation kesto on useimmiten noin neljä viikkoa ja tavoitteena on periaatteessa aina julkaisukelpoinen ohjelmisto iteraation lopuksi. Yksi iteraatio on periaatteessa kuin pieni perinteinen ohjelmistoprojekti [3]. Ketterissä menetelmissä yleisenä toimintamallina on aloittaa testaus yhdessä implementoinnin kanssa, jolloin se on jatkuvasti kehityksessä mukana oleva prosessi. Ketterissä lähestymistavoissakin on keskenään eroja. Jotkut suhtautuvat ketteriin menetelmiinkin siten, että jokin ketterä menetelmä, esimerkiksi tiukasti noudatettava testivetoinen kehitys (*Test-driven development, TDD*), on ainoa oikea tapa tehdä asioita. Testivetoisessa kehityksessä laaditaan ennen uuden ominaisuuden toteutusta testitapaus, jonka avulla todetaan myös ominaisuuden valmistuminen. Toiset taas suhtautuvat ketteryyteen lähestymistapana ja näkemyksenä, jossa tiettyjä tekniikoita ja työkaluja ei ole tarkoituskaan käyttää tiukasti yhdellä tavalla vaan tarkoituksen mukaisesti [4, s. 12-15].

2.2 Testauksen koulukunnat

Usein testausalan kirjallisuudessa sekä oppimateriaaleissa näkökulmat ja menetelmät ohjelmistotestaukseen jaetaan ketteriin ja perinteisiin. Tätä työtä varten tehdyssä taustatutkimuksessa nousi kuitenkin esiin myös laajempi näkemys ohjelmistotestaukseen: ajatusmallien ja lähtökohtien perusteella toteutuva jako erilaisiin testauksen koulukuntiin (*schools of testing*). Jako koulukuntiin auttaa ymmärtämään etenkin erilaisia tavoitteita ja arvoja. Koulukuntajako auttaa selventämään, miksi ohjelmistotestauksen asian-

tuntijat ovat eri mieltä asioista. Yhtenä syynä koulukuntiin jaottelussa on erimielisyyksien selventämisen kautta myös väittelyiden perustan parantaminen. Lisäksi koulukunta-ajattelu auttaa testauksen alalla toimivia ymmärtämään omia asemiaan ja se auttaa myös oppimiseen motivoinnissa. Koulukunta ei itsessään määrittele tiettyä tekniikkaa, vaan eri koulukuntien käyttämät tekniikat ja menetelmät voivat jopa täydentää toisiaan. Jaottelussa on kysymys etenkin erilaisista näkemyksistä testaukseen. Testauksen koulukunnat [5] ovat:

- *Analyttinen (Analytical)*
- *Tehdas/Tuotantolaitos/Standardilähtöinen (Factory/Standard)*
- *Laadunvalvonta/Laatuajattelu/Laatulähtöinen (Quality Control)*
- *Ketterä (Agile)*
- *Kontekstiohjautunut/Kontekstilähtöinen/Tilannelähtöinen (Context-Driven).*

Esiteltävistä koulukunnista yleisesti vain kontekstiohjautuneen edustajat ja jotkut ketterän koulukunnan edustajista käyttävät koulukunta-nimityksiä [6]. Koulukunta-ajattelua ajaa erityisesti kontekstiohjautunut koulukunta, johon koulukuntajaon kehittäjät kuuluvat [7, s. 261-264]. Muiden koulukuntien edustajat eivät pääsääntöisesti käytä koulukuntajakoa tai tunnusta kuuluvansa tiettyyn koulukuntaan, mutta poikkeuksiakin löytyy [8, s. 353]. Seuraavassa osassa esitellään kunkin koulukunnan arvoja ja näkemyksiä testaukseen.

Analyttisessä koulukunnassa testaus nähdään akateemisesta näkökulmasta matemaattisena teknisenä haasteena, johon yritetään löytää rajatuissa tutkimusolosuhteissa yleispäteviä ratkaisuja ja tekniikoita. Tutkittuja ratkaisuja on sitten tarkoitus hyödyntää käytännössä käyttämällä tarkkoja ja yksityiskohtaisia määrittelyksiä. Analyttisessä koulukunnassa testauksella tarkoitetaan vain tarkasti luotujen määrittelysten täyttymisen verifiointia. Testaus mielletään siis matematiikan ja ohjelmistotieteen sivuhaaraksi ja sen tulee olla objektiivista, perusteellista ja tiukasti määriteltyä. Esimerkkinä analyttisen koulukunnan menetelmästä on koodikattavuuden tutkiminen ja yleispätevien koodikattavuusmittarien kehittäminen.

Tehdaskoulukunnassa testaus nähdään projektin edistymisen mittarina ja erillisenä osana ohjelmistojen tuotantolinjalla. Testaus pitää olla tarkkaan suunniteltua, toistettavaa, ennustettavaa ja kustannustehokasta. Tehdaskoulukunnassa keskitytään systemaattisesti tarkkaan dokumentaatioon ja yksityiskohtaisiin ohjeisiin siitä, miten testaus tulee suorittaa. Testaus on siis tarkkaa sääntöjen seuraamista. Testauksen tuloksia analysoidaan tiettyjen määriteltyjen mittarien avulla, joilla varmistetaan, että määritetyt vaatimukset on testattu ja tulosten perusteella määräytyy projektin valmiustaso ja laatu. Tehdaskoulukunta kannustaa standardointiin, sertifiointiin sekä parhaiden menetelmien (*best practice*) julistamiseen ja käyttöön.

Laadunvalvontakoulukunnassa painotetaan laadunvarmistuksen tärkeyttä, ja testaus on osa laadunvarmistusta eikä varsinaista ohjelmistokehitystä. Testaajat saattavat joutua vahtimaan, että kehittäjät noudattavat määritettyjä prosesseja. Kurinalaisilla prosesseilla pyritään varmistamaan ohjelmiston laatu. Prosessien ylläpitäjänä ja laadunvarmistajana toimiva testaaja sanoo, onko ohjelmisto valmis vai ei.

Ketterässä koulukunnassa testaus nähdään ohjelmoinnin osana ja sen tehtävä on lähinnä varmistaa toiminnon valmistuminen sekä osoittaa, että iteraation tuotos on valmis julkaistavaksi. Testaus pyritään mahdollisuuksien mukaan automatisoimaan. Ketterässä koulukunnassa tehdään mahdollisesti myös tutkivaa testausta, mutta sitä ei arvosteta kovin korkealle eikä siihen tarvittavia taitoja juuri kehitetä tai edes tunnusteta erityisesti.

Kontekstiohjautunut koulukunta perustuu periaatteelle, että minkä tahansa toimintatavan arvo määräytyy aina kontekstin mukaan. Kontekstiohjautuneen koulukunnan seitsemän perusperiaatetta [7, s. 261-262; 9] ovat seuraavat:

- *Minkä tahansa menetelmän arvo riippuu kontekstista.*
- *On olemassa hyviä menetelmiä tietyssä kontekstissa, mutta parhaita menetelmiä (*best practice*) ei ole olemassa.*
- *Yhdessä työskentelevät ihmiset ovat kaikkein tärkein osa minkä tahansa projektin kontekstia.*
- *Projektit muotoutuvat ajan mittaan arvaamattomin tavoin.*
- *Tuote itsessään on ratkaisu johonkin ongelmaan. Jos ongelma ei ole ratkennut, niin tuote ei toimi.*

- *Hyvä ohjelmistojen testaaminen on haastava älyllinen prosessi.*
- *Ainoastaan arviointikyvyn ja taitojen kautta, kun niitä on harjoitettu yhteistyössä läpi koko projektin, olemme kykeneviä tekemään oikeita asioita oikeaan aikaan testataksemme tehokkaasti tuotteitamme.*

Kyseisten peruseriaatteiden painotetaan olevan periaatteita, ei suuntaviivoja tai käskyjä. Kontekstiohjautuneisuus on ajatusmalli tai lähestymistapa testaukseen, ei siis tietty tekniikka. Kontekstiohjatussa testauksessa on paljon ketterän toiminnan piirteitä ja päinvastoin. Molemmissa on tärkeää esimerkiksi ihmis- ja asiakaslähtöinen ajattelu sekä turhan työn minimointi, mutta kontekstiohjattu toiminta ja testaus on laajalaisempaa, koska siinä voidaan käyttää mitä vaan tiettyyn tilanteeseen sopivia tekniikoita tavoitteiden saavuttamiseksi, jopa tarkkaan dokumentoituja perinteisiä menetelmiä [9]. Erityisen tärkeänä pidetään tutkivaa testausta, joka on siis harjoittelematonta ja improvisoitua testausta. Kontekstiohjautunut testaus on monipuolista osaamista vaativaa älyllistä toimintaa. Muunlaista testausta, kuten automatisoituja testejä, nimitetäänkin ennemmin tarkastuksiksi eikä oikeaksi testaukseksi.

Kontekstiohjautunut koulukunta ja etenkin sen yksi alkuperäinen kehittäjä ja äänekäs puolestapuhuja James Bach kyseenalaistaa hyvin vahvasti testausalalla vallitsevia standardeja, termistöä ja näihin perustuvaa testauskirjallisuutta sekä sertifikaatteja. James Bachin mielestä IT-alan sertifikaatit, testauksen sertifikaatit mukaan lukien, esittävät yleensä yhden oikeana pidetyn näkemyksen ja oppijärjestelmän (*doctrine*) asioista. Hänen mielestään tällaisia näkemyksiä on helppo opettaa, tutkia ja sertifioida, mutta ne ovat suorastaan mieltä turruttavia ja tämän vuoksi sopimattomia käytettäväksi testauksessa [5]. Testauksessa on Bachin mukaan kyse juuri kriittisen ajattelun hyödyntämisestä käytännössä. Henkilökohtaisten taitojen korostaminen ja monien työkalujen käytön opettelu ja tehokas hyödyntäminen ovat keskeisiä asioita kontekstilähtöisyydessä.

Kontekstiohjautuneen koulukunnan edustajat ovat tuottaneet paljon testaukseen liittyvää ilmaista Internetissä vapaassa jaossa olevaa materiaalia [9; 10; 11]. He myös kehittävät ja opettavat kontekstiohjautuneen koulukunnan periaatteisiin pohjautuvia konkreettisia menetelmäoppeja (*methodology*) ja tekniikoita, joita voi hyödyntää testaamisessa ja ohjelmistokehityksessä, kuten *Rapid Software Testing* -menetelmäoppi [12] ja tutkivan testauksen hallintaan, mittaamiseen ja hallitsemiseen kehitetty *Session-Based Test Management* [13].

Tätä työtä varten tutkittiin myös joitakin sertifikaatti- ja kurssimateriaaleja [14]. Tutkitussa materiaalissa on toki paljon hyviä yleispäteviäkin asioita, mutta paljon myös menetelmiä, jotka eivät millään tavalla toimisi Intermarketing Oy:n ohjelmistokehityksessä, esimerkiksi tiimin pienen koon takia. Tämän vuoksi useimpia sertifikaattisisältöjä ei voida pitää yleispätevinä ja kaikissa tilanteissa toimivina ratkaisuin. Ohjelmistotestauksessa onkin hyvin tärkeää aktiivisesti tutkia alan julkaisuja, kuten kirjallisuutta, artikkeleita, asiantuntijoiden blogeja sekä muita materiaaleja ja osallistua alan yhteisöjen toimintaan ja esimerkiksi erilaisiin konferensseihin.

Tämän työn tavoitteiden kannalta lähestymistavaksi ja ajatusmalliksi testaukseen valittiin juuri kontekstilähtöisyys. Se sopii Intermarketing Oy:n ohjelmistokehitystiimille hyvin, sillä kyseinen lähestymistapa ei sulje pois mitään varsinaisia työkaluja, tekniikoita ja menetelmiä, eikä toisaalta pakota käyttämään tiimille sopimattomia menetelmiä. Lähestymistapa sopii myös sen vuoksi, että tiimi on tähänkin asti käyttänyt toiminnassaan ketteriä menetelmiä ja ollut hyvin itseohjautuva, mutta tällä hetkellä ohjelmistokehityksessä ei noudateta tiukasti tiettyä menetelmää. Kontekstiohjautuneen koulukunnan ajatusmalli ja lähestymistapa sopii tiimin toimintamalliin hyvin myös sen vuoksi, että useaan erilaiseen asiakasympäristöön kehitetään välillä erittäin nopeassakin tahdissa tarkkaan räätälöityjä ohjelmisto- ja laiteratkaisuja. Juuri kontekstilähtöisyys on tuolloin tärkeää ja pienen ohjelmistokehitystiimin henkilökohtaiset taidot ja yhteistoiminta korostuvat kyseisessä tilanteessa.

On tärkeää painottaa, että koulukunta-ajattelu ja kontekstilähtöinen näkökulma eivät siis tarkoita joidenkin tiettyjen menetelmien poissulkemista tai huonoksi leimaamista. Tässä työssä voidaan siis keskittyä etsimään työn toimeksiantajan ja tavoitteiden kannalta parasta mahdollista etenemistapaa avoimella näkökulmalla eri menetelmiin. Tässäkin työssä on tutkittu useasta eri näkökulmasta kirjoitettua lähdemateriaalia. Olenaisista tämän työn tavoitteiden kannalta on juuri se, että tässä työssä ei ole valittu etenemistavaksi esimerkiksi yhden tietyn ketterän tai perinteisen menetelmän parhaaksi määrittämää (*best practice*) tapaa. Tämän vuoksi pystytään heti alkuun välttämään työn toimeksiantajan tilanteeseen epäsoivia, mahdollisesti kalliiksi ja tehottomaksi paljastuvia menetelmiä.

3 Testiautomaatio

Kuten ohjelmistotestauksesta yleisesti, myös testiautomaatiosta on alalla useita erilaisia näkökantoja. Testiautomaatiolla tarkoitetaan yleensä tietokoneen automaattisesti suorittamaa testausta, eli automaattisten testien suoritusta. Ohjelmistotestauksen tavoin myös testiautomaatio on erittäin laaja aihealue, ja automatisoinnin hyödyntämiseen on olemassa useita erilaisia tekniikoita ja työkaluja. Tässä insinööriyön osioissa esitellään testiautomaatioon liittyviä oletuksia, mahdollisuuksia ja haasteita. Tarkoituksena on lopulta valita työn tavoitteiden kannalta hyödyllisin automaation hyödyntämisen käyttökohde sekä tekniikka, jota lähdetään tutkimaan tarkemmin.

3.1 Ongelmat

Useiden ohjelmistotestauksen kokeneiden ammattilaisten mukaan testiautomaatioon liitetään usein erilaisia vääriä oletuksia ja epärealistisia odotuksia [7, s. 95-127; 15; 16; 17; 18]. Kuten ohjelmistotestauksen näkökantojen käsittelyssä tuli aiemmin tässä työssä ilmi, jotkut pitävät testejä vain sarjana toimintoja ja testaus tarkoittaa näiden toimintojen toistamista kerta toisensa jälkeen. Jos näkökanta testaukseen on tämänkaltainen, saatetaan myös usein puhua testauksen täydestä automatisoinnista tai automaatiota painotetaan testausprosessin tärkeimpänä tavoitteena [19, s. 171-176]. Täyden testiautomaation haavekuvissa automaattiset testit löytävät sovelluksesta nopeasti kaikki virheet ja kattavat kaiken oleellisen kustannustehokkaasti. Ihmisen tehtäväksi testauksessa jää vain automaation toteuttaminen ja valvominen. Tavoitteena on, että ihmisen korvaaminen automatisoinnilla nopeuttaa testausta ja poistaa ihmisen tekemät virheet testauksessa. Tämän vuoksi manuaalista tutkivaa testausta ei välttämättä arvosteta tai tehdä ollenkaan. Manuaalista testausta saatetaan vertailla suoraan automaattisen kanssa testauksen nopeudessa ja kustannuksissa olettaen, että molemmilla tyyleillä testataan samoja asioita.

Tällaisen automaatioajattelun taustalla saattaa yksinkertaisesti olla kokemattomuus tai tietämättömyys [17, s.1], sillä testiautomaatio on testaamisen osa-alueena edelleen melko uusi ja siihen liittyviä haasteita ei välttämättä ymmärretä täysin [20, s. 78-79]. On myös mahdollista, että testaukseen ja automaatioon ei panosteta samalla vakavuudella kuin tuotteen kehittämiseen. Yleistä tietämättömyyttä saatetaan myös käyttää hyödyksi kauppaamalla muille kalliita automaatoratkaisuja ja työkaluja suurin lupauksin [17].

Nykyään testauskulttuuri ja automaatioon suhtautuminen on onneksi jo muuttunut parempaan suuntaan ja nykyään onkin olemassa paljon ilmaisia avoimen lähdekoodin automaatiotyökaluja. On kuitenkin tärkeää ymmärtää, että ohjelmistotestauksen alalla on edelleen tahoja, joiden mielestä kaikki testaus pitäisi pyrkiä automatisoimaan [19, s. 171-176]. Jos testauksesta ja automatisoinnista ei tiedetä tarpeeksi, niin saatetaan tehdä kalliita epäonnistumisia, kun innostutaan täydellisiltä kuulostavista automaatiotratkaisuksista.

Kaiken kattavaan testauksen automatisointiin ei pitäisi pyrkiä, sillä se on käytännössä mahdotonta tai aivan liian kallista, eikä silti korvaisi manuaalista testausta. Usein ohjelmistot ja niiden määrittelyt muuttuvat ja testiautomaation ylläpidosta saattaa tulla mahdoton tehtävä. Esimerkiksi pienetkin muutokset ohjelmistossa saattavat hajottaa suuren määrän testejä. Toisaalta myös erittäin helposti ylläpidettävän täyden testiautomaation suunnittelu ja toteuttaminen on todennäköisesti todella kallis ja lopulta mahdoton urakka. Myöskään kaikkea testausta, jota voidaan automatisoida, ei kannata automatisoida [18]. Tiettyjen asioiden automatisointi ei tuota välttämättä juuri ollenkaan lisäarvoa testaukselle ja ohjelmiston laadulle, varsinkaan jos automatisointi kuluttaa paljon resursseja testauksen muilta alueilta. Pahimmillaan turhaan suoritettavat automatisoidut testit saattavat ennemminkin johtaa todella pahasti harhaan, jos ne menevät jatkuvasti onnistuneesti läpi ja niihin luotetaan liikaa. On mahdollista, että testeissä on virheitä, eivätkä ne testaakaan oikeita asioita, joten vakavia ohjelmiston virheitä saattaa päästä testeistä läpi aiheuttaen myöhemmin entistä enemmän haittaa.

3.2 Mahdollisuudet

Testiautomaatiosta on hyvin toteutettuna toki paljonkin hyötyä testauksessa. Parhaimmillaan testiautomaatio toimii ihmisen toimesta suoritettavan testauksen tukena etenkin sellaisissa tehtävissä, jotka ovat aina samalla tavalla toistettavia, mutta kuitenkin tärkeitä tarkastuksia esimerkiksi regressiotestauksessa [19, s. 201-205; 21, s. 97-98]. Automaattisella testauksella voidaan myös testata asioita, joita manuaalisesti ei edes pystytä testaamaan, kuten ajan nopeamman kulumisen tai useiden käyttäjien simuloinnit. Useimmiten automatisoiduilla tarkastuksilla ei kuitenkaan löydetä uusia virheitä, ainakaan läheskään yhtä paljon kuin manuaalisella testauksella [15]. Ohjelmistojen testauksessa manuaalinen testaus on erittäin tärkeää, sillä yleensä juuri tutkivalla testauksella löydetään uusia virheitä testattavasta järjestelmästä. Tämä on huomattu myös

Intermarketing Oy:n ohjelmistokehitystiimissä. Usein erikoisimmat ja mahdollisesti täysin odottamattomat sovelluksen virheelliset käyttäytymiset ovat tulleet ilmi manuaalisen testauksen yhteydessä. Siksi automatisointia pyritäänkin käyttämään tutkivan testauksen tukena. Automatisointi voi myös esimerkiksi vapauttaa aikaa uusien testausmenetelmien ja parempien testitapausten suunnitteluun. Automaattisia ja manuaalisia testejä ei pidä myöskään rinnastaa suoraan keskenään, sillä niillä tulisi testata usein eri asioita. Tästä johtuen automaation hyödyllisyyttä ei myöskään tulisi mitata suoraan kaiken testaukseen menevän ajan säästössä. Testiautomaation suunnitteluun, toteuttamiseen ja ylläpitoon kuluu myös aikaa.

Automaation hyödyntämisen onnistumiseen vaikuttaa merkittävästi alustan testattavuus ja testausprosessien tila. Sovelluksen testattavuus tulisi ottaa huomioon jo kehitysvaiheessa ja koko testausstrategian tulee olla hyvässä kunnossa ennen automaation laajempaa toteutusta. Automaatiosta voi olla paljonkin hyötyä testauksen tukena, kun se suunnitellaan hyvin ja siihen sitoudutaan. Onkin tärkeää osata tehdä oikeita valintoja tekniikoissa ja automatisoinnin kohteissa, sillä huonosti toteutettu automaatio voi heikentää koko testiprosessia. Esimerkiksi todella vaikeasti ylläpidettävät automaattiset testit ja niiden jatkuva päivittäminen saattavat kuormittaa ohjelmistotestausta jopa enemmän kuin samojen testien suorittaminen manuaalisesti.

Tässä insinööriyössä on todettu jo aiemmin, että testaus tarkoittaa näkökulmasta riippuen eri asioita. Vaikka tässäkin työssä puhutaan yleisesti testauksesta, niin erityisesti testiautomaatiosta puhuttaessa on hyvä tiedostaa, että testin automatisoinnilla tarkoitetaan lähes aina tiettyjen tarkastusten suorittamisen automatisointia. Esimerkiksi käyttöliittymän läpi ajettavat automatisoidut regressiotestit eivät varsinaisesti testaa mitään, vaan ne ovat ennalta määritettyjä tarkastuksia. Erityisesti kontekstiohjatun testauksen koulukunnan edustajat painottavat testaus (*testing*) ja tarkastus (*checking*) -termien eroa [22]. Yhtenä perusajatuksena on, että testaaminen on ihmisen suorittamaa älyllistä toimintaa, jossa tarkastustyökalut voivat olla vain tukena. Työkalut voivat siis periaatteessa suorittaa tarkastusta ihmisten puolesta.

Vaikka automaatiotyökalut eivät voi ainakaan kontekstiohjatun testauksen näkökulmasta varsinaisesti suorittaa oikeaa testausta, niin toisaalta työkalujen avulla voidaan tehdä paljon muutakin kuin pelkkää tarkastusta. Automatisoinnin hyödyntämisellä testauksessa ei tarkoiteta vain testitapausten automaattista suorittamista. Testiautomaatiolla tarkoitetaan kaikkea automatisointityökalujen antamaa testauksen tukea testiprosessi-

en eri osa-alueilla [23]. Automatisointityökaluja voidaan hyödyntää esimerkiksi seuraavilla osa-alueilla:

- testien generoinnissa
(tietokanta-, aineisto- ja skriptigeneraattorit)
- järjestelmän konfiguroinnissa
(testiympäristön alustus)
- simuloinnissa
(järjestelmä- ja laiterajapintasimulaattorit)
- testien suorituksessa
(kuormitustestaus, tietoturvatestaus)
- luotaimissa
(staattiset analyysit, järjestelmän parametrien monitorointi)
- oraakkeleissa
(testien onnistumisen analysointi, virheiden havainnointi)
- aktiviteettien tallennuksessa ja kattavuusanalyyseissa
(mitä tehtiin, mitä testattiin)
- testien hallinnassa
(testauksen hallintajärjestelmät, testien tulosten tallennus).

Kyseisiä automatisointimahdollisuuksia ja työkaluja tullaan erittäin todennäköisesti tutkimaan ja hyödyntämään jossain vaiheessa Intermarketing Oy:n ohjelmistokehityksessä. Automaatiota hyödynnetään jo nykyäänkin esimerkiksi simulaattoreissa ja kuormitustestauksessa. Tällä hetkellä on kuitenkin kaikista tärkeintä kehittää järjestelmätestausta, joten tässä työssä päätettiin perehtyä seuraavaksi tarkemmin testitapausten automaattisessa suorituksessa käytettäviin erilaisiin skriptausmenetelmiin.

4 Testiskriptaus

Yleisellä tasolla skriptit ovat tavallisia tekstitiedostoja tai merkkijonoja, jotka sisältävät komentolauseita. Skriptit ovat siis komentosarjoja ja eräänlaisia tietokoneohjelmia, joiden avulla voidaan automatisoida tietokoneen suorittamia tehtäviä. Erilaisia skriptejä ja niiden sisältämiä komentoja käytetään yleensä jossain erityisessä tarkoituksessa, ja ne toimivat vain tietyssä rajatussa ympäristössä ja tietyillä ohjelmilla tai tulkeilla. Erilaisia skriptaustyyplejä ja niiden käyttötarkoituksia varten on siis erilaisia skriptikieliä eli komentosarjakieliä. Komentosarjakieliä käytetään esimerkiksi suorittamaan käyttöjärjes-

telmien komentoja erilaisissa komentotulkeissa, kuten Linux-jakeluiden bash-oletuskomentotulkissa tai Windows-käyttöjärjestelmien cmd-komentotulkissa. Alun perin komennoilla tarkoitettiin juuri käyttöjärjestelmän komentoja. Nykyään skripteillä tarkoitetaan kuitenkin muitakin kuin vain käyttöjärjestelmien komentosarjoja. Nykyisiä skriptikieliä ovat esimerkiksi Python, Perl, Ruby ja PHP. Yhteistä näille skriptikielille on se, että niitä ei tarvitse erikseen kääntää konekieliseksi ohjelmaksi kuten varsinaisten ohjelmointikielien lähdekoodeja. Skriptikieli onkin nykyään lähes synonyymi tulkattavalle kielelle [24].

Tässä työssä skriptauksella tarkoitetaan lähinnä testiskriptausta. Testiskriptejä käytetään usein käsikirjoituksena tai komentosarjana tietyn testin toimenpiteiden suorittamiseksi. Toisin sanoen testiskripti on testitapausten tai sen osan suorituksen toimintasekvenssin kuvaus tai suoritusohje. Myös testiskriptien muoto ja käyttötarkoitus määräytyvät niiden erityisen tarkoituksen, ympäristön ja käytettävän tulkin mukaan. Tämän insinööriyön tavoitteen kannalta ei ole tarkoituksenmukaista luoda sovelluksille mahdollisimman kattavia testiskripti- ja testitapauskokoelmia, vaan ainakin aluksi olisi tarkoitus luoda skriptit kaikista tärkeimpien ja usein toistettavien testitapausten ja tarkastusten suorittamiseksi. Automatisointia on tarkoitus hyödyntää erityisesti käyttöliittymätestauksessa sekä käyttöliittymän kautta suoritettavassa järjestelmä- ja hyväksymistestauksessa, etenkin regressiotesteissä. Yhtenä tavoitteena on myös se, että jatkossa myös muut kuin ohjelmointitaitoiset pystyisivät tarvittaessa ainakin suorittamaan ja muokkaamaan, sekä mahdollisesti jopa luomaan uusia skriptejä.

Käyttöliittymän läpi suoritettavan testauksen automatisointiin käytetään käytännössä aina jonkinlaista skriptausmenetelmää. Tässä työn osiossa esitellään kyseisiä menetelmiä. Jokaisesta menetelmästä käydään läpi menetelmän toimintamalli, edut, ongelmat sekä soveltuvuus Intermarketing Oy:n tämänhetkisiin tarpeisiin. Tiedot skriptausmenetelmistä perustuvat pääasiassa Robot Framework -testiautomaatiokehityksen pääkehittäjän Pekka Klärckin DI-työn [25] ja testiautomaatiota esittelevän kalvosarjan [26] pohjalle. Esittelyissä on käytetty myös kyseisen kalvosarjan eri menetelmiä kuvaavia kuvia ja esimerkkejä, joitakin hieman muokaten. Testiautomaatiokehityksen määrittelmä ja käyttötarkoitus esitellään myöhemmin yhden esiteltävän menetelmän, avainsanaohjatun testauksen yhteydessä. Kaikki esiteltävät menetelmät ovat:

- *nauhoitus ja toisto*
- *lineaarinen skriptaus*

- *modulaarinen skriptaus*
- *aineisto-ohjattu testaus*
- *avainsanaohjattu testaus.*

Skriptausmenetelmät esitellään tässä työssä ohjelmistotestauksen evoluution mukaisessa järjestyksessä. Ohjelmistotestauksen evoluutioaskeleet voidaan jakaa myös kolmeen päatasoon [27], jotka ovat:

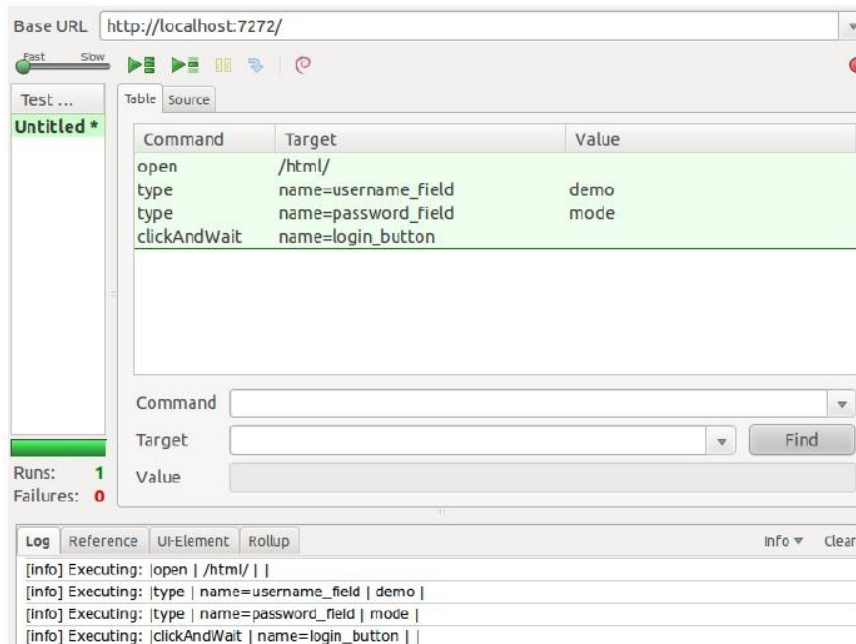
- *taso 1. manuaalinen testaus*
- *taso 2. skriptaus ja automaattinen testien suoritus*
- *taso 3. mallipohjainen testaus.*

Evoluution tasot ja niihin sisältyvät tekniikat on hyvä tuntee, sillä kaikkia testausmenetelmiä voidaan mahdollisesti hyödyntää testauksessa kontekstista riippuen. Kaikki tässä työssä esiteltävät menetelmät kuuluvat toisen tason evoluutioaskeleeseen, sillä ensimmäisen tason manuaalista testausta on juuri tarkoitus automatisoida ja kolmannen tason mallipohjaisessa testauksessa lähestymistapa testiskriptaukseen on hyvin erilainen. Mallipohjaisessa testauksessa testitapauksia ja skriptejä generoidaan automaattisesti tiettyjen järjestelmän toimintoja kuvaavien mallien pohjalta [27]. Mallipohjainen testaus on testausmenetelmänä toki hyvin mielenkiintoinen ja lupaava, mutta kyseinen tyyli ei kuitenkaan sovi tässä tapauksessa haettuun käyttötarkoitukseen, koska tarkoitus on pääasiassa automatisoida tarkkaan ennalta määritettyjen tarkastusten suoritus. Vaikka esitellyistä toisen tason menetelmistä toisia voidaan pitää kehittyneempinä kuin toisia, niin kaikille saattaa löytyä käyttöä tämän työn tavoitteiden kannalta. Tarkoitus on valita näistä menetelmistä parhaiten soveltuva vaihtoehto. Lopuksi esitellään myös valittua menetelmää ja työn tavoitteita parhaiten tukeva automaatiotyökalu, jota tullaan käyttämään konseptin todennuksessa.

4.1 Nauhoitus ja toisto

Nauhoitus ja toisto -tyylissä tallennetaan järjestelmän kanssa tapahtuneet interaktiot skriptiksi ja tätä samaa skriptiä toistetaan kerrasta toiseen automaattisena testinä. Testin tarkoitus siis on ajaa tietty sekvenssi läpi ja tarkastaa, että saatiinko oikea tulos

skriptiin lisättyjen tarkastuspisteiden avulla. Kuvassa 1 on esimerkki nauhoitus ja toisto -työkalusta.



Kuva 1. Nauhoitus ja toisto -toiminto Selenium IDE -työkalussa [26].

Edut

Tärkein etu nauhoitus ja toisto -tyylissä on se, että testien luonti ja suoritus on nopeaa ja helppoa. Lisäksi kynnys kokeilla tällaista testausta on alhainen, koska tässä tyylissä ei tarvita ohjelmointitaitoja välttämättä ollenkaan. Tämä johtuu siitä, että nauhoitus ja toisto tapahtuvat usein valmiilla työkalulla.

Ongelmat

Suurin ongelma tässä tyylissä on se, että luodut testit ovat erittäin hauraita ja vaikeita tai mahdottomia ylläpitää. Usein yksikin muutos käyttöliittymässä saattaa rikkoa kaikki testit. Tuloksena nauhoituksesta saadaan useita huonosti strukturoituja ja dokumentoituja tarkastussekvenssejä, joita ei voida käyttää muualla uudestaan. Testattavan järjestelmän pitää myös olla valmis ennen kuin voidaan aloittaa automatisointi. Tämä tyyli ei siis tue skriptien tekemistä etukäteen.

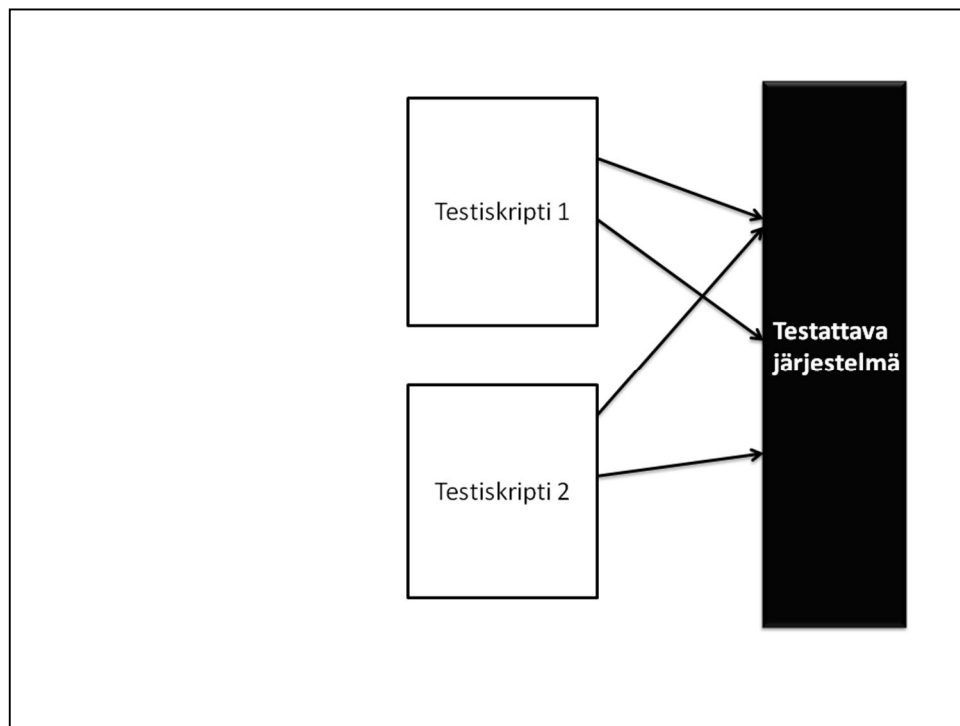
Soveltuvuus

Nauhoitus ja toisto -tyyli ei sovellu tämän työn tarpeisiin, sillä tarkoitus on pyrkiä modulaarisuuteen ja hyvään ylläpidettävyyteen. Näitä tavoitteita ei voida saavuttaa kyseisellä tyylillä, varsinkaan sen takia, että työn toimeksiantajalla on useita eri sovelluksia, joita

pitäisi testata erilaisilla konfiguraatioilla asiakasympäristöstä riippuen. Nauhoitus ja toisto -tyyli ei sovellu laajamittaiseen ja ylläpidettävään automaatioon ollenkaan.

4.2 Lineaarinen skriptaus

Linearisessa skriptauksessa ohjelmoidaan testiskripti suorittamaan tietty sekvenssi. Syöte- ja tulostiedot ovat osa skriptiä, ja skriptit ovat suoraan yhteydessä testattavaan järjestelmään tai sen osaan. Tämä yhteys on kuvattu kuvassa 2. Lineaariset skriptit ovat käytännössä samoja, joita tallennus ja toisto -työkalut tuottavat, mutta käsin kirjoitettuja.



Kuva 2. Lineaarista skriptausta havainnollistava kuvaus [26].

Edut

Etuna lineaarisessa skriptaustyyliässä on erityisesti se, että skriptien luominen on nopeaa sekä joustavaa ja skriptaamiseen voidaan käyttää yleisiä maksuttomia skriptauskieliä. Koodiesimerkissä 1 on kuvattu esimerkki Python-skriptauskiellä luodusta web-käyttöliittymätestistä.

```

from selenium import selenium

se = selenium('localhost', 4444, '*firefox', 'http://localhost:7272')
se.start()
se.open('/html')
se.set_speed(1000)
se.type('username_field', 'demo')
se.type('password_field', 'mode')
se.click('login_buttton')
se.wait_for_page_to_load(5000)
if se.get_title() == 'Welcome Page':
    print 'Login test passed.'
else:
    print 'Login test failed!'
se.stop()

```

Koodiesimerkki 1. Pythonilla luotu web-käyttöliittymätestin esimerkki [26].

Ongelmat

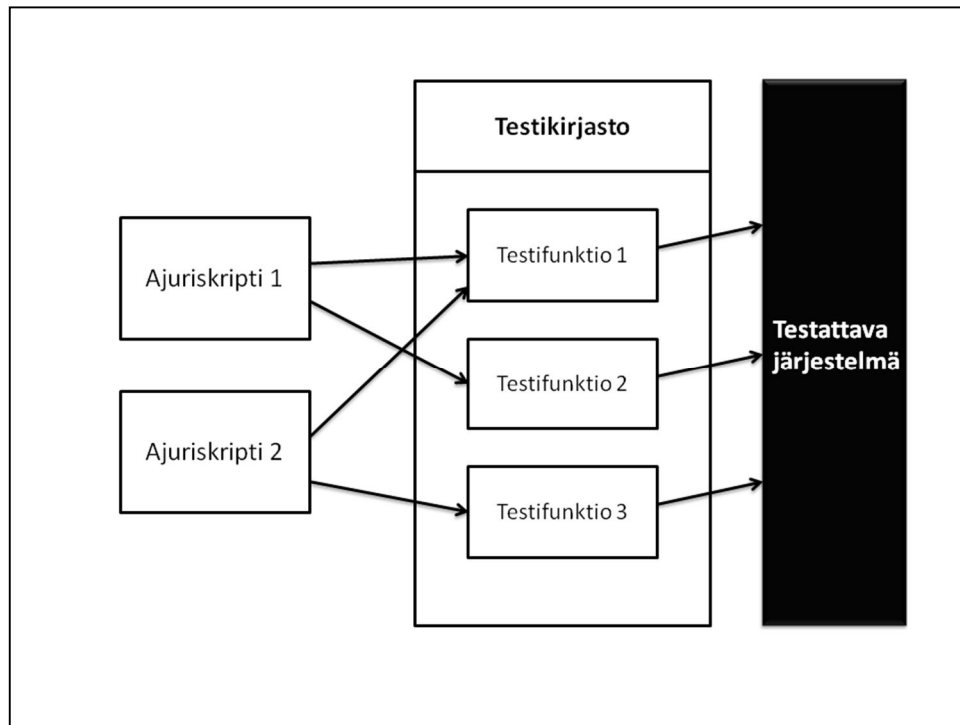
Ongelmana tässä tyyliässä voidaan pitää sitä, että skriptien luontiin vaaditaan ohjelmointitaitoja. Tässä tyyliässä joudutaan lisäksi luomaan useita testiskriptejä, joita ei voida käyttää muualla uudestaan. Tämän modulaarisuuden puutteen vuoksi ylläpito on todella vaikeaa ja yksikin muutos järjestelmässä saattaa rikkoa kaikki skriptit.

Soveltuvuus

Tällä hetkellä useissa yrityksen sovelluksissa osa yksikkö- ja integraatiotesteistä on toteutettu periaatteessa tällä tavalla, mutta kyseinen tapa ei sovellu tämän työn varsinaisena päämääränä olevaan automaation järjestelmä- ja hyväksymistestauksessa, etenkin huonon ylläpidettävyyden vuoksi.

4.3 Modulaarinen skriptaus

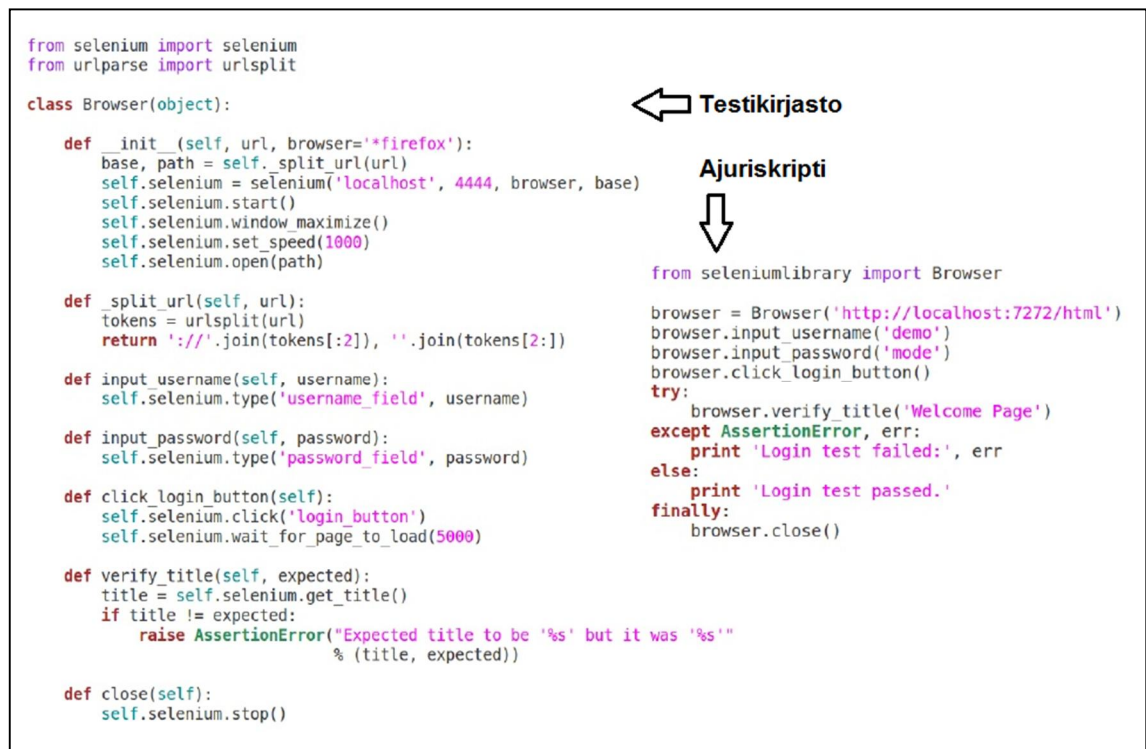
Modulaarisessa skriptauksessa varsinaisessa testiskriptissä eli ajuriskriptissä määritetään testin rakenne vastaavalla tavalla kuin lineaarisessa skriptauksessa, mutta järjestelmään yhteydessä olevat toiminnot on irrotettu testin kulkua kuvaavista skripteistä erilliseen testikirjastoon. Tämä on kuvattu kuvassa 3. Testin kulkua kuvaavissa ajuriskripteistä voidaan siis kutsua geneerisiä testifunktioita testikohtaisesti aina tarvittaessa, vaikka useaan kertaankin.



Kuva 3. Havainnollistava kuvaus modulaarisesta skriptausmenetelmästä [26].

Edut

Merkittävänä etuna modulaarisessa skriptauksessa on juuri modulaarisuus ja sen avulla saavutettava koodin uudelleenkäytettävyys. Tämän ansioista uusien testien luonti nopeutuu ja myös ylläpidettävyys paranee, koska muutokset keskittyvät pienempiin alueisiin. Ajuriskriptit ovat melko yksinkertaisia ja helppoja ymmärtää, ja myös kokeuttomampi ohjelmoija pystyy luomaan uusia skriptejä. Koodiesimerkissä 2 on esitetty esimerkit testikirjastosta ja sitä käyttävästä ajuriskriptistä.



Koodiesimerkki 2. Esimerkki testikirjastosta ja sen testifunktioita käyttävästä ajuriskriptistä [26].

Ongelmat

Ongelmana modulaarisessa skriptauksessa on testikirjastojen luomisen työläys ja se, että skriptien ymmärtämiseen vaaditaan edelleen ohjelmointitaitoja. Vaikka funktiot ovat erillään testikirjastossa, niin testeissä käytettävä aineisto sisältyy edelleen ajuriskripteihin. Tämän vuoksi uusia testejä varten tarvitaan aina uusi ajuriskripti.

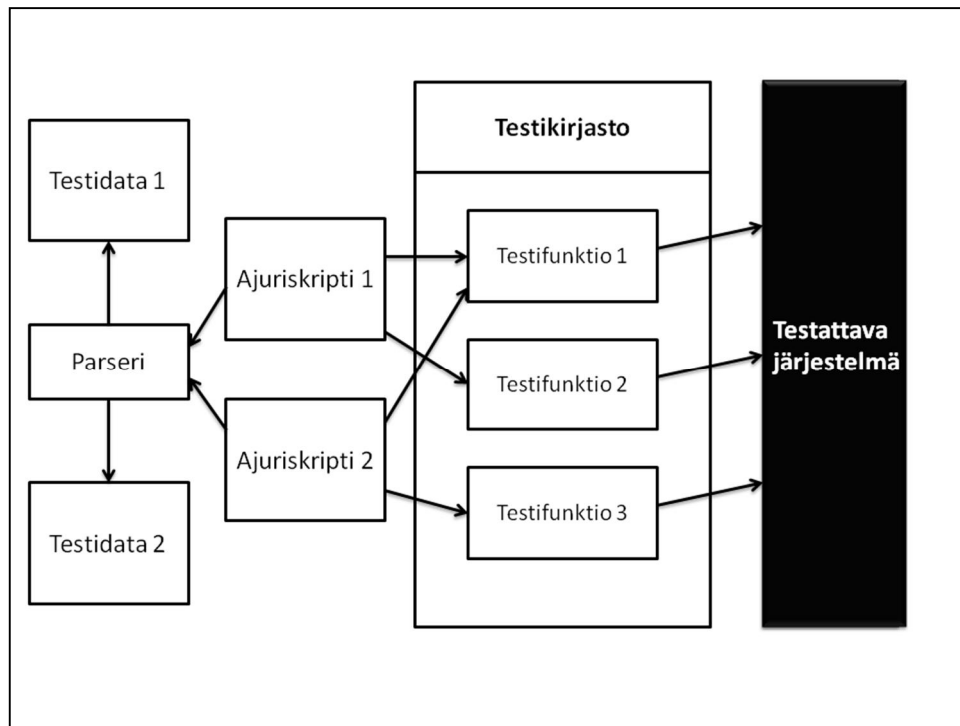
Sovellettuuus

Modulaarinen skriptaus on jo huomattavasti parempi tyyli kuin lineaarinen skriptaus, mutta ei tarpeeksi modulaarinen ja ylläpidettävä. Tässä tyylissä vaaditaan myös edelleen ohjelmointitaitoja, joten kukaan ohjelmistokehitystiimin ulkopuolinen ei voisi osallistua testien luontiin.

4.4 Aineisto-ohjattu testaus

Aineisto-ohjattu testaus on käytännössä muuten samanlaista kuin modulaarinen skriptaus, mutta testattavaan järjestelmään yhteydessä olevien testifunktioiden lisäksi myös testeissä käytetty aineisto on erillään ajuriskripteistä. Ajuriskriptit käyttävät testikohtaista eri aineistoja parserin eli jäsentimen kautta. Tämä on havainnollistettu kuvassa 4.

Aineiston irrotus ajuriskriptistä mahdollistaa sen, että täysin sama ajuriskripti voi suorittaa usean samankaltaisen testin eri aineistolla.



Kuva 4. Aineisto-ohjattua testausta esittävä kuvaus [26].

Edut

Aineisto-ohjatussa testauksessa on samat edut kuin modulaarisessa skriptauksessa, mutta uusien testien luonti ja vanhojen muokkaus on entistäkin helpompaa ja nopeampaa. Uusien testien luonti ei vaadi enää välttämättä ohjelmointitaitoja, joten ylläpitovastuuta voidaan jakaa testaajien ja ohjelmoijien kesken. Testaajat voivat myös esimerkiksi olla vastuussa testiaineistosta ja ohjelmoijat muusta varsinaisesta automaatiokoodista. Aineisto-ohjatut testit voidaan kuvata esimerkiksi yksinkertaisessa ja helposti ymmärrettävässä taulukkomuodossa. Taulukossa 1 on kuvattu esimerkkejä samankaltaisista yksinkertaisista laskutoimituksia suorittavista testeistä, joiden rakenne on kaikissa sama, mutta testeissä tarkastetaan kuitenkin eri asioita vain aineistoa muuttamalla.

Taulukko 1. Esimerkki samankaltaisista testeistä, joissa käytetään eri aineistoja [26].

	A	B	C	D	E
1	Test Case	Number 1	Operator	Number 2	Expected
2	Add 01	1	+	2	3
3	Add 02	1	+	-2	-1
4	Sub 01	1	-	2	-1
5	Sub 02	1	-	-2	3
6	Mul 01	1	*	2	2
7	Mul 02	1	*	-2	-2
8	Div 01	2	/	1	2
9	Div 02	2	/	-2	-1

Ongelmat

Yhtenä ongelmana aineisto-ohjatussa testauksessa voidaan pitää sitä, että testitapaukset ovat edelleen hyvin samankaltaisia, jos voidaan vaihtaa ainoastaan skriptissä käytettävää dataa. Jos halutaan uutta toiminnallisuutta, esimerkiksi lisäämällä taulukon 1 laskutoimituksiin toinen operaattori ja kolmas numero, niin joudutaan luomaan uusi skripti, mikä vaatii jälleen ohjelmointitaitoja. Automatisoinnin toteuttamisessa joudutaan näkemään alussa mahdollisesti paljonkin vaivaa, kun luodaan parsereita ja muita uudelleenkäytettäviä komponentteja.

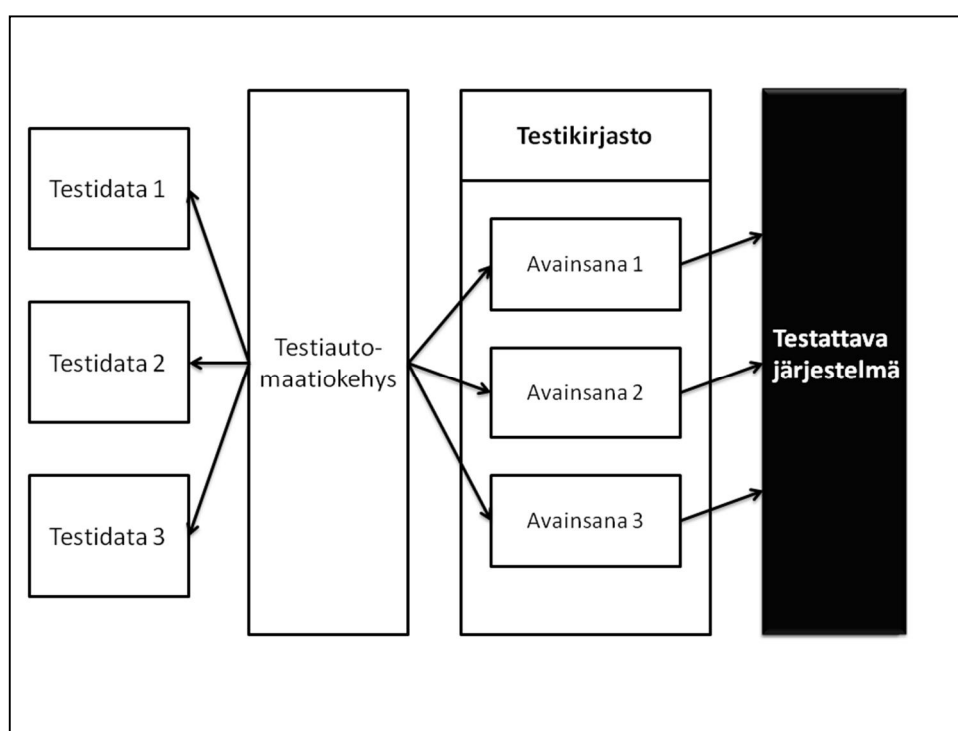
Soveltuvuus

Aineisto-ohjattu testaus on jo melko hyvä ja kattava vaihtoehto, mutta testeihin ei edelleenkään saada kovin paljon vaihtelua tarpeeksi helposti. Automaation laajempi käyttöönnotto ja uudenaisten testien luominen on turhan työlästä ja vaatii edelleen ohjelmointitaitoja.

4.5 Avainsanaohjattu testaus

Avainsanaohjattu testaus on edistyneempi muoto aineisto-ohjatusta testauksesta. Tässä menetelmässä myös toimintaohjeet siitä, miten dataa käytetään, eli avainsanat, on irrotettu testiskripteistä. Avainsanat (*keyword*, *action word*) ovat periaatteessa funktioita tai metodeita. Avainsanat ja niihin liittyvä testiaineisto siis ajavat testien suoritusta. Yksi avainsanaohjattu testi voi koostua useasta toiminnosta ja yhtä toimintoa taas kuvaa yksi avainsana sekä sille mahdollisesti annettavat parametrit. Avainsanat voivat siis koostua toisista avainsanoista.

Kaikki luodut testit voidaan suorittaa käyttäen yhtä testiautomaatiokehystä. Testiautomaatiokehys on kokoelma oletuksia, muotomäärittäjiä, konsepteja, kirjastoja ja työkaluja, joita käytetään automatisoinnin tukena. Avainsanaohjatussa testauksessa testiautomaatiokehys huolehtii testien suorituksesta ja testiajojen tulosten raportoinnista, sekä liittymismekanismeista testattavaan järjestelmään. Testiautomaatiokehys saattaa tarjota myös valmiita yleiskäyttöisiä avainsanoja testien tekemiseen. Avainsanaohjatussa testauksessa ei ole siis tarvetta luoda tai ylläpitää ajuriskriptejä. Ajuriskriptissä kuvatun testin rakenteen tilalla on mahdollisesti useamman abstraktiotason avainsanoista koostuva testin kuvaus. Kuvassa 5 on havainnollistettu testiautomaatiokehysten sijoittuminen tässä menetelmässä.



Kuva 5. Avainsanaohjatussa testauksessa testiautomaatiokehys hoitaa myös ajuriskriptien tehtäviä [26].

Edut

Avainsanaohjatussa testauksessa on useita etuja. Testin tekijä voi koota haluamansa testin vapaasti olemassa olevista yleiskäyttöisistä avainsanoista. Eri abstraktiotason avainsanoja käyttämällä testeistä voidaan myös luoda helposti luettavia, kuten koodiesimerkissä 3 näkyvässä testissä. Korkeamman tason avainsanat voivat esimerkiksi kuvata testitapauksen askeleita selkokielellä ja seuraavan matalamman tason avainsanat käsittelevät esimerkiksi testattavaa järjestelmää ja aineistoa. Selkeitä korkean tason testejä voi suunnitella, vaikka ei osaisi ohjelmoida. Tämän vuoksi myös testiauto-

maation työtaakkaa voidaan jakaa esimerkiksi testaajien ja ohjelmoijien välillä tehokkaasti. Hyvin toteutetut avainsanat ja testisetit mahdollistavat myös erittäin hyvän automaattiorakenteiden ylläpidettävyyden. Lisäksi sopivilla avainsanoilla voidaan myös tarvittaessa luoda aineisto-ohjattuja testejä.

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

Koodiesimerkki 3. Esimerkki avainsanaohjatusta testistä [26].

Ongelmat

Ongelmana avainsanaohjatussa testauksessa voidaan pitää sitä, että automaattisten testien toteutus on mahdollisesti työläämpää toteuttaa laajassa mittakaavassa verrattuna aineisto-ohjattuun testaukseen, jos avainsanoja ei ole valmiiksi saatavilla. Lähinnä menetelmän tehokkaan hyödyntämisen opetteluun ja uusien avainsanojen tarkkaan suunnitteluun saattaa mennä enemmänkin aikaa.

Soveltuvuus

Avainsanaohjattu testaus vaikuttaa sopivan hyvin haettuun käyttötarkoitukseen ja laajempaankin käyttöön. Se on tutkituista menetelmistä kaikkein kehittynein ja ylläpidettävvin. Ehkä suurin tätä menetelmää tukeva asia on se, että nykyään on olemassa avoimen lähdekoodin testiautomaatiotyökaluja tai -kehyksiä, joissa on jo valmiina useita suoraan käytettävissä olevia avainsanakirjastoja ja monia eri toimintoja. Tämän vuoksi päästään heti luomaan ainakin joitain testejä. Tutkimusten perusteella tässä työssä käytettäväksi menetelmäksi valitaan siis avainsanaohjattu testaus. Avainsanaohjattua testausta tukeva testiautomaatiokehys esitellään seuraavaksi.

4.6 Robot Framework

Robot Framework on yleiskäyttöinen avoimen lähdekoodin testiautomaatiokehys, joka on suunniteltu etenkin hyväksyntätason testaukseen ja hyväksymistestilähtöiseen kehi-

tykseen (*Acceptance test -driven development, ATDD*). Se on julkaistu Apache 2.0 -lisenssillä ja on Nokia Siemens Networks (nykyinen Nokia Solutions and Networks) kehittämä ja sponsoroima. Robot Framework on toteutettu Pythonilla, mutta se tukee täysin myös Jythonia, joka on Python-kielen Java-toteutus. Robot Framework hyödynää avainsanaohjattua testausta ja sen käyttämä helppokäyttöinen taulukkorakenteinen syntaksi mahdollistaa testien kirjoittamisen esimerkiksi useissa helposti luettavissa tekstimuodoissa tai HTML-kuvauskielellä luodussa taulukkomuodossa. Kuvassa 6 on esitetty yksinkertainen avainsanaohjattu testijoukko (testikokoelma, *Test Suite*) yksinkertaisessa välilyöntierottelua käyttävässä tekstimuodossa, sekä sama testijoukko taulukkomuodossa.

*** Settings ***				
Library	OperatingSystem			
*** Variables ***				
\${MESSAGE}	Hello, world!			
*** Test Cases ***				
My Test				
[Documentation]	Example test			
Log	\${MESSAGE}			
My Keyword	/tmp			
Another Test				
Should Be Equal	\${MESSAGE}	Hello, world!		
*** Keywords ***				
My Keyword				
[Arguments]	\${path}			
Directory Should Exist	\${path}			

Setting	Value	Value	Value
Library	OperatingSystem		
Variable	Value	Value	Value
\${MESSAGE}	Hello, world!		
Test Case	Action	Argument	Argument
My Test	[Documentation]	Example test	
	Log	\${MESSAGE}	
	My Keyword	/tmp	
Another Test	Should Be Equal	\${MESSAGE}	Hello, world!
Keyword	Action	Argument	Argument
My Keyword	[Arguments]	\${path}	
	Directory Should Exist	\${path}	

Kuva 6. Esimerkki täysin samasta testistä kahdessa eri muodossa [28].

Välilyöntierottelua käyttävässä tekstimuodossa merkkijonoissa voi olla mukana välilyöntejä. Toiminnot, parametrit ja muuttujien arvot erotellaan toisistaan siten, että merkkijonon väli on enemmän kuin yksi välilyönti. Testitapausten ja avainsanojen sisällöt määritellään sisennyksillä. Välilyöntierottelua käyttävä tekstimuoto mahdollistaa testijoukkojen kirjoittamisen erittäin luettavassa ja selkeästi jäsenellyssä muodossa. Kuvan 6 esimerkissä on asetuksissa (*Settings*) otettu käyttöön Robot Frameworkin mukana tuleva *OperatingSystem*-avainsanakirjasto ja muuttujissa (*Variables*) on esitelty yksi muuttuja. Testitapauksia (*Test Cases*) on tässä esimerkissä kaksi kappaletta. Näistä ensimmäisen (*My Test*) lopussa käytetään esimerkin lopussa luotua korkeamman tason avainsanaa (*My Keyword*).

Robot Framework mahdollistaa testiskriptien kirjoittamisen myös aineisto-ohjatun testauksen tai esimerkiksi käyttäytymislähtöisen ohjelmistokehityksen (*Behavior-driven development, BDD*) testien muodossa. Käyttäytymislähtöisen ohjelmistokehityksen testien muotoisesta testistä on esitetty esimerkki taulukossa 2. Kyseistä muotoa nimit-

tään joskus myös Gherkin-kieleksi käyttäytymislähtöisyyttä hyödyntävän testiautomaatiotyökalu Cucumberin käyttämän kielen mukaan.

Taulukko 2. Esimerkki Gherkin-kielisistä avainsanojatuista testeistä [28].

Test Case	Step
Add two numbers	Given I have Calculator open
	When I add 2 and 40
	Then result should be 42
Add negative numbers	Given I have Calculator open
	When I add 1 and -2
	Then result should be -1

Keyword	Action	Argument	Argument
I have \${program} open	Start Program	\${program}	
I add \${number 1} and \${number 2}	Input Number	\${number 1}	
	Push Button	+	
	Input Number	\${number 2}	
	Push Button	=	
Result should be \${expected}	\$(result) =	Get Result	
	Should Be Equal	\$(result)	\${expected}

Robot Frameworkin testien kirjoittamiseen on tarjolla myös RIDE-editori (Robot IDE), jonka avulla testejä voidaan luoda ja myös suorittaa. RIDE-editorin lisäksi Robot Frameworkiin on joko saatavilla, tai se sisältää valmiiksi muitakin hyödyllisiä työkaluja ja liitännäisiä esimerkiksi testien editointiin, suorittamiseen ja dokumentointiin. Robot Framework sisältää valmiiksi useita yleiskäyttöisiä avainsanakirjastoja ja siihen on tarjolla myös runsaasti valmiita hyödyllisiä ulkoisia avainsanakirjastoja. Lisäksi omien kirjastojen luonti on mahdollista ja hyvin yksinkertaista, jos valmiiden kirjastojen tarjoamat avainsanat eivät riitä omissa testeissä. Omia kirjastoja voi luoda helposti Pythonilla tai Javalla. Javalla luotavia kirjastoja voidaan laajentaa myös *JavalibCore*-kirjastolla (Javan JAR-kirjasto), jonka avulla omiin avainsanakirjastototeutuksiin voidaan annotaatioiden avulla esimerkiksi lisätä dokumentaatioita ja nimetä avainsanoja joustavammin ja tarkemmin.

Tätä Insinööriyötä varten tutkittiin pikaisesti myös muita, lähinnä avoimen lähdekoodin testiautomaatiotyökaluja ja -kehyksiä, mutta esimerkiksi useat työkalut on jo saatavilla Robot Frameworkin liitännäisinä. Melko tunnettu testiautomaatiokehys FitNesse [29] on ominaisuuksiltaan myös melko kattava, mutta Robot Framework valittiin, koska siinä on tarjolla kattavammin valmiita toimintoja, ja se on todella helposti laajennettavissa esimerkiksi omilla avainsanakirjastoilla. Yksi tärkeimmistä valmiista avainsanakirjastoista on kattava *SwingLibrary*-avainsanakirjasto, jonka avulla voidaan testata Swing-

käyttöliittymiä. Kyseisestä kirjastosta on merkittävän paljon hyötyä, koska suuressa osassa Intermarketing Oy:n ohjelmistoja on Swing-käyttöliittymä. Tämän kaiken lisäksi Robot Frameworkin dokumentaatio on hyvin kattava ja sisältää paljon havainnollisia esimerkkejä.

5 Konseptin todennus

Tässä insinööriyön osiossa on tarkoitus suunnitella ja toteuttaa testiautomaatiota yhteen Intermarketing Oy:n sovellukseen. Aiemmin tässä työssä valittiin automatisoinnin menetelmäksi avainsanaohjattu testaus. Tarkoitus on toteuttaa konseptin todennuksena (*Proof of concept*) yhteen testitapaukseen yksi automaattisesti suoritettava testi ja yksi puoliautomaattisesti (*semi-automatic*) suoritettava, manuaalisen osuuden sisältävä testi. Molemmat testit ovat siis käyttöliittymän läpi tehtäviä regressiotestejä.

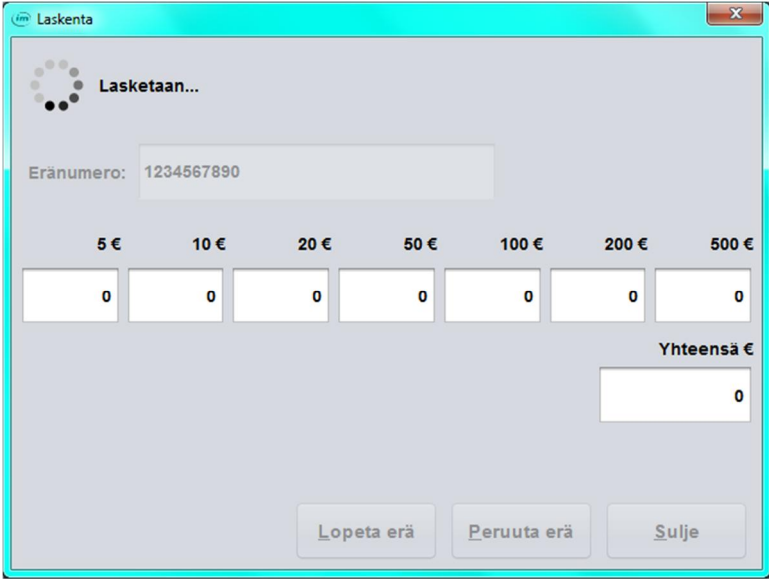
Erittäin merkittävä pohjatyö tätä konseptin todennuksen tekemistä varten on kokemus, jota tämän työn tekijä saanut työskennellessään Intermarketing Oy:n ohjelmistokehitystiimissä ohjelmistotestaajana ja -kehittäjänä usean vuoden aikana. Tuona aikana yrityksen ohjelmisto- ja laiteratkaisut sekä asiakaskohtaiset ympäristöt ja järjestelmät ovat tulleet hyvin tutuiksi. Tämän vuoksi yrityksen tarpeet ja konseptin todennuksen tavoite ovat hyvin selkeät. Lisäksi tämän työn tekijä on ollut alusta asti kehittämässä konseptin todennuksen kohteena olevaa sovellusta, joten kyseinen ohjelmisto- ja laiteratkaisu on koodia ja laitteistoa myöten tuttu.

Konseptin todennuksen toteutuksessa on tarkoitus pysyä eri tekniikoiden ja työkalujen tarjoamien mahdollisuuksien tarkastelussa. Kuten aiemmin tässä työssä automatisointia ja avainsanaohjattua testausta tarkastellessa todettiin, suunnittelu ja käyttöönotto ovat usein melko työläitä projekteja. Laajemman automatisoinnin tarkka suunnittelu jätetään konseptin todennuksesta siis pois, mutta lopuksi pohditaan hieman myös jatkokehitysmahdollisuuksia.

5.1 Määritelmä

Konseptin todennus tehdään ohjelmisto- ja laiteratkaisuun, jossa on Java Swing-käyttöliittymällinen sovellus, jonka avulla käytetään seteleitä lajittelevaa ja laskevaa

laitetta. Yhden laskentaerän tulokset tallennetaan XML-muodossa aina uuteen aikaleimalla nimettyyn tiedostoon. Automatisoitavana testitapauksena konseptin todennuksessa on yhden onnistuneen laskentaerän suoritus. Kuvassa 7 näkyy sovelluksen laskentanäkymä laskennan ollessa käynnissä.



The screenshot shows a window titled "Laskenta" with a loading indicator and the text "Lasketaan...". Below this, there is a text input field for "Eränumero:" containing the value "1234567890". Underneath, there are seven columns representing different denominations: 5 €, 10 €, 20 €, 50 €, 100 €, 200 €, and 500 €. Each column has a corresponding input field, all of which contain the value "0". To the right of these fields is a label "Yhteensä €" followed by an input field containing "0". At the bottom of the window, there are three buttons: "Lopeta erä", "Peruuta erä", and "Sulje".

Kuva 7. Sovelluksen laskentanäkymä.

Testi sisältää sovelluksen käynnistyksen, laskentaerän eli transaktion aloituksen, suorittamisen ja päättämisen, sekä laskennan tulosten tarkastuksen XML-lokista. Koko laskentaerän kulku sovelluksessa on kuvattu ruutukaappauksin liitteessä 1 (*Laskentaerän suoritus*) ja XML-tiedoston sisältö esimerkkiaineistolla on esitetty koodiesimerkissä 4.

```

<transaction>
  <id>1234567890</id>
  <status>committed</status>
  <starttime>2013-11-13 20:55:00</starttime>
  <endtime>2013-11-13 20:55:43</endtime>
  <mix>
    <notes>
      <note>
        <value>5</value>
        <quantity>12</quantity>
      </note>
      <note>
        <value>10</value>
        <quantity>5</quantity>
      </note>
      <note>
        <value>20</value>
        <quantity>10</quantity>
      </note>
      <note>
        <value>50</value>
        <quantity>5</quantity>
      </note>
    </notes>
  </mix>
</transaction>

```

Koodiesimerkki 4. Esimerkki laskentaerän XML-lokitiedostosta.

Testiin sisältyy manuaalisena toimenpiteenä setelien asetus laitteeseen. Tämän manuaalisen toimenpiteen vuoksi kyseessä oleva testitapaus sopii hyvin konseptin todennuksen kohteeksi. Juuri manuaalisesti suoritettavia laitteiden käyttötoimenpiteitä sisältävät järjestelmätestit ovat tämän työn kannalta yksi olennainen tutkimuksen kohde. Koska testattavaan sovellukseen on toteutettu myös fyysistä laitetta simuloiva simulaattori, niin testitapaukselle on mahdollista toteuttaa myös täysin automaattinen käyttöliittymätesti. Tällaisista täysin automatisoiduista testeistä on hyötyä esimerkiksi regressiotestinä, jotka suoritetaan sovellukseen tehtävien muutosten yhteydessä.

5.2 Toteutus

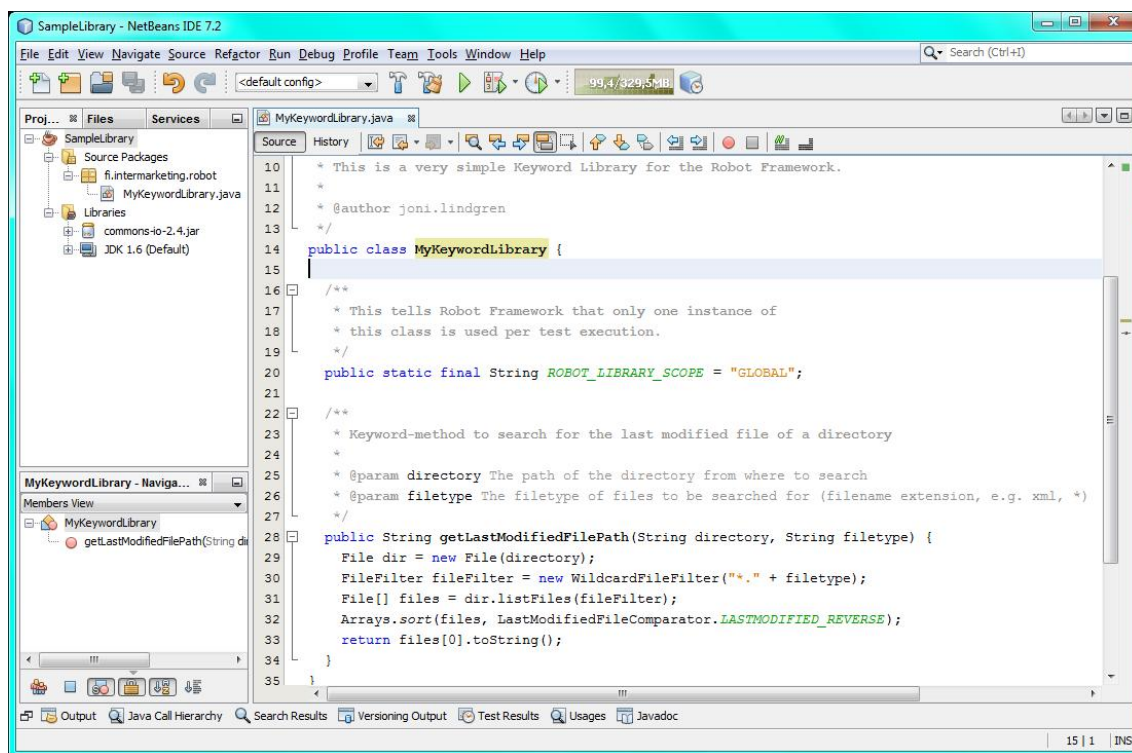
Koska konseptin todennuksen tavoitteena on testata Java-sovellusta, niin Robot Frameworkia käytetään Jythonin kanssa. Ympäristön asennus oli melko yksinkertainen toimenpide. Robot Frameworkin dokumentaatioissa on selkeät ohjeet asennusten suorittamiseen eri tavoilla [30]. Jythonista asennettiin versio 2.7b1 ja Robot Frameworkista versio 2.8.1. Jython-asennus suoritettiin valmiin asennusohjelman avulla (*jython-installer-2.7-b1.jar*). Lisäksi Jython-asennuksen bin-hakemisto täytyi lisätä Windowsin PATH-ympäristömuuttujaan. Tämän jälkeen ladattiin Robot Frameworkin lähdekoodijulkaisupaketti (*robotframework-2.8.1.tar.gz*). Seuraavaksi Robot Frameworkin purettu

hakemisto avattiin komentoikkunassa ja enää tarvitsi suorittaa asennuskomento (*jython setup.py install*).

Asennuksen jälkeen oli mahdollista aloittaa Robot Frameworkin käytön opettelu. Erityisesti tässä vaiheessa Robot Frameworkin kattava dokumentaatio ja esimerkit osoittautuivat erittäin hyödyllisiksi. Konseptin todennuksen tavoitteiden perusteella testin toteuttamiseen tarvittiin seuraavat avainsanakirjastot:

- *XML* (mukana Robot Frameworkin julkaisupaketissa)
- *Collections* (mukana Robot Frameworkin julkaisupaketissa)
- *SwingLibrary* (ladattava erikseen Robot Frameworkin sivuilta).

Lisäksi konseptin todennusta varten luotiin myös yksi oma avainsanakirjasto Javalla. Kyseinen kirjasto sisältää yhden avainsanan, jonka avulla haetaan automatisoitavassa testitapauksessa viimeisimmän laskentaerän XML-lokitiedosto. Avainsanalle annetaan parametreiksi hakemistopolku ja haettavan tiedoston tiedostotyyppi, minkä jälkeen avainsana palauttaa kyseissä hakemistossa viimeksi muokatun tiedoston tiedostopolun. Avainsanakirjasto on erittäin yksinkertainen toteuttaa. Kuvassa 8 näkyy tehdyn avainsanakirjaston Java-toteutus kokonaisuudessaan.



Kuva 8. Ruutukaappaus itse tehdyn avainsanakirjaston NetBeans-projektista.

Kuvassa 7 näkyvä *MyKeywordLibrary*-luokka vastaa Robot Frameworkissa avainsanakirjastoa ja luokan *getLastModifiedFilePath*-metodi vastaa avainsanaa. Kyseisen metodin nimi toimii sellaisenaan avainsanana, mutta avainsanamuodossa sallitaan myös isojen ja pienten kirjainten vaihdot sekä välilyöntien käyttö, kunhan kahta välilyöntiä ei käytetä peräkkäin. Projektin koonnin jälkeen avainsanakirjasto on heti valmis käytettäväksi. Itse tehdyn avainsanakirjaston Java-projektin pakattu JAR-tiedosto otetaan Robot Frameworkin testeissä käyttöön samalla tavalla kuin valmiit avainsanakirjastotkin. Kirjaston nimessä täytyy tosin olla luokan hakemistorakenne kokonaan näkyvissä.

Konseptin todennuksessa luotiin lopulta yksi normaalin onnistuneen laskentaerän suorittava testi. Avainsanaohjattu testi luotiin Robot Frameworkin yksinkertaisessa välilyöntierottelua käyttävässä tekstimuodossa. Testi jaettiin kahteen eri tekstitiedostoon, testikokoelmaan (*SmokeSuite.txt*) ja sitä laajentavaan laskentaerään liittyviä korkeamman tason avainsanojen toteutuksia sisältävään resurssitiedostoon (*transaction.txt*). Koodiesimerkissä 5 näkyy *SmokeSuite*-tiedoston sisältö.

```

*** Settings ***
Force Tags      release 1.2.0
Library         SwingLibrary
Resource        _keywords/transaction.txt
Suite Setup     Start Test Application

*** Test Cases ***
Normal Transaction
    [Tags]      smoke

    Open Transaction View
    Set Valid Transaction Id
    Start Transaction
    Wait For Money Counting To Finish
    End Transaction
    Close Transaction View
    Verify Transaction Result From Log File

*** User Keywords ***
Start Test Application
    Start Application    fi.intermarketing.counter.gui.ImCounter
    Select Main Window

```

Koodiesimerkki 5. Testikokoelmatiedoston sisältö.

SmokeSuite-tiedosto sisältää kolme osiota: asetukset (*Settings*), testitapaukset (*Test Cases*) ja omat avainsanat (*User Keywords*). Testitapaukset osioissa on yksi testitapaus, jonka nimi on *Normal Transaction*. Kyseinen testitapaus koostuu itse tehdystä korkeamman tason avainsanoista. Testitapaukselle on määritetty yksi tagi (*smoke*) testitapauksen tasolla. Tagit auttavat testien lajittelussa ja esimerkiksi priorisoinnissa. Lisäksi koko testikokoelmalle on määritetty asetukset-kohdassa kaikkiin kyseisessä testikokoelmassa oleviin testitapauksiin periytyvä tagi (*release 1.2.0*), jonka avulla voidaan

esimerkiksi määrittää tarkkaan jokaisessa eri julkaisussa toimivat testit. Muita tagien käyttötarkoituksia voisi olla esimerkiksi lajittelu positiivisiin ja negatiivisiin testitapauksiin, tai niiden merkitseminen tehtävähallintatyökalujen (JIRA, Agilefant) tehtävä- tai virhetunnisteilla.

Asetuksissa esiteltävällä *Suite Setup* -avainsanalla voidaan alustaa testiajo suorittamalla argumenttina annettu avainsana, joka on tässä tapauksessa samassa tiedostossa omissa avainsanoissa määritetty *Start Test Application*. Kyseisellä avainsanalla käynnistetään testiajon aluksi testattava sovellus käyttämällä *SwingLibrary*-avainsanakirjaston *Start Application* -avainsanaa, jolle annetaan argumentiksi testattava Java-sovelluksen pääluokka (*fi.intermarketing.counter.gui.ImCounter*). Tämän jälkeen avautuva sovelluksen päänäkymä valitaan *Select Main Window* -avainsanalla aktiiviseksi näkymäksi. *SwingLibrary* tarjoaa siis avainsanoja Swing-käyttöliittymien komponenttien käsittelyyn. Kyseisen avainsanakirjaston hyödyntäminen ja käytön opettelu oli melko helppoa, sillä kirjasto tarjoaa myös avainsanan sovelluksen komponenttien selvittämiseen (*List Components in Context*). Tämä tarkoittaa käytännössä sitä, että testattavan sovelluksen koodiin ei tarvitse koskea eikä sitä tarvitse tuntea, mutta silti voidaan kirjoittaa käyttöliittymätestejä. *SwingLibrary*n kaltaiset avainsanakirjastot ovat juuri tästä syystä yksi Robot Frameworkin merkittävistä edusta.

SmokeSuite-tiedoston testitapauksessa (*Normal Transaction*) käytettävät omatekoiset korkean tason avainsanat on määritetty erillisessä resurssitiedostossa, joka tuotu asetuksissa testikokoelman käyttöön (*_keywords/transaction.txt*). Tämän kyseisen avainsana-resurssitiedoston sisältö näkyy kokonaisuudessaan liitteessä 2 (*Konseptin todennuksen avainsanat*). Kyseisen tiedoston sisältö esitellään seuraavaksi selvyuden vuoksi kolmessa osassa, koodiesimerkeissä 6, 7 ja 8.

```

*** Settings ***
Library XML
Library Collections
Library fi.intermarketing.robot.MyKeywordLibrary

*** Variables ***
${COUNTING_DELAY}      10
${RESULT_LOG_DIRECTORY} transactions
${RESULT_LOG_FILETYPE}  xml
${VALID_TRANSACTION_ID} 1234567890
${INVALID_TRANSACTION_ID} 1234567

# Expected denomination
${SEUR}      12
${10EUR}     5
${20EUR}     10
${50EUR}     5
${100EUR}    0
${200EUR}    0
${500EUR}    0

```

Koodiesimerkki 6. Avainsana-resurssitiedoston asetukset ja yleiset muuttujat.

Koodiesimerkissä 6 näkyy transaction-tiedoston alkuosa, jossa on asetukset-osiossa esitelty käytettävät avainsanakirjastot: *XML*, *Collections* ja itse tehty *MyKeywordLibrary*. Tämän jälkeen on esitelty avainsanojen käyttämät muuttujat, jotka ovat:

- laskentaviive (*COUNTING_DELAY*)
- laskennan tulosten hakemistopolku (*RESULT_LOG_DIRECTORY*)
- laskennan tulosten tiedostomuoto (*RESULT_LOG_FILETYPE*)
- oikeanlainen eränumero (*VALID_TRANSACTION_ID*)
- vääränlainen eränumero (*INVALID_TRANSACTION_ID*)
- odotettu laskennan tulos setelilajeittain (*5-500EUR*).

Kyseisiä muuttujia voidaan siis käyttää avainsanoissa esimerkiksi testikohtaisessa konfiguroinnissa tai odotettuina tuloksina. Koodiesimerkissä 7 näkyy SmokeSuite-tiedoston testitapauksessa käytettyjen korkeamman tason avainsanojen toteutukset transaction-tiedostossa. Kyseiset avainsanat ja niiden toiminta ovat seuraavat:

- *Open Transaction View* (laskentanäkymän avaaminen)
- *Close Transaction View* (laskentanäkymän sulkeminen)
- *Set Valid Transaction Id* (oikeanlaisen eränumeron syöttö)
- *Set Invalid Transaction Id* (vääränlaisen eränumeron syöttö)
- *Start Transaction* (laskentaerän aloitus)

- *End Transaction* (laskentaerän lopetus)
- *Wait For Counting To Finish* (laskentaviiveen odotus)
- *Verify Transaction Result From Log File* (XML-lokitiedoston tarkastus).

Toteutuksissa on käytetty testikokoelmasta periytynyttä *SwingLibrary*-kirjastoa käyttöliittymätoimintoihin, kuten tekstikenttien täyttämisiin (*Insert Into Text Field*), painikkeiden painamisiin (*Push Button*) ja tarkistamisiin (*Button Should Be Enabled/Disabled*) sekä näkymien valitsemisiin (*Select Dialog/Main Window*). Avainsanoissa on käytetty myös *Collections*-kirjastoa List- ja Dictionary-tyyppisten muuttujien käsittelyyn.

```

*** Keywords ***
Open Transaction View
    Push Button            cmdCashCount
    Select Dialog          Laskenta
    Button Should Be Disabled btnStart

Close Transaction View
    Push Button            btnClose
    Select Main Window

Set Valid Transaction Id
    Insert Into Text Field txtBatchNumber  ${VALID_TRANSACTION_ID}

Set Invalid Transaction Id
    Insert Into Text Field txtBatchNumber  ${INVALID_TRANSACTION_ID}

Start Transaction
    Button Should Be Enabled btnStart
    Push Button            btnStart

End Transaction
    Button Should Be Disabled btnClose
    Push Button            btnStop

Wait For Money Counting To Finish
    Sleep                  ${COUNTING_DELAY}

Verify Transaction Result From Log File
    ${EXPECTED_DENOMINATION}= Create Dictionary  5=${5EUR}      10=${10EUR}
    ...                               20=${20EUR}      50=${50EUR}
    ...                               100=${100EUR}  200=${200EUR}
    ...                               500=${500EUR}
    ${RESULT_LOG_FILE}= Get Last Modified File Path  ${RESULT_LOG_DIRECTORY}  ${RESULT_LOG_FILETYPE}
    Transaction Result Should Be Correct  ${RESULT_LOG_FILE}  ${EXPECTED_DENOMINATION}

```

Koodiesimerkki 7. Korkean tason avainsanojen toteutukset.

Laskentaerän XML-lokitiedoston sisältö tarkastetaan koodiesimerkissä 7 näkyvän alimmaisen avainsanan (*Verify Transaction Result From Log File*) avulla, jossa odotetut laskennan tulokset muutetaan Dictionary-muuttujaksi ja haetaan itse Javalla tehdyn avainsanakirjaston avainsanalla (*Get Last File From Path*) uusin XML-laskentatiedosto. Tämän jälkeen odotettu denominaatio ja uusimman laskentatiedoston polku annetaan

parametreinä alemman tason avainsanalle (*Transaction Result Should Be Correct*), joka on kuvattu koodiesimerkissä 8. Varsinainen XML-laskentatiedoston käsittely ja tarkistaminen tapahtuu juuri tällä avainsanalla. Kyseisessä avainsanassa käytetään XML-avainsanakirjaston avainsanoja, mutta myös Robot Frameworkin sisäisen kirjaston tarjoamaa FOR-silmukkaa. Avainsanoihin saadaan siis tarvittaessa myös lisää toiminnallisuutta esimerkiksi ehto- ja toistolauseilla.

```
Transaction Result Should Be Correct
  [Arguments]    ${RESULT_LOG_FILE} ${EXPECTED_DENOMINATION}
  ${root}=      Parse XML  ${RESULT_LOG_FILE}
  @{values}=    Get Elements Texts  ${root}  mix/notes/note/value
  @{quantities}= Get Elements Texts  ${root}  mix/notes/note/quantity
  ${length}=    Get Length  ${values}
  :FOR  ${i}  IN RANGE  ${length}
  \  Log  i:  ${i}
  \  ${value} = Get From List  ${values}  ${i}
  \  ${quantity} = Get From List  ${quantities}  ${i}
  \  ${expected_quantity} = Get From Dictionary  ${EXPECTED_DENOMINATION}  ${value}
  \  Should Be Equal  ${expected_quantity}  ${quantity}
```

Koodiesimerkki 8. Matalamman tason avainsanan toteutus.

Robot Frameworkin tekstimuotoisen testikokoelmatiedoston voi suorittaa helposti esimerkiksi cmd-komentoikkunassa. Komentoikkunassa pitää ensin asettaa erillisten avainsanakirjastojen ja testattavan sovelluksen JAR-tiedostot Javan ympäristömuuttujaan (*CLASSPATH*). Testiajo käynnistetään tämän jälkeen komennolla, johon voidaan suoritettavan tiedoston nimen lisäksi lisätä erilaisia hyödyllisiä argumentteja. Robot Framework luo testiajon jälkeen automaattisesti yksityiskohtaisen raportin ja lokin HTML-muodossa. Kuvassa 9 näkyy ruutukaappauksina esimerkit epäonnistuneen ja onnistuneen testiajon raporteista sekä onnistuneen testiajon lokista. Epäonnistuneissa testeissä toiminnon tai tarkastuksen epäonnistunut suoritus näkyy raportoinnissa erittäin yksityiskohtaisesti. Myös testattavan sovelluksen Java-poikkeukset näkyvät testin suoritustiedoissa.

SmokeSuite Test Report

Summary Information

Status: 1 critical test failed

Start Time: 20131117 02:45:00 GMT
End Time: 20131117 02:45:17 GMT
Elapsed Time: 00:00:16.970
Log File: log.html

SmokeSuite Test Report

Summary Information

Status: All tests passed

Start Time: 20131117 02:45:00.475
End Time: 20131117 02:45:17.445
Elapsed Time: 00:00:16.970
Log File: log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:14	
All Tests	1	1	0	00:00:14	

Test Details

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
release 1.2.0	1	1	0	00:00:14	
smoke	1	1	0	00:00:14	

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
SmokeSuite	1	1	0	00:00:17	

SmokeSuite Test Log

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:14	
All Tests	1	1	0	00:00:14	

Test Execution Log

TEST SUITE: SmokeSuite

Full Name: SmokeSuite
Source: C:\POC\SmokeSuite\src
Start / End / Elapsed: 20131117 02:45:03.475 / 20131117 02:45:17.445 / 00:00:16.970
Status: 1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed

TEST CASE: Normal Transaction

Full Name: SmokeSuite\Normal Transaction
Tag: release 1.2.0, smoke
Start / End / Elapsed: 20131117 02:45:03.366 / 20131117 02:45:17.443 / 00:00:14.374
Status: PASS (critical)

KEYWORD: transaction:Open Transaction View
KEYWORD: transaction:Set Valid Transaction Id
KEYWORD: transaction:Start Transaction
KEYWORD: transaction:Wait For Money Counting To Finish
Start / End / Elapsed: 20131117 02:45:05.600 / 20131117 02:45:15.607 / 00:00:10.007
KEYWORD: Balloon Sleep (COUNTING_DELAY)

Procedural Note: Pauses the test executed for the given time.
Start / End / Elapsed: 20131117 02:45:05.600 / 20131117 02:45:15.395 / 00:00:10.005
02:45:15.603 INFO #tag: 10 seconds

Kuva 9. Esimerkkejä konseptin todennuksessa toteutettujen Robot Framework testien automaattisesta raportoinnista.

Testiajon käynnistävän komennon argumentilla voi esimerkiksi asettaa hakemistopolun kyseisille tiedostoille. Kaikista hyödyllisimpänä ominaisuutena on kuitenkin mahdollisuus asettaa suoritettavien testien sisäisiä muuttujien arvoja. Tästä ominaisuudesta on hyötyä esimerkiksi konfiguraatioiden ja testausympäristöjen muuttuessa. Kyseistä ominaisuutta hyödynnetään tässä konseptin todennuksessa siten, että testitapauksen osittain manuaalisen version suorituksessa setelien laskentaan asetetaan isompi aikaviive kuin simulaattoria vasten ajettaessa. Tällä tavalla voidaan käyttää täysin samaa Robot Framework -testiä sekä täysin automaattisessa testissä että osittain manuaalisessa testissä. Testiajojen helppoa suorittamista varten luotiin kaksi komentojonotiedostoa (.bat) joiden sisällöt on esitetty koodiesimerkissä 7. Molemmissa on määritetty myös automaattisesti muodostettaville testiajojen tuloksille omat hakemistonsa.

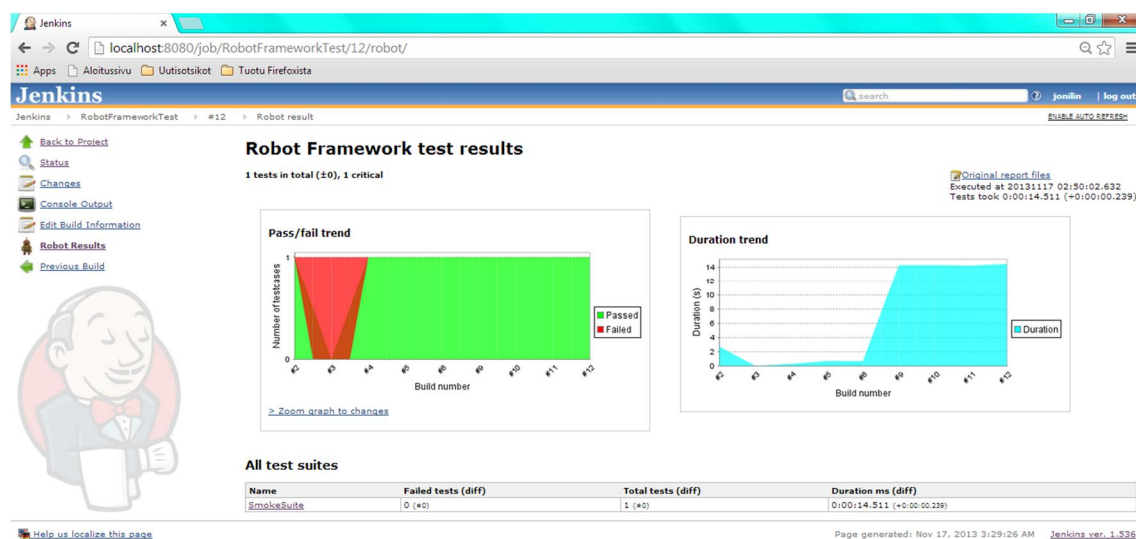
```
@echo off
set CLASSPATH=swinglibrary-1.7.0.jar;im-counter-gui.jar;SampleLibrary.jar
jybot --outputdir _results/Automatic SmokeSuite.txt

@echo off
set CLASSPATH=swinglibrary-1.7.0.jar;im-counter-gui.jar;SampleLibrary.jar
jybot --outputdir _results/SemiAutomatic --variable COUNTING_DELAY:30 SmokeSuite.txt
```

Koodiesimerkki 9. Automaattisen ja puoliautomaattisen testiajojen käynnistävien bat-tiedostojen sisällöt.

Testien automatisoinnissa on usein myös tärkeää saada suoritettua täysin automaattiset testit helposti ja automaattisesti. Usein automaattiset regressiotestit suoritetaan osana jatkuvaa integraatiota, esimerkiksi säännöllisin väliajoin ja aina, kun ohjelmaan tehdään muutoksia. Tällä tavalla mahdollisia virheitä löydetään paljon nopeammin jo

sovelluksen uuden version kehityksen yhteydessä. Kyseistä tarkoitusta varten asennettiin kokeiltavaksi avoimen lähdekoodin jatkuvaan integraatioon tarkoitettu työkalu Jenkins. Muita vastaavia työkaluja ei lähde tutkimaan kovin tarkasti, koska Jenkins oli jo osittain tuttu työkalu ja siihen on suoraan tarjolla Robot Framework -liitännäinen. Jenkins asennettiin toistaiseksi vain paikallisesti ja vain tätä konseptin todennusta varten. Jenkinsin ja Robot Framework -liitännäisen asennukset ja käyttönotot sujuivat erittäin helposti. Jenkinsistä asennettiin versio 1.536 ja liitännäisestä versio 1.3.1. Kuvassa 10 on esitetty esimerkki Jenkinsin Robot Framework -testiajojen tuloksista.



Kuva 10. Jenkinsin Robot Framework -testiajojen tuloksia.

Robot Framework -testiajon suorittamiseksi Jenkinsiin tarvitsi vain luoda uusi liitännäistä käyttävä tehtävä, jossa tarvitsi vain määrittää konseptin todennuksessa aiemmin luodun automaattisen testiajon suorittavan komentojonotiedoston sijainti. Jenkinsin kautta pääsee myös tarkastelemaan suoraan kuvassa 9 esiteltyjä automaattisesti luotuja raportteja.

5.3 Tulokset ja jatkokehitys

Kaiken kaikkiaan konseptin todennusta voidaan pitää onnistuneena. Valitut työkalut ja tekniikat osoittautuivat toimiviksi ja testitapauksen automatisointi onnistui tavoitteiden mukaisesti. Lisäksi toteutettu ratkaisu on korkean tason avainsanojen ansiosta myös erittäin ylläpidettävä. Automaation laajempi hyödyntäminen vaatii toki enemmän suunnittelua ja aikaa, mutta pelkästään jo konseptin todennuksen testistä ja sen avainsanoista voidaan suoraan luoda uusia tärkeitä testejä kyseiselle sovellukselle. Pienellä

muokkauksella korkean tason avainsanoja voidaan käyttää myös useissa muissa Intermarketing Oy:n sovelluksissa. Tietyissä Intermarketing Oy:n sovellusratkaisuissa monimutkaisia samalla tavalla toistettavia testitapauksia sisältävien kattavien manuaalisten regressiotestien suorittaminen saattaa kestää jopa useamman tunnin. Automaatisoimalla sovellusten usein samanlaisena toistettavia manuaalisia regressiotestejä voidaan säästää huomattavasti aikaa. Taulukossa 3 näkyy konseptin todennuksessa automatisoidun yhden testitapauksen eri testaustyylien suoritusajoja.

Taulukko 3. Normaali laskentaerä -testitapauksen suoritusajoja eri testaustyyliä.

Suoritustapa	Suoritusajo (s)	Fyysinen laite	Simulaattori
Manuaalinen	60	X	
Manuaalinen	40		X
Puoliautomaattinen	25	X	
Automaattinen	15		X

Vertailutaulukossa 3 näkyvät suoritusajat koskevat siis vain yhden sovelluksen yhtä testitapausta. Eri ohjelmisto- ja laiteratkaisuille on useita testitapauksia ja lisäksi jokaiselle asiakaskohtaiselle toteutukselle on usein vielä omia testien variaatioita. Aikaa tullaan säästämään siis moninkertaisesti, kun useampia testitapauksia ja kokonaisia testikokoelmia saadaan automatisoitua. Joidenkin sovellusten monimutkaisemmissa testitapauksissa aikaa säästyy vielä enemmänkin automaation avulla, sillä mitä monimutkaisempi toimintojen sarja suoritetaan, sitä useammin ihminen joutuu pysähtymään ja miettimään seuraavaa askelta. Manuaalisten toimintojen suhteellinen lisääntyminen testitapauksessa ei juuri vähennä automaation tuomaa nopeutusta ja arvoa puoliautomaattisissa testeissäkään, sillä usein manuaaliset toimenpiteet ovat lopulta melko yksinkertaisia ja nopeita suorittaa. Täysin manuaalisissa regressiotesteissä aikaa kuluu juuri siihen, että testaja joutuu jatkuvasti miettimään testin kulkua ja tarkkailemaan sovelluksen tilaa sekä toimintaa. Automaattiset testit ovatkin huolellisesti toteutettuna parhaimmillaan huomattavasti ihmistä tarkempia. Muutaman sekunnin kestävässä automaattisessa testiajossa voidaan jatkuvasti tarkkailla esimerkiksi sovelluksen tilaa sekä käyttöliittymän komponenttien toimintaa ja näkyvyyttä. Intermarketing Oy:n ohjelmisto- ja laiteratkaisuilla käsitellään usein suuria määriä rahaa, joten on myös hyvin tärkeää, että laskentojen tuloksia ja laitteiden saldoja tarkkaillaan jatkuvasti. Ihmiseltä vastaavat tarkastukset vievät huomattavasti enemmän aikaa, ja on myös mahdollista, että tarkastuksia jää jopa tekemättä epähuomiossa.

Hyödyntämällä konseptin todennuksessa jo toimiviksi todettuja tekniikoita, sekä esimerkiksi testiympäristön automaattista konfigurointia, voidaan siis useamman tunnin kestäneet regressiotestit suorittaa mahdollisesti minuuteissa ja silti jopa entistä tarkemmin. Säästettyä aikaa kannattaisi käyttää esimerkiksi tutkivaan manuaaliseen testaukseen ja sen kehitykseen sekä uusien testitapausten suunnitteluun. Testitapausten ja automaattisten testien huolellinen suunnittelu on erittäin tärkeää testiautomaation hyödyntämisessä, sillä testien antamat tulokset täytyy pystyä verifioimaan luotettavasti.

Robot Framework osoittautui erittäin monipuoliseksi yleiskäyttöiseksi automaatiotyökaluksi, jota tullaan jatkossa hyödyntämään Intermarketing Oy:n ohjelmistotestauksessa. Lisäksi se on avoimen lähdekoodin työkalu, eli ilmaisuuden lisäksi se on myös laajennettavissa. Robot Frameworkin avainsanoajatut testit ovat helposti luettavissa ja ylläpidettävissä ja niiden käyttö on melko helppo oppia jopa ilman ohjelmointitaitoja. Robot Framework sisältää myös useita lupaavia kirjastoja, joita voisi hyödyntää jatkossa yrityksen muissakin huomattavasti monimutkaisemmissa sovelluksissa. Esimerkiksi *Selenium2Library*-kirjastoa voidaan käyttää web-käyttöliittymien testaukseen ja *Database*-kirjastoa tietokantojen tarkkailuun ja käsittelyyn. Lisäksi Robot Frameworkia voidaan hyödyntää myös esimerkiksi testiympäristöjen konfiguroinnissa. Automaattiset testit olisi tärkeää viedä jatkossa osaksi ohjelmistojen versionhallintaa ja testien suoritus osaksi jatkuvaa integraatiota Jenkinsin Robot Framework -liitännäisen avulla. Kun testit on helposti hallittavissa ja suoritettavissa, saadaan niistä suurin hyöty irti. Esimerkiksi selkeidenkin ongelmien ja ”rikkinäisten” julkaisuehdokkaiden (Release Candidate) havaitseminen nopeutuu huomattavasti. Myös ohjelmiston julkaisuvalmiutta pystytään seuraamaan paremmin.

Tämän insinööriyön ja konseptin todennuksen pohjalta voidaan hyödyntää automaatiota tehokkaasti eri tavoin Intermarketing Oy:n ohjelmistotestauksessa. Jatkokehitysideoita ja tarkemman tutkimuksen kohteita ovat kootusti esimerkiksi seuraavat:

- konseptin todennuksen kehitys ja laajennus koko sovelluksen regressiotestaukseen
- puoliautomaattisten testien manuaalisissa toiminnoissa käytettävän aikaviiveen korvaaminen tauko-dialogeilla (*Pause Execution* Robot Frameworkin *Dialogs*-avainsanakirjastossa), joissa voisi olla myös manuaalisen toiminnon suoritusohjeet
- valmiiden avainsanakirjastojen ja työkalujen tutkiminen yrityksen muiden monimutkaisempien sovellusten varalta

- omien generisten avainsanakirjastojen ja avainsanojen suunnittelu ja toteutus
- testitapausten ja ylläpidettävien avainsanaohjattujen testien suunnittelu ja toteutus kaikkiin yrityksen ohjelmisto- ja laiteratkaisuihin
- testauksen hallintatyökalujen tutkiminen ja käyttöönotto
- oheistoiminnan, kuten sovellusten asennusten ja testiympäristöjen asiakaskohtaisten konfigurointien automatisointi
- automaation hyödyntämisen tutkiminen tutkivan testauksen hallinnoinnissa.

Tarkemman, kattavamman ja yleisesti paremman testauksen avulla saavutetaan lopulta jopa hyvin kustannustehokkaasti parempi ohjelmistojen laatu, joka taas vaikuttaa suoraan myös asiakastytyvyyteen ja ohjelmistoratkaisujen maineeseen.

6 Yhteenveto

Tämän työn tavoitteena oli tutkia automaation hyödyntämistä testauksessa. Erityisesti tavoitteena oli tutkia ja etsiä ratkaisuja työn toimeksiantajan ohjelmistotestauksen tiettyihin haasteisiin. Tärkeimpänä ratkaistavana haasteena oli usein samalla tavalla toistettavan manuaalisen testauksen vähentäminen ja automaation hyödyntäminen ohjelmisto- ja laiteympäristössä, jossa on käytössä fyysisiä laitteita, joiden testausta ei edes voida automatisoida kokonaan. Lähtötilanne oli sellainen, että järjestelmätestauksen kaikki vaiheet jouduttiin suorittamaan aina manuaalisesti, vaikka testauksessa on useita aina samalla tavalla toistettavia vaiheita, jotka tietokone voisi suorittaa nopeammin ja tarkemmin.

Ratkaisua lähdettiin lähestymään tutkimalla ensin ohjelmistotestausta ja siihen liittyviä erilaisia näkökantoja ja lähestymistapoja yleisesti. Tavoitteena tässä osiossa oli saada tietoa erilaisien lähestymistapojen mahdollisista eduista ja haitoista sekä mahdollisesta soveltuvuudesta tämän työn tavoitteen saavuttamiseksi. Tutkituista ajatusmalleista kontekstiohjattu testauksen koulukunta osoittautui työn kannalta erittäin hyväksi suuntaa antavaksi näkökulmaksi testaukseen ja automatisoinnin hyödyntämiseen.

Tämän jälkeen keskityttiin tutkimaan testiautomaatiota. Tässä osiossa perehdyttiin automatisoinnin yleisiin ongelmiin, olettamuksiin ja mahdollisuuksiin. Tavoitteena oli tutkia

automatisointia insinööriyön tavoitteiden kannalta ja tutkimuksen kautta yrittää välttää automatisoinnissa usein tehtäviä virheitä. Työn tavoitteiden kannalta kaikista tärkeintä oli saada automatisoitua käyttöliittymän läpi tehtävää järjestelmätestausta, joten tutkimuksessa keskityttiin lähinnä testien automaattiseen suorittamiseen. Tutkimuksen jälkeen oli selvää, että täyteen automaatioon ei kannata pyrkiä ja työn kannalta automaattisten testien hyödyllisin käyttötarkoitus on olla tukena manuaaliselle tutkivalle testaukselle.

Seuraavaksi tutustuttiin skriptaukseen ja erilaisiin testiskriptaustekniikoihin. Testiskriptauksen erilaisista lähestymistavoista ja tekniikoista käytiin läpi kunkin edut ja ongelmat, sekä soveltuvuus tämän työ kannalta. Menetelmistä valikoitui lopulta selkeästi kehittynein ja potentiaalisin vaihtoehto eli avainsanaohjattu testaus. Kyseinen menetelmä mahdollistaa tarvittaessa hyvinkin laajamittaisen ja ylläpidettävän testiautomaattioratkaisun. Seuraavaksi esiteltiin automatisoinnissa käytettävä työkalu, Robot Framework -testiautomaatiokehys. Robot Framework valittiin, koska se soveltuu erinomaisesti aiemmin avainsanaohjatun testauksen toteuttamiseen.

Skriptausmenetelmän ja testiautomaatiokehyyksen valinnan jälkeen siirryttiin konseptin todennus -vaiheeseen. Aluksi esiteltiin ohjelmisto- ja laiteratkaisu, johon konseptin todennus oli tarkoitus toteuttaa. Kyseessä oli setelinlaskin/lajittelija, jota käytetään Java-sovelluksella, jossa on Swing-käyttöliittymä. Järjestelmällä suoritetaan laskentaeriä, joiden tulokset tallennetaan XML-muotoisina lokitiedostoina. Konseptin todennuksen määritelmäosuudessa päätettiin luoda yhdelle testitapaukselle, normaalille onnistuneelle laskentaerälle, kaksi automaatiota käyttävää testiä. Toinen testeistä tuli olla täysin automaattisesti suoritettava simulaattoria vasten ajettava käyttöliittymätesti ja toinen mahdollisimman automaattinen käyttöliittymän läpi tehtävä järjestelmätesti, joka sisältäisi vain yhden pakollisen manuaalisen osuuden. Manuaalinen toimenpide oli laskettavien seteleiden asettaminen laitteeseen.

Seuraavaksi tehty Robot Frameworkin asennus ja käyttöönotto sujuivat erittäin helposti, ja tämän jälkeen päästiin suunnittelemaan ja toteuttamaan varsinaisia testejä. Konseptin todennuksen tavoite saavutettiin lopulta tekemällä yksi testi, jota voidaan käyttää molemmassa määrittelyn testivariaatioissa. Puoliautomaattinen testi poikkeaa automaattisesta vain siten, että siinä käytetään pakollisen manuaalisen toiminnon kohdalla pidempää aikaviivettä kuin simulaattoria vasten ajettaessa. Aikaviiveen käyttö oli yksinkertainen ja tehokas ratkaisu manuaalisen osuuden hoitamiseksi. Konseptin todennuk-

sessä luotuja avainsanaohjattuja testejä voidaan myös hyödyntää, kun jatkossa toteutetaan uusia testejä kyseiselle sovellukselle. Melko vähällä vaivalla kyseisiä testejä voidaan käyttää myös useissa muissakin yrityksen ohjelmisto- ja laiteratkaisuissa. Konseptin todennuksen lopuksi pohdittiin vielä erilaisia jatkokehitysideoita, joista olisi toteutuessaan huomattavasti hyötyä työn toimeksiantajalle.

Kaiken kaikkiaan insinööriötä voidaan pitää onnistuneena. Tutkimusten perusteella päädyttiin onnistuneeseen ja hyödylliseen testiautomaatiokokeiluun. Tutkimuksen ja konseptin todennuksen pohjalta työn toimeksiantaja sai paljon tärkeää tietoa testauksesta sekä testiautomaation haasteista ja mahdollisuuksista. Työn toimeksiantajan ohjelmistokehityksessä tullaan jatkossa ottamaan testiautomaatiota käyttöön tämän insinööriöityön pohjalta. Työn tutkimusosuudessa mainittuja automaatiomenetelmiä tullaan tutkimaan tarkemmin ja konseptin todennuksen testiautomaatiokokeilua sekä jatkokehitysideoita ryhdytään tutkimaan ja hyödyntämään yrityksen ohjelmistokehityksessä.

Lähteet

- 1 Konttinen, Valtteri ja Koivisto, Juha-Pekka. Ohjelmisto- ja tuotekehitys. Intermarketing Oy, Espoo. Haastattelut 9.9.2013.
- 2 Vesiputousmalli. Wikipedia. Verkkodokumentti. <<http://fi.wikipedia.org/wiki/Vesiputousmalli>>. Linkki varmistettu 25.11.2013.
- 3 Ketterä ohjelmistokehitys. Wikipedia. Verkkodokumentti. <http://fi.wikipedia.org/wiki/Ketter%C3%A4_kehitys>. Linkki varmistettu 25.11.2013.
- 4 Crispin, Lisa ja Gregory, Janet. 2009. Agile testing: a practical guide for testers and agile team. Crawfordsville, Indiana: Addison-Wesley Professional. 9. painos, 2013.
- 5 Bach, James. Schools of testing... here to stay. Blogi. <<http://www.satisfice.com/blog/archives/134>>. Linkki varmistettu 25.11.2013.
- 6 Bach, James. The Dual Nature of Context-Driven Testing. Blogi. <<http://www.satisfice.com/blog/archives/565>>. Linkki varmistettu 25.11.2013.
- 7 Kaner, Cem, James Bach ja Bret Pettichord . 2002. Lessons learned in software testing: a context-driven approach. New York: John Wiley & Sons, Inc.
- 8 Jorgensen, Paul. 2008. Software testing: a craftsman's approach. Boca Raton, Florida: Auerbach Publications. 3.painos.
- 9 The Seven Basic Principles of the Context-Driven School. Verkkodokumentti.. <<http://context-driven-testing.com/>>. Linkki varmistettu 25.11.2013.
- 10 Bach, James. Publications. Verkkodokumentti. <<http://www.satisfice.com/>>. Linkki varmistettu 25.11.2013.
- 11 Bolton, Michael. Publications. Verkkodokumentti. <<http://www.developsense.com/publications.html> >. Linkki varmistettu 25.11.2013.
- 12 Rapid Software Testing. Verkkodokumentti. <http://www.satisfice.com/info_rst.shtml>. Linkki varmistettu 25.11.2013.
- 13 Session-Based Test Management. Verkkodokumentti. <<http://www.satisfice.com/sbtm/>>. Linkki varmistettu 25.11.2013.

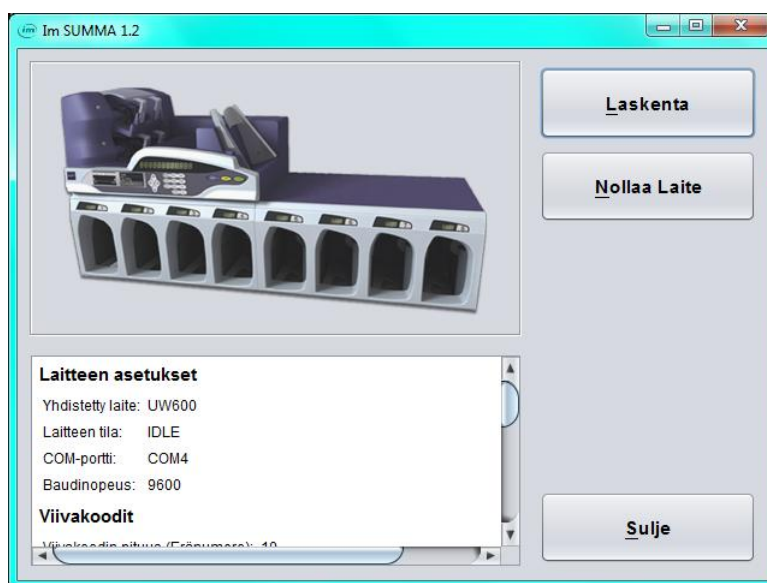
- 14 ISTQB: Downloads. Verkkosivusto. <<http://www.istqb.org/downloads.html>>. Linkki varmistettu 25.11.2013.
- 15 Bach, James. Test automation snake oil. Verkkodokumentti. <http://www.satisfice.com/articles/test_automation_snake_oil.pdf>. Linkki varmistettu 25.11.2013.
- 16 Laatu ja testaus 2/2012: Automatisointi. Verkkodokumentti. <<http://testausosy.fi/wp-content/uploads/2012/11/LT-Vol1Ed2.pdf>>. Linkki varmistettu 25.11.2013.
- 17 Vuori, Matti. 2013. Noin 80 ajatusta testiautomaatiosta. Verkkodokumentti. <http://testausosy.fi/wp-content/uploads/2013/06/noin_80_ajatusta_testiautomaatiosta.pdf>. Linkki varmistettu 25.11.2013.
- 18 Pyhäjärvi, Maaret ja Pöyhönen, Erkki. Testauskirja: Testauksen automatisointi - materiaalit. Verkkodokumentti. <<http://www.testauskirja.com/materiaalit.htm>>. Linkki varmistettu 25.11.2013.
- 19 Elfriede, Dustin. 2003. Effective software testing: 50 specific ways to improve your testing. Boston, Massachusetts: Addison-Wesley Professional. 1.painos.
- 20 Kasurinen, Jussi Pekka. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo. 1.painos.
- 21 Farrel.Vinay, Peter. 2008. Manage software testing. Boca Raton, Florida: Auerbach Publications. 1.painos.
- 22 Bach, James. Testing and checking refined. Blogi. <<http://www.satisfice.com/blog/archives/856>>. Linkki varmistettu 25.11.2013.
- 23 Bach, James. Agile Test Automation. Verkkodokumentti. <<http://www.satisfice.com/presentations/agileauto.pdf>>. Linkki varmistettu 25.11.2013.
- 24 Komentosarjakieli. Verkkodokumentti. Wikipedia. <<http://fi.wikipedia.org/wiki/Komentosarjakieli>>. Linkki varmistettu 25.11.2013.
- 25 Laukkanen, Pekka. Data-Driven and Keyword-Driven Test Automation Frameworks. Diplomityö. <<http://eliga.fi/Thesis-Pekka-Laukkanen.pdf>>. Linkki varmistettu 25.11.2013.
- 26 Klärck, Pekka. Introduction to Test Automation. Verkkodokumentti. <<http://www.slideshare.net/pekkaklarck/introduction-to-test-automation>>. Linkki varmistettu 25.11.2013.

- 27 Puolitaival, Olli-Pekka. Model-based testing tools. Verkkodokumentti.
<<http://www.cs.tut.fi/tapahtumat/testaus08/Olli-Pekka.pdf>>. Linkki varmistettu 25.11.2013.
- 28 Robot Framework User Guide, Version 2.8.1. Verkkodokumentti.
<<http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html?r=2.8.1>>. Linkki varmistettu 25.11.2013.
- 29 FitNesse: An Example FitNesse Test. Verkkodokumentti.
<<http://fitnesse.org/FitNesse.UserGuide.TwoMinuteExample>>. Linkki varmistettu 25.11.2013.
- 30 Robot Framework Installation and uninstallation instructions. Verkkodokumentti.
<<https://code.google.com/p/robotframework/wiki/Installation>>. Linkki varmistettu 25.11.2013.

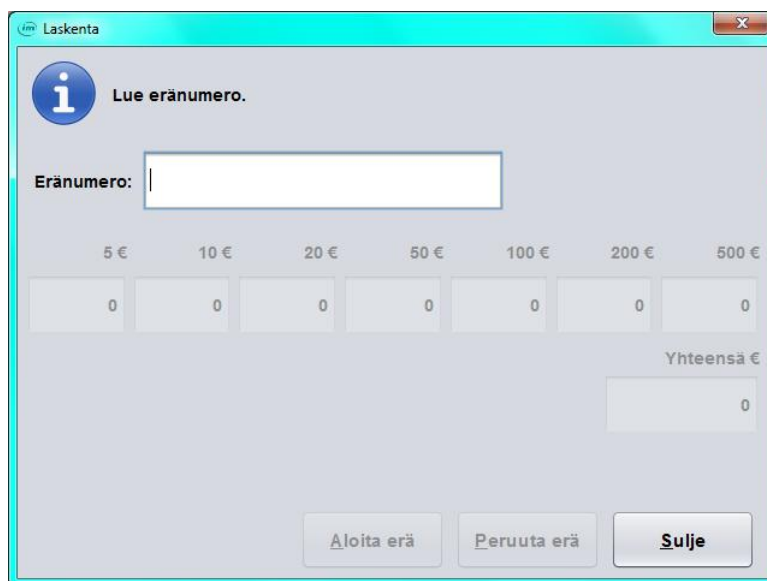
Laskentaerän suoritus

Tässä liitteessä on kuvattu onnistuneen laskentaerän suorituksen vaiheet ruutukaappauksin. Numeroituja toimintakuvauksia seuraa toimintaa seuraava tilanne ruutukaappauksena.

1. Käynnistetään sovellus.



2. Painetaan Laskenta-painiketta.



3. Syötetään oikean pituinen eränumero.

Laskenta

i Eränumero syötetty.

Eränumero: 1234567890

5 €	10 €	20 €	50 €	100 €	200 €	500 €
0	0	0	0	0	0	0

Yhteensä €

0

Aloita erä Peruuta erä Sulje

4. Painetaan Aloita erä -painiketta.

Laskenta

⋮ Aloitetaan erä '1234567890'...

Eränumero: 1234567890

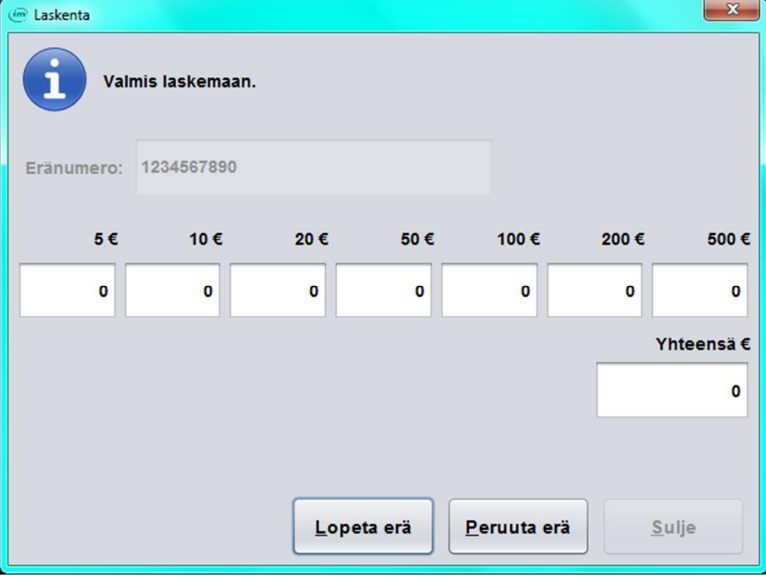
5 €	10 €	20 €	50 €	100 €	200 €	500 €
0	0	0	0	0	0	0

Yhteensä €

0

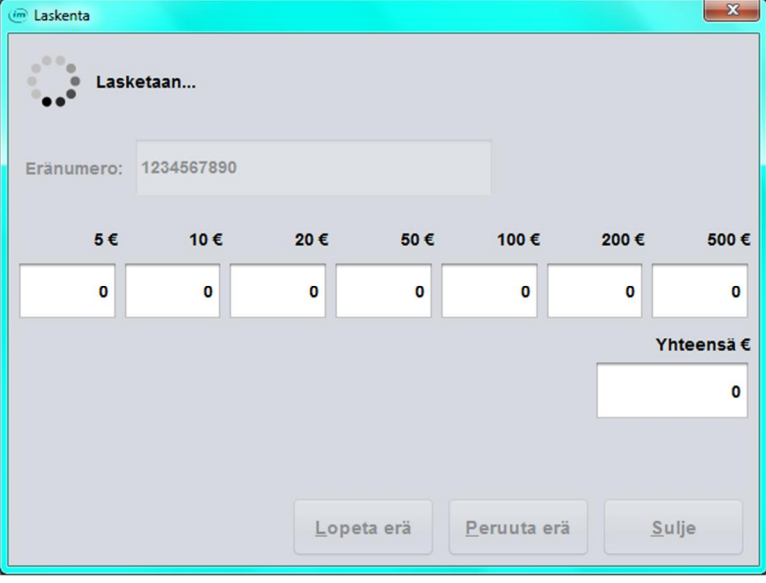
Aloita erä Peruuta erä Sulje

5. Odotetaan hetki erän aloittamista.



The screenshot shows a window titled "Laskenta" with a status bar at the top. The main content area features a blue information icon and the text "Valmis laskemaan." Below this is a text input field labeled "Eränumero:" containing the value "1234567890". Underneath are seven buttons representing denominations: "5 €", "10 €", "20 €", "50 €", "100 €", "200 €", and "500 €". Each button has a corresponding input field below it, all containing the number "0". To the right of these fields is a label "Yhteensä €" and a larger input field containing "0". At the bottom of the window are three buttons: "Lopeta erä", "Peruuta erä", and "Sulje".

6. Asetetaan setelilajitelma laitteeseen.



The screenshot shows the same "Laskenta" window, but the status has changed to "Lasketaan...". A loading spinner icon is now visible at the top left. The "Eränumero:" field still contains "1234567890". The denomination buttons and their input fields remain, with all values still at "0". The "Yhteensä €" field also remains at "0". The buttons "Lopeta erä", "Peruuta erä", and "Sulje" are still present at the bottom.

7. Odotetaan hetki laskennan päättymistä.

The screenshot shows a window titled "Laskenta" with a status bar at the top. On the left, there is an information icon (i) and the text "Valmis laskemaan.". Below this, a text field labeled "Eränumero:" contains the value "1234567890".

5 €	10 €	20 €	50 €	100 €	200 €	500 €
12	5	10	5	0	0	0

Yhteensä €

560

At the bottom, there are three buttons: "Lopeta erä", "Peruuta erä", and "Sulje".

8. Painetaan Lopeta erä -painiketta

The screenshot shows the same "Laskenta" window, but now with a loading spinner icon and the text "Lopetetaan erä '1234567890'...". The "Eränumero:" field still contains "1234567890".


5 €	10 €	20 €	50 €	100 €	200 €	500 €
12	5	10	5	0	0	0

Yhteensä €

560

The buttons "Lopeta erä", "Peruuta erä", and "Sulje" are still present at the bottom.

9. Odotetaan hetki erän päättämistä.



Laskenta

i Lue eränumero.

Eränumero:

5 € 10 € 20 € 50 € 100 € 200 € 500 €

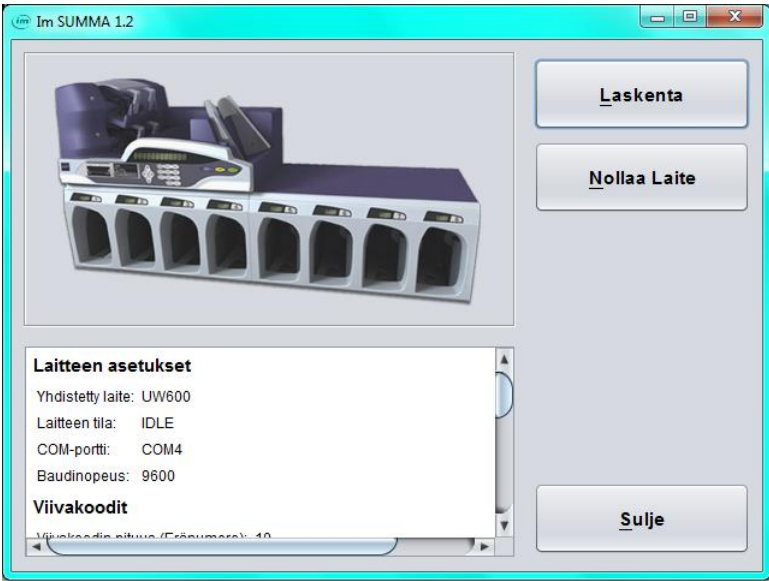
0 0 0 0 0 0 0

Yhteensä €

0

Aloita erä Peruuta erä Sulje

10. Painetaan Sulje-painiketta



Im SUMMA 1.2

Laitteen asetukset

Yhdistetty laite: UW600
Laitteen tila: IDLE
COM-portti: COM4
Baudinopeus: 9600

Viivakoodit

Viivakoodin pituus (Eränumero): 10

Laskenta

Nollaa Laite

Sulje

Konseptin todennuksen avainsanat

Tässä liitteessä on kuvattu konseptin todennuksessa luodun avainsana-resurssitiedoston (*transaction.txt*) sisältö kokonaisuudessaan.

```

*** Settings ***
Library XML
Library Collections
Library fi.intermarketing.robot.MyKeywordLibrary

*** Variables ***
${COUNTING_DELAY}          10
${RESULT_LOG_DIRECTORY}    transactions
${RESULT_LOG_FILETYPE}     xml
${VALID_TRANSACTION_ID}    1234567890
${INVALID_TRANSACTION_ID}  1234567

# Expected denomination
${5EUR}      12
${10EUR}     5
${20EUR}     10
${50EUR}     5
${100EUR}    0
${200EUR}    0
${500EUR}    0

*** Keywords ***
Open Transaction View
    Push Button          cmdCashCount
    Select Dialog        Laskenta
    Button Should Be Disabled  btnStart

Close Transaction View
    Push Button          btnClose
    Select Main Window

Set Valid Transaction Id
    Insert Into Text Field  txtBatchNumber  ${VALID_TRANSACTION_ID}

Set Invalid Transaction Id
    Insert Into Text Field  txtBatchNumber  ${INVALID_TRANSACTION_ID}

Start Transaction
    Button Should Be Enabled  btnStart
    Push Button              btnStart

End Transaction
    Button Should Be Disabled  btnClose
    Push Button              btnStop

Wait For Money Counting To Finish
    Sleep                    ${COUNTING_DELAY}

Verify Transaction Result From Log File
    ${EXPECTED_DENOMINATION}= Create Dictionary  5=${5EUR}      10=${10EUR}
    ...                               20=${20EUR}    50=${50EUR}
    ...                               100=${100EUR}  200=${200EUR}
    ...                               500=${500EUR}
    ${RESULT_LOG_FILE}= Get Last Modified File Path  ${RESULT_LOG_DIRECTORY}  ${RESULT_LOG_FILETYPE}
    Transaction Result Should Be Correct  ${RESULT_LOG_FILE}  ${EXPECTED_DENOMINATION}

Transaction Result Should Be Correct
    [Arguments]          ${RESULT_LOG_FILE}  ${EXPECTED_DENOMINATION}
    ${root}= Parse XML  ${RESULT_LOG_FILE}
    @${values}= Get Elements Texts  ${root}  mix/notes/note/value
    @${quantities}= Get Elements Texts  ${root}  mix/notes/note/quantity
    ${length}= Get Length  ${values}
    :FOR ${i} IN RANGE  ${length}
        \ Log i: ${i}
        \ ${value} = Get From List  ${values}  ${i}
        \ ${quantity} = Get From List  ${quantities}  ${i}
        \ ${expected_quantity} = Get From Dictionary  ${EXPECTED_DENOMINATION}  ${value}
        \ Should Be Equal  ${expected_quantity}  ${quantity}

```